第 26 章　JSP＋Struts+Hibernate开发实例

以上章节详细介绍了 Struts 和 Hibernate 技术，这一章将综合使用这两个当今最为流行的 Web 开发技术，在第 24 章创建的实例基础上，来重新实现用户登录系统，并且增加购物车功能，Struts 提供了很好的 MVC 设计模式的支持；Hibernate 是很好的数据层技术，可以实现完全的面向对象数据库设计。接下来就向读者展示这两种技术完美结合。

本章要点包括以下内容：

- 列举实例介绍
- Struts＋Hibernate 技术的结合使用
- Hibernate 的自动生成工具使用

26.1　实例介绍

26.1.1　实例所实现的功能

除了第 24 章中实例所展示的功能之外，将增加如下功能：

（1）当用户登录后，在首页中将显示“产品展示”链接和“我的购物车”链接，如图 26.1 所示。具体如何实现，请查看下面将要创建的 index.jsp 页面代码。

图 26.1　首页效果

（2）单击“产品展示”链接，出现如图 26.2 所示的页面效果，此页面显示出所有产品信息。并可以单击“加入购物车”链接把此产品加入到购物车。具体实现将在后面讲解。

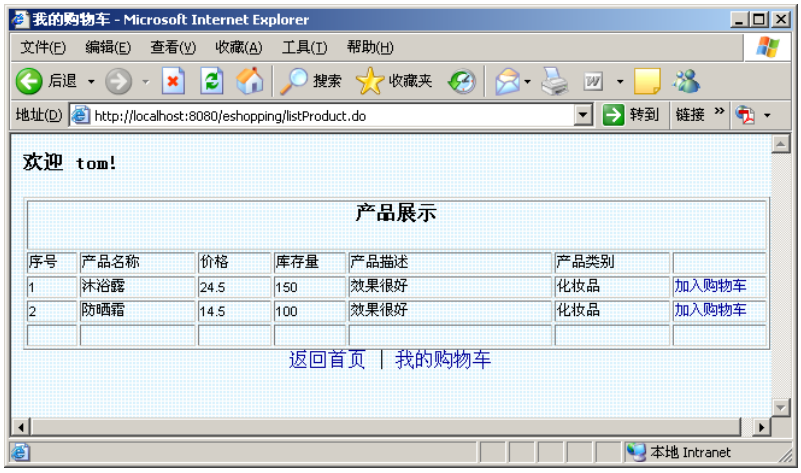

图 26.2　产品展示

（3）在首页中单击“我的购物车”链接可以查看到已经购买到的产品信息，包括此产品的购买数量。在如图 26.3 所示的页面中，单击“删除”链接可以把此产品从购物车中消除，。此处有一个不完善的地方，当顾客想减少产品数量，而不是删除时，将无法操作。如果读者有兴趣，可以自行实现。其实非常简单，只要实现一个操作数据库表 user_product 中存放购买数量的方法即可。

图 26.3　我的购物车

26.1.2　文件的结构

在二十四章创建的实例基础，需要创建和修改的文件如下：

（1）修改 User.java 接口类和 DBUser.java 类文件：在这两个类中增加一个 List 集合类型的 products 属性，用于存储用户购买的所有产品。需要注意的是，DBUser 为持久层的实体类，User 只是 DBUser 类的一个接口。这样的设计模式只是为了后期代码扩展。

（2）创建 Product.java 接口类和 DBProduct.java 类文件：需要注意的是，此类中也定义了一个 List 集合类型的 users 属性，用于存储购买此类产品的所有用户信息，所以 DBUser 和 DBProduct 实体类之间是多对多的关系。另外还定义了一个 Kind 类型的属性，该属性记录该产品所属的类型。

（3）创建 Kind.java 接口类和 DBKind.java 类文件：该类为产品类型的实体类。DBProduct 对 DBKind 实体类是多对一的关系，即多个产品可以为一种类型，但是不允许一个商品同时属于多个类型。

（4）修改 Struts 的配置文件 struts-config.xml，详细见下面的 struts-config.xml 文件。

（5）在 MyLogin 项目的 j2src\cn\login 目录下创建 hibernate 的映射文件 model.hbm.xml，详细见本章下面内容。

（6）根据创建的实体类以及映射文件显示的实体之间的关系来创建相应的数据库表，读者除了手动建立数据库表之外还可以使用 net.sf.hibernate.tool.hbm2ddl.SchemaExportTask 工具自动生成相应的数据库表。在此实例中，将手动创建如下数据库表：p_kind、product、user_product 以及 users（此表在第十二章介绍的实例中已经创建，此处直接使用，不需要修改）。user_product 为 users 和 product 之间的关系表，记录着它们之间的多对多的关系。

（7）创建 AbstractProduct.java 抽象类和 SqlProduct.java 类文件：该类定义了对产品实体类进行数据库操作的各种方法，AbstractProduct.java 为抽象类，实现了 listProduct()和 listProducts()两个方法。这里 SqlProduct 只是空实现，随着功能的增加，将再对 SqlProduct 进行扩展。

（8）修改第十四章创建的 AbstractUser.java 抽象类和 SqlUser.java 类文件，这两个类主要是使用 Hibernate 进行数据层操作。具体详见这两个文件的源代码。

（9）创建 ShopCarForm.java 类和 ShopCarAction.java 类。

（10）创建相应的页面 JSP 文件：myShopCar.jsp 用于显示购物车中的信息；listProduct.jsp 显示出所有的产品信息。另外修改 index.jsp 页面，在<logic:present name="user">体中添加“产品展示”和“我的购物车”两个链接。

（11）另外，还需要使用到上一章已经创建的 HibernateUtil.java 类文件。

26.2　Struts＋Hibernate开发

有了以上的介绍之后，下面正式进入开发阶段。

26.2.1　创建实体类

需要创建的实体类依次如下：

26.2.1.1．修改用户实体类 DBUser 和接口 User

在第 24 四章实例中创建的 DBUser 添加一个 List 集合类型的属性 products 变量，用于记录所有的购买的商品信息。添加代码如下：

```
private List products = new ArrayList();
```

相应的 set 和 get 方法如下：

```
public void setProducts(List products){
    this.products = products;
}
public List getProducts(){
    return products;
}
```

在这个实例当中还继续使用了接口继承的编程思路，对以上的 set 和 get 方法在 User 接口中进行定义。如下所示：

```
public void setProducts(List products);
public List getProducts();
```

读者一定要注意的是，DBUser 被用作为实体类，而不是 User 接口类作为实体类。

26.2.1.2．创建商品实体类 DBProduct 和 Product 接口

在包 cn.login.model 下创建商品接口类 Product，代码如下：

```
package cn.login.model;
import java.util.List;
public interface Product {

    /*****只定义各属性对应的 set 和 get 方法，但不实现*****/
    public void setId(int id);
    public int getId();
    public void setName(String name);
    public String getName();
    public void setPrice(float price);
    public float getPrice();
    public void setStocks(int stocks);
    public int getStocks();
    public void setKind(Kind kind);
    public Kind getKind();
    public void setDiscription(String discription);
    public String getDiscription();
    public void setUsers(List users);
    public List getUsers();
}
```

程序说明：DBProduct 实体类（包名为 cn.login.model）继承这个接口类，并定义各个相应属性。注意 List 集合类型的 users 属性是用来存储购买这种产品的用户，所以这里商品实体类和用户实体类之间是多对多的关系。

26.2.1.3．创建商品种类实体类 DBKind 和 Kind 接口

在 cn.login.model 包下创建 Kind 接口类，详细代码代码如下：

```
package cn.login.model;
public interface Kind {
    /*****只定义各属性对应的 set 和 get 方法，但不实现*****/
    public void setId(int id);
    public int getId();
    public void setName(String name);
    public String getName();
    public void setDiscription(String discription);
    public String getDiscription();
}
```

程序说明：SqlKind 实体类（包名为：cn.login.model）继承这个接口类，并定义相应属性，此实体类用于记录产品的种类信息。商品实体类 DBProduct 和种类实体类 DBKind 之间是多对一的关系。

三个实体类 DBuser、DBProduct 和 DBKind 之间的关系如下：

- 用户实体类 DBuser 和商品实体类 DBProduct 之间是多对多的关系，即一名用户可以购买多个商品，一个商品也可以由多个用户购买，所在这两个实体类中都定义一个 List 类型的属性，用于存储对方信息。对应数据库表，需要在这两个之间创建一个 user_product 表来表示它们之间的关系，以及存储彼此的信息。
- 商品实体类 DBProduct 和种类实体类 DBKind 之间是多对一的关系，即多个商品可以同属于一种商品类别，但是一个商品只能属于一种类别。在 DBProduct 实体类中定义了一个 Kind 类属性，

来表示此商品所属的商品类别。

下面根据需要实现的功能以及三个实体类之间的关系，来配置相应的映射文件 model.hbm.xml 以及 Struts 配置文件 struts-cofig.xml。

26.2.2　创建XML配置文件

26.2.2.1　创建映射文件 model.hbm.xml

在 MyLogin 项目中的 j2src\cn\login 目录下创建 Hibernate 的实体类到数据库表的映射文件 model.hbm.xml，根据前一章的讲解，这是 Hibernate 中最为重要的一步。它不仅表示了实体之间的关系，还定义实体类如何向数据库表进行映射，也就是说这一文件告诉了 Hibernate 应该如何进行数据层的操作。详细的代码内容如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
      <!--  关于实体类 SqlUser 的映射关系  -->
     <class name="cn.login.model.DB User" table="users">
          <id name="user_id" unsaved-value="0">
               <generator class=" assigned "/>
          </id>
          <property name="password"/>
          <property name="name"/>
          <property name="sex"/>
          <property name=”birth”/>
          <property name="descrption"/>
          <list name="products" table="user_product" lazy=”true”>
              <key column="user_id"/>
              <index column="position"/>
              <many-to-many class="cn.login.model.DBProduct" column="product_id"/>
          </list>
     </class>
     <!--  关于实体类 SqlKind 的映射关系  -->
     <class name="cn.login.model.DBKind" table="p_kind">
         <id name="id" unsaved-value="0">
              <generator class="increment"/>
         </id>
         <property name="name"/>
         <property name="discription"/>
     </class>
     <!--  关于实体类 SqlProduct 的映射关系  -->
     <class name="cn.login.model.DBProduct" table="product">
         <id name="id" unsaved-value="0">
              <generator class="increment"/>
         </id>
         <property name="name"/>
```

```
        <property name="price"/>
        <property name="stocks"/>
        <property name="discription"/>
        <many-to-one name="kind" column="kind_id" class="cn.login.model.SqlKind" cascade="save-update"/>
    </class>
</hibernate-mapping>
```

代码说明：

（1）代码中定义的<generator class="increment"/>表示关键字段 id 自动增加，对应的 SQL 语句，就是使用 Max 函数把数据库表的最大 id 值取出，然后加一在赋值给新的 id。关于 generator 的属性在上一章已经有详细介绍。

（2）根据口诀“类的字段有则映射有、类的字段无则映射无”，所以在 DBProduct 实体类的映射配置中定义了<many-to-one>关系。

- name=”kind”对应 DBProduct 实体类内中的属性名
- column="kind_id"对应数据库表 product 中的字段 kind_id，通过这个外部关键字来与数据库表 p_kind 进行关联。
- class="cn.login.model.DBKind"定义了商品类别的实体类 DBKind。
- cascade="save-update"属性设置表示 Hibernate 会进行自动关联存储，例如，存储实体类 DBProduct 时，会把 DBProduct 指定的 DBKind 实体类也一便存储。

（3）在 DBUser 实体类映射配置中定义了 List 集合映射，此处没有使用 Set 集合类，是因为 Set 集合不能存储重复的对象，而一个用户很有可能会购买多个同一商品，所以这里使用 List 集合类，并进行 List 集合映射定义。读者需要注意的一点，当使用 List 集合映射时，就不可以再定义 cascade="save-update" 属性。如配置文件所示。

（4）在<list>中设置了 lazy=”true”，表示使用延迟，这是 Hibernate 中非常重要的概念，主要使用在多对多以及一对多关系中。具体在实例程序中讲解。

（5）此处定义了用户和商品之间单向的多对多关系。

- <key column="user_id"/>定义了数据库表 user_product 中的外部关键字 user_id 与 users 表进行关联。
- 由于这里使用的 List 集合映射，所以还要指定<index column="position"/>来表示实例在 List 中的位置，在创建 user_product 表时，需要创建一个 position 字段。
- <many-to-many>中的 class 属性指定 SqlProduct 实体类，并且定义 column="product_id"，即使用 product_id 外键来与 product 数据库表相关联。

26.2.2.2　修改 Struts 配置文件 struts.config.xml

首先在<form-beans>体中添加一个 ActionForm 类，代码如下：

```
<form-bean name="shopCarForm" type="cn.login.ShopCarForm"/>
```

然后在<global-forwards>体中添加如下代码：

```
        <forward
            name="listProduct"
            path="/listProduct.do"/>
        <forward
            name="myShopCar"
            path="/myShopCar.do"/>
        <forward
            name="index"
            path="/index.do"/>
```

在<action-mappings>体中添加的代码如下：

```
<action
        path="/listProduct"
        type="org.apache.struts.actions.ForwardAction"
        parameter="/listProduct.jsp"
        scope="request">
</action>
<action
        path="/myShopCar"
        type="org.apache.struts.actions.ForwardAction"
        parameter="/myShopCar.jsp"
        scope="request">
</action>
<action
        path="/shopCarAction"
        type="cn.login.ShopCarAction"
        parameter="method"
        name="shopCarForm"
        validate="true"
        scope="request"
        input="/listProduct.jsp">
        <forward name="return_myShopCar" path="/myShopCar.jsp"/>
</action>
<action
        path="/index"
        type="org.apache.struts.actions.ForwardAction"
        parameter="/index.jsp"
        scope="request">
</action>
```

代码说明：

此处创建的 ShopCarAction 类是一个 DispatchAction 类，parameter="method"的设置表示接收来自页面的 method 属性，通过这个属性值来指定这个 DispatchAction 类调用其中某个方法（例如添加、删除或者修改等）。

26.2.3　创建数据库表

除了使用到以上章节创建的 users 表之后，根据以上创建的实体类，以及实体类之间的关系，创建如下三个数据库表。

26.2.3.1　创建 p_kind 表

在以上章节创建的数据库 mydb1 中新建一个 p_kind 表，用户存放商品类别的信息，SQL 语句如下：

```
DROP TABLE IF EXISTS `mydb1`.`p_kind`;
CREATE TABLE   `mydb1`.`p_kind` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(45) NOT NULL default '',
  `discription` varchar(1024) NOT NULL default '',
  PRIMARY KEY   (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

主关键字为 id，name 为类别名称，discription 字段为信息描述。

26.2.3.2．创建 product 表

用于存储商品信息的 product 表，相应的 SQL 语句如下：

```
DROP TABLE IF EXISTS `mydb1`.`product`;
CREATE TABLE   `mydb1`.`product` (
  `id` int(10) unsigned NOT NULL auto_increment,
  `name` varchar(45) NOT NULL default '',
  `price` float NOT NULL default '0',
  `stocks` int(10) unsigned NOT NULL default '0',
  `discription` varchar(1024) NOT NULL default '',
  `kind_id` int(10) unsigned NOT NULL default '0',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

主键为“id”，通过外部关键字“kind_id”与数据库表 p_kind 相关联。

26.2.3.3．创建 user_product 表

此表为 userinfo 和 product 表之间的关联表，具体创建的 SQL 语句如下：

```
DROP TABLE IF EXISTS `mydb1`.`user_product`;
CREATE TABLE   `mydb1`.`user_product` (
  `user_id` int(10) unsigned NOT NULL default '0',
  `product_id` int(10) unsigned NOT NULL default '0',
  `id` int(10) unsigned NOT NULL auto_increment,
  `position` int(10) unsigned NOT NULL default '0',
  PRIMARY KEY  (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=gbk;
```

读者也可以使用工具，根据 XML 映射文件自动生成如上的数据库表。这样可以简化创建表格的工作，此处选择手动创建，只是为了让读者加深了解 Hibernate 是如何把实体类映射到数据库表。

26.2.4　创建相应的操作类

有读者可能认为以下的有些操作类是不需要创建的，完全可以把这部分工作移交到 Action 类中实现，这确实是可以的。但是笔者认为在开发应用系统时，尽量在设计上把功能细分，并且更多地使用继承开发模式，这样有利用后期代码的扩展和维护，而且不至于使 Action 类代码过于冗长。

26.2.4.1．创建操作商品的类 AbstractProduct.java 和 SqlProduct.java

这里继续使用继承的思想，首先在包 cn.login.model 下创建 AbstractProduct.java 抽象类：

```
package cn.login.model;
import java.util.List;
import org.hibernate.HibernateException;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import cn.login.db.HibernateUtil;
import cn.login.model.DBProduct;
public abstract class AbstractProduct {

    /*****取出单个产品信息（包括产品类型）******/
```

```
    public List listProduct(int id) throws HibernateException
    {
        Product product = new DBProduct();
        Session session = HibernateUtil.currentSession();    //获取到 Session
        Transaction tx = session.beginTransaction();         //创建一个事务
        Query query = session.createQuery("select p,p.kind.name from DBProduct as p where p.id=:id");
        query.setInteger("id",id);                           //给 HQL 语句的 id 字段赋值，实现条件查询
        List list = query.list();
        tx.commit();                                         //事务批量执行
        HibernateUtil.closeSession();                        //关闭 session
        return list;
    }
    /*****取出所有产品信息（包括产品类型）*****/
    public List listProducts() throws HibernateException
    {
        Session session = HibernateUtil.currentSession();
        Transaction tx = session.beginTransaction();
        Query query = session.createQuery("select p,p.kind.name from SqlProduct as p");
        List list = query.list();
        tx.commit();
        HibernateUtil.closeSession();
        return list;
    }
}
```

程序说明：

- listProduct(int id)方法根据给定的商品 id 取出相对应的实例对象，此处使用到的 HibernateUtil 类已经在上一章创建了，此类专门对 session 进行管理，包括创建和关闭。
- 调用 query.setInteger("id",id)给 HQL 语句中的 id 字段进行赋值，需要赋值的字段前加上冒号，以跟别的字段相区别。
- listProducts()方法把所有关于 DBProduct 实体类的实例对象取出来。

然后再在包 cn.login.model 下创建相应的继承类 SqlProductFactory，但是暂时不定义任何方法，只是用于后期扩展。

26.2.4.2．修改关于 User 的操作类 AbstractUser.java 和 SqlUser.java

这两个类已经在第七章节中创建了，修改这两个类，在抽象类中实现 ListUsers()和 ListUser()两个方法，另外定义四个方法，以实现 Hibernate 的数据库操作，修改代码如下：

```
    /******取出所有用户信息*****/
    public Iterator ListUsers() throws HibernateException{
        Session session = HibernateUtil.currentSession();
        Transaction tx = session.beginTransaction();
        Query query = session.createQuery("from DBUser");
        List list = query.list();
        tx.commit();
        HibernateUtil.closeSession();
        return list.iterator();
    }
    /*****根据用户名取出对应的一名用户信息*****/
    public User ListUser(String userid){
```

```
            User user = null;
            Session session = HibernateUtil.currentSession();
            Transaction tx = session.beginTransaction();
            Query query = session.createQuery("from DBUser as u where u.user_id=:userid");
            query.setParameter("userid",userid);
            List list = query.list();
            if(!list.isEmpty())                    //判断 list 集合类是否为空
                  user = (User)list.get(0);    //取出对象信息，因为存在 List 中的是 Object 对象，所要这里下塑造型
            tx.commit();
            HibernateUtil.closeSession();
            return user;
      }
      /*****根据传递过来的 session 以及用户名取出对应的一名用户信息*****/
      public User ListUser(Session session,String userid){
            User user = null;
            Transaction tx = session.beginTransaction();
            Query query = session.createQuery("from DBUser as u where u.user_id=:userid");
            query.setParameter("userid",userid);
            List list = query.list();
          if(!list.isEmpty())
                  user = (User)list.get(0);
            tx.commit();
            return user;
      }
……
      public abstract List listMyProducts(int id);                        //取出该用户购物车中的所有商品
      public abstract int countProduct(int user_id,int p_id);             //计算该商品购买的数量
      public abstract void buyProduct(User user,Product product);    //把商品加入到购物车中
      public abstract void delProduct(User user,Product product);    //把商品从购物车中删除
```

程序说明：

（1）此抽象类实现的两个方法和 ProductFactory 类中实现的两个方法相类似，程序设计的思想就是父类中首先实现公用方法，然后在继承类中实现专有的方法。

（2）请注意 ListUser(Session session,String userid)方法和 ListUser(String userid)方法的区别，仅仅是前一个方法通过传递一个 Session，而不是在内部生成 Session。因为在 SqlUser 实体类中包含了商品实例的集合类，这里使用了延迟（在 model.hbm.xml 文件中配置了 lazy=”true”），在获取延迟字段的数据之前是不能关闭 Session 的，所以这里通过传递一个 Session 参数获取到一个用户实例，然后在关闭 Session 之前取出这个实例中的各个字段（包括集合类中的产品类）。

在继承类 SqlUser 中实现 AbstractUser 抽象类定义的四个方法，代码如下：

```
……
      public List listMyProducts(int id) throws HibernateException
      {           //取出该用户购物车中的所有商品
            Session session = HibernateUtil.currentSession();
            Transaction tx = session.beginTransaction();
                  //使用特殊的 elements()函数返回所有的 product 实例
            Query query = session.createQuery("select u,p from DBUser as u,DBProduct as p
                        where p in elements(u.products) and u.id=:id");
            query.setInteger("id",id);
            List list = query.list();
```

```
        tx.commit();
        HibernateUtil.closeSession();
        return list;
    }
    public int countProduct(int user_id,int p_id) throws HibernateException
    {                                                   //计算该商品购买的数量
        User user = new DBUser();
        Product product = new DBProduct();
        Session session = HibernateUtil.currentSession();
        Transaction tx = session.beginTransaction();
        Query query = session.createQuery("from DBUser as u where u.id=:id");
        query.setInteger("id",user_id);
        int count = 0;
        List list = query.list();
        Iterator iterator = null;
        if(!list.isEmpty())                             //判断 list 集合类是否为空
        {
            user = (User)list.get(0);
            list = user.getProducts();                  //取出用户购买的所有商品实例
            iterator = list.iterator();
            while(iterator.hasNext())
            {
                product = (Product)iterator.next();
                if(product.getId() == p_id)             //判断商品是否为指定的商品，如果是，即统计此商品的个数
                    count++;
            }
        }
        tx.commit();
        HibernateUtil.closeSession();
        return count;
    }
    public void buyProduct(User user,Product product) throws HibernateException
    {           //把商品加入到购物车中
        Session session = HibernateUtil.currentSession();
        Transaction tx = session.beginTransaction();
        user = (User)session.load(DBUser.class,new Integer(user.getId()));
        user.getProducts().add(product);                //把此商品添加到用户实体类的 Products 集合中
        session.update(user);                           //调用 session 的 update 方法，来更新此用户信息
        tx.commit();
        HibernateUtil.closeSession();

    }
    public void delProduct(User user,Product product) throws HibernateException
    {                                                   //把商品从购物车中删除
        Session session = HibernateUtil.currentSession();
        Transaction tx = session.beginTransaction();
        Product product1 = new DBProduct();
        user = (User)session.load(DBUser.class,new Integer(user.getId()));
        List list = user.getProducts();
        Iterator iterator = list.iterator();
```

```
            while(iterator.hasNext())
            {
                product1 = (Product)iterator.next();
                if(product1.getId() == product.getId())//找出需要删除的商品
                    user.getProducts().remove(product1);      //把此一个商品从 Product 的集合类中除去
            }
            session.update(user);                                    //重新更新 user
            tx.commit();
            HibernateUtil.closeSession();
        }
```

程序说明：

（1）countProduct()方法为获取到用户购买某一个商品的数量，因为在用户实体类中定义是 List 集合类型，允许存储重复对象，此方法就是统计某重复实例的个数。

（2）在 buyProduct(User user,Product product)和 delProduct(User user,Product product)两个方法当中，注意到这样一段代码：user = (User)session.load(SqlUser.class,new Integer(user.getId()))。如果没有这个代码，就会报出类似“此实体类的 Session 已经关闭或者不存在”错误。这是因为传递过来的 User 不是在这个 Session 中创建或者生成的，所以就不能取出用户实体类中的 List 集合中的商品实例（不存在，只能在 Session 中存在）。所以这里调用 load()方法函数来重新装载实例。

26.2.5　创建Struts的Action和ActionForm类

在功能介绍中，读者已经了解，需要实现商品的购买和删除等操作，当然开发者可以针对每个操作创建一个 Action 和 ActionForm 类，但是那样的话，需要创建的 Action 和 ActionForm 太多，不易维护。

这里通过 DispatchAction 来创建一个 Action 类，可以将上面的所有方法集合在一起，节省开发 Action 类的时间，而且易于 Action 类文件的维护。

26.2.5.1．创建 ShopCarForm 类

此类部分代码如下：

```
……
public class ShopCarForm extends ValidatorActionForm{
    /*********/
    private static final long serialVersionUID = 1L;
    private int p_Id = 0;
    private String p_Name = null;
    private float p_Price = 0;
    private int p_Stocks = 0;
    private String p_Discription = null;
    private int p_Kind = 0;

……//各属性对应的 set 和 get 方法。

    public ActionErrors validate(ActionMapping mapping,HttpServletRequest request){
        return null;
    }
}
```

代码说明：

- 此处的数据验证方法 validate 暂时空实现。在配置文件 struts-config.xml 需要指定 validate=”true”。

- 此类用于数据的封装，并交付给 Action 类进行处理。
- 由于 Action 类继承了 DispatchAction 类，所以这里的 ActionForm 需要继承 ValidatorActionForm 类。

26.2.5.2．创建 ShopCarAction 类

此类实现了 buyProduct()和 delProduct()两个方法，部分代码如下：

```
public class ShopCarAction extends DispatchAction{

    /*****加入购物车*****/
    public ActionForward buyProduct(ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) throws Exception
   {
        ShopCarForm actionForm = (ShopCarForm)form;
        int p_Id = actionForm.getP_Id();            //取出页面传递过来的商品 id
        HttpSession session = request.getSession();
        User user = (User)session.getAttribute(Constants.CURRENT_USER);    //取出当前登录的用户
        AbstractProduct aProduct = new SqlProduct();
        List list = aProduct.listProduct(p_Id);      //根据给定的商品 id，获取出该商品对象
        if(!list.isEmpty())                          //判断 List 集合类是否为空
        {
            Object[] rows = (Object[])(list.get(0));
            Product product = (Product)rows[0];
            new SqlUser ().buyProduct(user,product);          // 调用 buyProduct()把商品加入购物车
        }
        return mapping.getInputForward();                     //返回配置文件中 input 属性指定的页面
    }
    /*****从购物车中删除*****/
    public ActionForward delProduct(ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) throws Exception
  {
        ShopCarForm actionForm = (ShopCarForm)form;
        int p_Id = actionForm.getP_Id();
        System.out.println(p_Id);
        HttpSession session = request.getSession();
        User user = (User)session.getAttribute(Constants.CURRENT_USER);
        AbstractProduct aProduct = new SqlProduct();
        List list = aProduct.listProduct(p_Id);
        if(!list.isEmpty())
        {
            Object[] rows = (Object[])(list.get(0));
            Product product = (Product)rows[0];
            new SqlUser().delProduct(user,product);  //调用 delProduct()把商品删除
        }
        return mapping.findForward(Constants.RETURN_MYSHOPCAR);
  }
```

```
}
```

程序说明：

（1）Struts 是如何知道调用其中的方法来进行相应的操作了，在页面中传递过来了一个 method 变量（已经在 struts-config.xml 配置文件中定义，parameter=”method”）。通过判断 method 属性值为“buyProduct”还是“delProduct”来进行相应的方法调用。

（2）Constants.RETURN_MYSHOPCAR，是调用了 Constants 类中的常量 RETURN_MYSHOPCAR，这里常变量的值为“return_myShopCar”。需要在之前定义。

26.2.6　创建页面JSP文件

最后需要创建页面显示 JSP 文件：listProduct.jsp 展示所有商品信息；myShopCar.jsp 显示出登录用户的购物车中的商品信息。

26.2.6.1　创建 listProduct.jsp 文件

此文件的详细代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<%@ page import="cn.login.model.*" %>
<%@ page import="cn.login.model.sql.*"%>
<%@ page import="java.util.*"%>
<html>
<head>
<META http-equiv=Content-Type content="text/html;charset=GBK">
<link rel="stylesheet" href="include/CSS.CSS" type="text/css">
<title>我的购物车</title>
<body background="images/bg.gif">
<logic:present name="user">
  <h4>欢迎 <bean:write name="user" property="userid"/>!</h4>
</logic:present>
<table width="620"   border="1" align="center">
  <tr>
    <td colspan="7" align="center"><h4>产品展示</h4></td>
  </tr>
  <tr>
    <td width="40">序号</td>
    <td width="100">产品名称</td>
    <td width="60">价格</td>
    <td width="60">库存量</td>
    <td width="180">产品描述</td>
    <td width="100">产品类别</td>
    <td width="80"> </td>
  </tr>
<%
            int count=1;
            Product product = new DBProduct();
            User user = new DBUser();
```

```
            Set set = new HashSet();
            AbstractProduct aProduct = new SqlProduct();
            List list = aProduct.listProducts();
            if(!list.isEmpty())                    //判断 List 集合是否为空
            {
                count=1;
                ListIterator iterator =list.listIterator();
                while(iterator.hasNext())
                {
                    Object[] row = (Object[])iterator.next();
                    product = (Product)row[0];
                    String kind_name = (String)row[1];
                    out.println("<tr>");
                    out.println("<td>"+count+"</td>");
                    out.println("<td>"+product.getName()+"</td>");
                    out.println("<td>"+product.getPrice()+"</td>");
                    out.println("<td>"+product.getStocks()+"</td>");
                    out.println("<td>"+product.getDiscription()+"</td>");
                    out.println("<td>"+kind_name+"</td>");
                    out.println("<td><a
href='/login/shopCarAction.do?method=buyProduct&p_Id="+product.getId()+"'>加入购物车</a></td>");
                    out.println("</tr>");
                    count++;
                }
            }
%>
</table>
<center><html:link  forward="index">返回首页</html:link>  |  <html:link  forward="myShopCar">我的购物车
</html:link></center>
</body>
</head>
</html>
```

程序说明：

（1）注意一定要加上如下两段代码，不然会出现乱码：

```
<%@ page contentType="text/html;charset=GBK" %>
<META http-equiv=Content-Type content="text/html;charset=GBK">
```

（2）还要注意 charset 属性全都是小写，千万不要写成大写，不然也会出现乱码，笔者就出现过类似的错误，结果就是花费了大量的时间找错误。

（3）一定要引入使用到的标签和类文件。

（4）在十四章的实例中，存在 Session 中的用户信息仅仅是用户名，而这里改成了用户对象 User，所以这里的<bean:write name="user" property="userid"/>，必须指定 property="userid"，表示在页面上打印出用户名。

26.2.6.2　创建 myShopCar.jsp 文件

显示出登录用户的购物车中所有商品信息的页面，代码如下：

```
<%@ page contentType="text/html;charset=GBK" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
```

```
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<%@ page import="cn.login.model.*" %>
<%@ page import="cn.login.model.sql.*"%>
<%@ page import="java.util.*"%>
<html>
<META http-equiv=Content-Type content="text/html;charset=GBK">
<link rel="stylesheet" href="include/CSS.CSS" type="text/css">
<head><title>所购物品</title></head>
<body background="images/bg.gif">
<logic:present name="user">
  <h4>欢迎 <bean:write name="user" property="userid"/>!</h4>
</logic:present>
<table width="540"    border="1" align="center">
  <tr>
    <td colspan="6" align="center"><h4>我的购物车</h4></td>
  </tr>
  <tr>
    <td width="40">序号</td>
    <td width="100">产品名称</td>
    <td width="60">价格</td>
    <td width="180">产品描述</td>
     <td width="60">数量</td>
    <td width="80"> </td>
  </tr>
<%
   User session_user = (User)session.getAttribute("user");
   if(session_user !=null)
   {
          int count=1;
            Product product = new DBProduct();
            User user = new DBUser();
            AbstractUser aUser = new SqlUser ();
            List list1 = aUser.listMyProducts(session_user.getId());
            if(!list1.isEmpty())
            {
                  ListIterator iterator = list1.listIterator();
                  while(iterator.hasNext())
                  {
                        Object[] rows = (Object[])iterator.next();
                        //user = (User)rows[0];
                        product = (Product)rows[1];
                        int num = aUser.countProduct(session_user.getId(),product.getId());
                        out.println("<tr>");
                        out.println("<td>"+count+"</td>");
                        out.println("<td>"+product.getName()+"</td>");
                        out.println("<td>"+product.getPrice()+"</td>");
                        out.println("<td>"+product.getDiscription()+"</td>");
                        out.println("<td>"+num+"</td>");
                        out.println("<td><a href='/login/shopCarAction.do?method=delProduct
                              &p_Id="+product.getId()+"'>删除</a></td>");
```

```
                    out.println("</tr>");
                    count++;
                }
            }
        }
%>
</table>
<center><html:link  forward="index">返回首页</html:link>  |  <html:link  forward="listProduct">继续购物</html:link></center>
</body>
</html>
```

代码说明：此页面功能和上一个页面功能类似，其中调用 aUser.countProduct(int userid,int p_id)方法来统计此商品的购买数量。

26.2.7　运行此Web应用

使用 Lomboz 启动 tomcat 服务器，在 IE 中输入地址 http://localhost:8080/login 显示实例效果。执行效果已经在第一小节中有所介绍。

26.3　Hibernate的自动生成工具

在编辑好*.hbm.xml 映射文件之后，是可以使用 net.sf.hibernate.tool.hbm2ddl.schemaExportTask 来自动生成数据库表的，这样可以大大简化创建表格的工作。具体实现方法如下：

（1）首先在之前创建的 HibernateUtil 类中添加如下两种方法：

```
/** 根据映射文件自动创建数据库表，同时把生成的 sql 语句输出到 c:\sms.sql 文件中  ***/
public static void createDbTable() throws HibernateException{
    Configuration conf = new Configuration().configure(new File("hibernate.cfg.xml"));
    SchemaExport dbExport = new SchemaExport(conf);
    dbExport.setOutputFile("d:\\sms.sql");
    dbExport.create(true,true);
}
/**
 * 根据映射文件中新增的字段更新数据库表中的字段
 * （1）表的原始数据不会被删除
 * （2）根据 XML 映射文件中的字段来更新，如果表中出现同名字段，怎不改变
 * （3）表中有字段在 xml 文件中没有定义，则也不删除
 */
public static void updateDbTable() throws HibernateException{
    Configuration conf = new Configuration().configure(new File("hibernate.cfg.xml"));
    new SchemaUpdate(conf).execute(true,true);
}
```

程序说明：其中也可以不指定 hibernate.cfg.xml，即直接 Configuration conf = new Configuration().configure())，这时会自动查找 WEB-INF/classes 目录下的配置文件。前一种方法根据映射文件创建数据库表，后面方法根据映射文件中新增的字段来更新数据库中已经创建的表。

（2）创建一个调用类 createTable.java（包含 public static void main(String[] args)主方法）来调用以

上创建的两个方法，从而自动创建出相应的数据库表，其中调用代码如下：

```
cn.login.db.HibernateUtil.createDbTable();
cn.login.db.HibernateUtil.updateDbTable();
```

在 Eclipse 中运行这个主类，即可以在 hibernate.cfg.xml 配置文件指定的数据库中创建出实体类相对应的数据库表。

注意：使用这个方法自动创建数据库表时，需要映射文件中的属性设置尽量详细，例如字段的类型、长度等等，不然的话，自动创建的数据库表选择默认值。

除了这种使用 Java 程序的方法之外还可以使用 Ant 来自动创建数据库表，其具体使用方法请参考相应资料。

26.4　本章小结

前两章具体讲解了 struts 和 Hibernate 技术，本章结合这两个技术重新实现了用户登录应用，并添加了购物车功能，展示了 struts 和 Hibernate 的完美结合：Strut 所提供了框架支持（视图－控制－模型），使得整个 Web 系统结构清晰、便于管理和维护；Hibernate 实现了数据层的持久化，使得数据层真正得到了面向对象。

之上所有章节已经比较全面地向读者介绍了 JSP 各方面的知识，由首先的纯 JSP 编写 Web 应用，到后来的 JavaBean 使用，以及 JSP 标签化的出现使得页面趋向统一。再到 MVC 框架的提出，而 struts 又是 MVC 框架实现的最好选择。Hibernate 的使用使得尴尬的非面向对象数据层成为过去，开发者可以摆脱数据层的表示而专注业务逻辑的实现。