第 14 章　JavaBean实现用户注册登录系统

本章将使用 JavaBean 来重新实现第 12 章的用户注册系统，JavaBean 可以实现代码重用，以及使得业务逻辑、存取数据等操作和 JSP 页面代码相分离，这样可以使整个系统结构更加清晰。虽然 JavaBean 可以部分实现控制层（Control）和视图层（View）的相互独立，但是它并没有完全实现 MVC 开发模式，有关 MVC 概念和实现过程将在本书的高级部分介绍。

本章要点包括以下内容：

- JavaBean 的环境配置
- 使用 JavaBean 重新实现用户注册系统
- 使用登录验证码功能
- MD5 转码技术

14.1　JavaBean环境配置

创建的 JavaBean 类文件（此处是已经被编译好的*.class 文件）应该保存在 Web 模块的 WEB-INF/classes 目录下。Tomcat 容器会自动加载 Web 模块的 WEB-INF/classes 目录下所有 class 类文件。如果以第 12 章的 Register 模块为例，则存放 Class 类文件的目录为 Register/WEB-INF/classes。

注意：Register/WEB-INF/classes **目录下的所有** class **类文件都不是后缀为** java **的源代码文件。**

在使用 Lomboz＋Eclipse 创建 Register 模块的时候，Lomboz 会自动在 Web 模块的根目录下创建 src 和 j2src 两个源文件夹。本章将使用 j2src 源文件夹来存放 JavaBean 源代码文件，删除 src 文件夹（使用不到该文件夹）。JavaBean 的环境配置步骤如下：

（1）右击 MyRegister 项目，选择“Properties”命令，弹出如图 14.1 所示的对话框。

（2）单击左边的“Java Build Path”选项，然后选择“Source”选项卡，如图 14.1 所示。

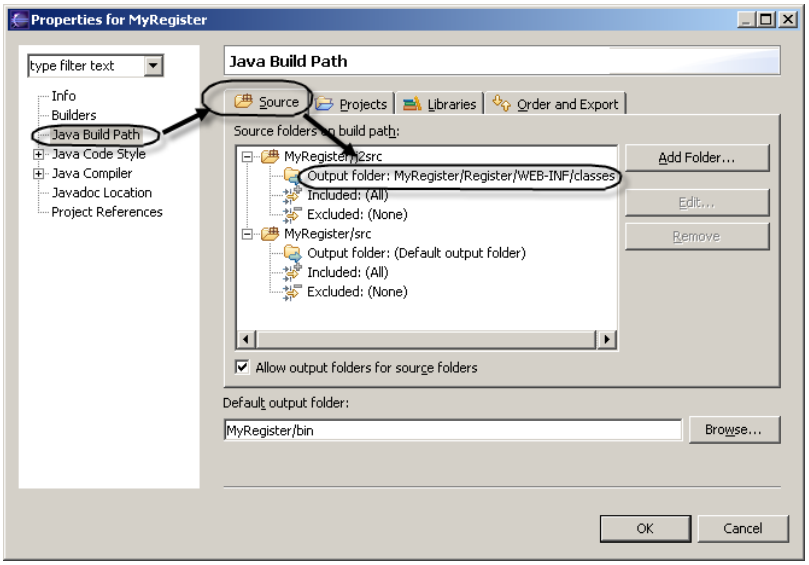

图 14.1　配置 JavaBean 环境变量

（3）选中下方的“Allow output folders for source folders”选项，单击“Edit”按钮来编辑 MyRegister/j2src 的输出文件夹，弹出如图 14.2 所示的对话框。

（4）单击“Browse”按钮，按照图 14.2 所示设置 class 文件的输入目录，此处的输出目录为 Register/WEB-INF/classes。单击“OK”按钮完成输出文件夹的设置。

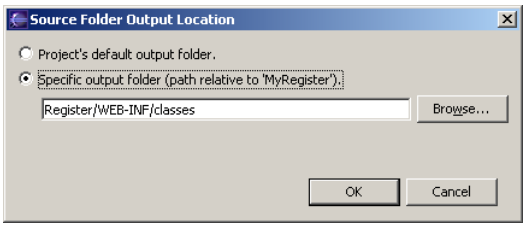

图 14.2　选择 class 文件输出文件夹

（5）在如图 14.1 所示对话框中把 src 源文件夹删除，否则 MyProject 项目会起冲突。

这样的配置之后，创建的 JavaBean 源代码将保存在 j2src 目录下。相应的 class 文件（Java 源文件编译之后的文件）会自动生成在“Register/WEB-INF/classes”目录下，达到了源文件和 class 文件的相互分离。

注意：使用 Eclipse 工具开发 JavaBean 时，是不需要手动来编译 Java 文件的。在 j2src 目录下每创建一个 Java 文件，系统会自动在“Register/WEB-INF/classes”目录下生成一个相应的 class 文件，这样大大提高了开发效率。

14.2　创建JavaBean

在十二章中创建的用户注册系统是由纯 JSP 实现的，即所有的数据库操作、业务逻辑控制等等都是在 JSP 文件中实现的，把这种模式称为 Model 1 体系结构。这样的开发模式比较适合小型应用和初学者使用，代码比较集中，对初学者比较容易理解。当需要开发一个非常庞大的应用系统时，它的很多缺点都会明显的暴露出来：

- 代码重复：把大量的业务逻辑、数据库操作等语句直接写到 JSP 文件中，会使得代码大量的重复，在修改的时候也可能会出现遗漏。
- 可维护性差：把业务逻辑与显示逻辑写在一起，JSP 页面会显得非常混乱，使得后期维护非常的困难，而且要求开发人员要对 HTML 以及程序设计语言都要很有经验，不便于工作的分工。
- 可测试性差：使用 Model 1 模式开发的应用很难进行测试。这样会增加很多的工作量。

为了降低代码重复性，可以将共用代码写在一个独立 JSP 文件中，然后在需要调用的 JSP 文件用使用<include>来引用该外部文件。这样虽然在一点程度上解决了问题，但是可测试性还是很差，而且整个系统结构不够清晰，当 JSP 文件越来越多时，会显得非常混乱。

这一章将使用 JavaBean 来重新实现用户注册系统。将一些逻辑操作独立出来写到 JavaBean 类中，然后在 JSP 文件进行调用，从而使得 JSP 文件更多地只负责数据显示功能。JavaBean 可以部分实现试图层和控制层的相互独立。本章需要创建的 JavaBean 类文件以及所在包的情况如图 14.3 所示。

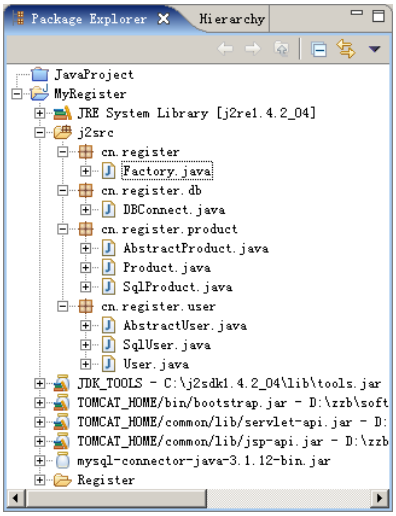

图 14.3　JavaBean 类文件目录

创建的各 JavaBean 类文件需要实现的功能如下：

- DBConnect.java 数据库操作类：创建数据连接以及实现数据库基本操作。
- User.java 接口类：是一个用户对象接口类，对应数据库 user 表，定义了（接口类不可以实现方法）对各个字段进行赋值和取值的方法。
- AbstractUser.java 抽象类：该类实现 User.java 接口类中定义的方法，以及定义用户属性变量。
- SqlUser.java 用户类：该类继承 AbstractUser.java 类，并定义对用户进行操作的各类方法，例如用户身份验证方法、获取和存储用户信息操作方法等。
- Product.java 商品接口类：是商品对象的接口类，对应数据库中 product 表。并定义了对各个字段进行复值和取值的方法（接口类不能实现方法）。
- AbstractProduct.java 抽象类：实现 Product.java 接口类中定义的方法，以及定义商品的属性变量。
- SqlProduct.java 商品类：继承 AbstractProduct.java 抽象类，并定义操作商品的各类方法，其中包括商品信息的获取和存储方法。
- Factory.java 抽象类：是个抽象类，统一对所有类文件进行初始化和管理。
- SqlFactory.java 实现类：继承 Factory.java 类，实现 Factory.java 类定义的方法。
- DateFormat.java 类：实现符合"yyyy-MM-dd"格式的字符串日期转化成 Long 类型的日期类型。

14.2.1　数据库操作类DBConnect.java

本节将创建数据库连接和数据库基本操作方法的类 DBConnect.java。其中使用到了数据库连接池技术（Tomcat 中的连接池配置已经在前面小节进行了介绍），并使用到了 JNDI 和 DataSource 对象来获取到数据库连接。该数据库操作类 DBConnect.java 的源代码如下：

```
package cn.register.db;
import java.sql.*;
import javax.naming.InitialContext;
import javax.sql.DataSource;

public class DBConnect {
    private Connection conn;
    private Statement   stmt;
```

```
private PreparedStatement pstmt;
private ResultSet    rst;
private String       str1;                                    //创建数据库的一个 Connection 连接
private void init(){
      try{
           InitialContext ctx = new InitialContext();
            DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/mysql");
            conn = ds.getConnection();
      }
      catch(Exception e) {
            e.printStackTrace();
      }
}
public DBConnect(){                    //构造函数
      try{
            init();                    //获得一个数据库连接
            stmt = conn.createStatement();
      }
      catch(Exception e)   {e.printStackTrace();}
}
                                       //执行数据库查询语句，s 为 sql 语句，把查询结果赋给 ResultSet
public void excuteQuery(String s) {
      try{
            if(stmt != null) {
                  rst = stmt.executeQuery(s);
            }
      }
      catch(Exception e) {
            e.printStackTrace();
      }
}
                                       //对数据库进行 update 操作
public int excuteUpdate(String s) {
      int status = 0;
      try{
            if(stmt != null)
                  status = stmt.executeUpdate(s);
      }
      catch(Exception e) {
            e.printStackTrace();
      }
      return status;
}
                                       //以下为赋值方法
public void setString(int i,String s) {                        //字符串赋值
   try { pstmt.setString(i,s); }
   catch(Exception e) { e.printStackTrace(); }
}
public void setBoolean(int i, boolean flag) {                   //布尔型赋值
    try{ pstmt.setBoolean(i, flag);}
```

```
    catch(Exception e) { e.printStackTrace();}
}
public void setDate(int i, Date date) {                          //日期类型赋值
    try{pstmt.setDate(i, date);}
    catch(Exception e) { e.printStackTrace();}
}
public void setTime(int i, Time time) {                          //时间类型赋值
    try{pstmt.setTime(i,time);}
    catch(Exception e) { e.printStackTrace();}
}
public void setShort(int i, short word0) {                       //Short 类型赋值
    try{pstmt.setShort(i,word0);}
    catch(Exception e) {e.printStackTrace();}
}
public void setInt(int i, int j) {                               //整数类型赋值
    try{pstmt.setInt(i,j);}
    catch(Exception e) { e.printStackTrace();}
}
public void setLong(int i, long l) {                             //长整型赋值
    try{pstmt.setLong(i, l);}
    catch(Exception e) { e.printStackTrace();}
}
public void setFloat(int i, float f) {                           //浮点型赋值
    try{pstmt.setFloat(i, f);}
    catch(Exception e) { e.printStackTrace();}
}
public void setDouble(int i, double d) {                         //Double 类型赋值
    try{pstmt.setDouble(i, d);}
    catch(Exception e) {e.printStackTrace();}
}
                                        //以下为取值方法
public boolean getBoolean(int i)   throws Exception{             //取得布尔类型的字段，通过列号
    return rst.getBoolean(i);
}
public boolean getBoolean(String s) throws Exception{            //取得布尔类型的字段，通过字段名
    return rst.getBoolean(s);
}
public Date getDate(int i) throws Exception{                     //取得 Date 类型的字段，通过列号
    return rst.getDate(i);
}
public Date getDate(String s) throws Exception{                  //取得 Date 类型的字段，通过字段名
    return rst.getDate(s);
}
public Time getTime(int i) throws Exception{                     //取得 Time 类型的字段，通过列号
    return rst.getTime(i);
}
public Time getTime(String s) throws Exception{                  //取得 Time 类型的字段，通过字段名
    return rst.getTime(s);
}
public double getDouble(int i) throws Exception{                 //取得 Double 类型的字段，通过列号
```

```
        return rst.getDouble(i);
    }
    public double getDouble(String s) throws Exception{          //取得 Double 类型的字段，通过字段名
        return rst.getDouble(s);
    }
    public float getFloat(int i) throws Exception{               //取得 Float 类型的字段，通过列号
        return rst.getFloat(i);
    }
    public float getFloat(String s) throws Exception{            //取得 Float 类型的字段，通过字段名
        return rst.getFloat(s);
    }
    public int getInt(int i) throws Exception{                   //取得整数类型的字段，通过列号
        return rst.getInt(i);
    }
    public int getInt(String s) throws Exception{                //取得整数类型的字段，通过字段名
        return rst.getInt(s);
    }
    public long getLong(int i) throws Exception{                 //取得长整型的字段，通过列号
        return rst.getLong(i);
    }
    public long getLong(String s) throws Exception{              //取得长整型的字段，通过字段名
        return rst.getLong(s);
    }
    public short getShort(int i) throws Exception{               //取得 Short 类型的字段，通过列号
        return rst.getShort(i);
    }
    public short getShort(String s) throws Exception{            //取得 Short 类型的字段，通过字段名
        return rst.getShort(s);
    }
    public String getString(int i)    throws Exception{          //取得字符串类型的字段，通过列号
       return rst.getString(i);
    }
    public String getString(String s)    throws Exception {      //取得字符串类型的字段，通过字段名
       return rst.getString(s);
    }
                                        //指针下移一位
     public boolean next(){
            try {return rst.next();}
            catch(Exception e) {
                  e.printStackTrace();
                  return false;
            }
     }
                                        //释放内容
     public void close(){
            try{
                  if(conn !=null)    conn.close();
                  if(stmt !=null)    stmt.close();
                  if(rst != null)    rst.close();
            }
```

```
            catch(Exception e)   {e.printStackTrace();}
        }
}
```

程序说明：

- DBConnect.init()方法获得一个数据库连接，即返回一个 Connection 类型，由于这个方法是内部调用，所以设置为 private 类型。
- 这个类重新实现了操作 Sql 语句的 executeQuery()、executeUpdate()、next()以及 close()等方法。这个类使得对数据库的基本操作完全从 JSP 页面中解脱出来。
- “一次操作一次连接”模式有很大的不足，由于连接操作频繁会加重 CPU 资源，使得运行速度降低，这里实现的数据库连接池技术可以弥补传统模式的不足。使用了连接池技术后，所有数据库连接创建、分配和释放都统一由连接池管理器执行。

14.2.2　创建用户接口类User.java和抽象类AbstractUser.java

Java 是一个面向对象的开发语言，它的思想就是一切皆为对象。在本实例当中，将用户当着一个对象来处理。

首先创建用户对象的接口类 User.java，在这个类中只对用户的各个属性定义赋值和取值方法，并没有实现。然后创建 AbstractUser.java 抽象类来实现 User.java 接口类中定义的方法。最后由 SqlUser.java 类继承 AbstracUser.java 抽象。这样的一个开发方式是为了后期代码的扩展，例如以后要创建一个管理员用户，则可以直接继承 AbstractUser.java 抽象类来实现。

下面创建 User.java 接口类和 AbstractUser.java 抽象类。

1．User.java 接口类

User.java 接口只定义各种取值和赋值方法，具体源代码如下：

```
package cn.register.user;

public interface User {
   public abstract void setUser_id(String user_id);
   public abstract String getUser_id();
   public abstract void setPassword(String password);
   public abstract String getPassword();
   public abstract void setName(String name);
   public abstract String getName();
   public abstract void setSex(String sex);
   public abstract String getSex();
   public abstract void setBirth(long birth);
   public abstract long getBirth();
   public abstract void setDescription(String description);
   public abstract String getDescription();
}
```

程序说明：该类文件定义了对各类用户属性（对应 User 数据库表中的各字段）的赋值和取值方法。

2．AbstractUser.java 抽象类

创建的抽象 AbstractUser.java 继承结果 User.java，具体实现代码如下：

```
package cn.register.user;
```

```
public abstract class AbstractUser implements User {
    private String user_id;                    //用户名
    private String password;                   //用户密码
    private String name;                       //用户姓名
    private String sex;                        //用户性别
    private long birth;                        //用户出生年月
    private String description;                //用户描述

    public void setUser_id(String user_id){
        this.user_id = user_id;
    }
    public String getUser_id(){
        return user_id;
    }
    public void setPassword(String password){
        this.password = password;
    }
    public String getPassword(){
        return password;
    }
    public void setName(String name){
        this.name = name;
    }
    public String getName(){
        return name;
    }
    public void setSex(String sex){
        this.sex = sex;
    }
    public String getSex(){
        return sex;
    }
    public void setBirth(long birth){
        this.birth = birth;
    }
    public long getBirth(){
        return birth;
    }
    public void setDescription(String description){
        this.description = description;
    }
    public String getDescription(){
        return description;
    }
}
```

程序说明：该抽象类实现了 User.java 接口类中所定义的所有赋值和取值方法，另外，还定义了所有的用户属性变量。该抽象类用于扩展，下面创建的 SqlUser.java 类将继承该类，如果以后需要创建管理员用户对象时，可以直接继承该抽象类完成。

14.2.3　SqlUser.java用户操作类

该类继承了上面创建的 AbstractUser.java 类，所有拥有 AbstractUser.java 类所有的属性和方法。另外，该类还定义了多个对 User 用户进行数据库操作的方法，具体代码如下：

```
package cn.register.user;
import cn.register.db.DBConnect;
import cn.register.MD5;
public class SqlUser extends AbstractUser{
    private boolean checkRPwd(String password, String rpassword)
    {
        if(password !=null && password != null){
            if(password.equals(rpassword))
                return true;
            else
                return false;
        }else
            return false;
    }
    public boolean saveUser(User user){
        String User_id = user.getUser_id();
        String Password = user.getPassword();
        String Name = user.getName();
        String Sex = user.getSex();
        long Birth = user.getBirth();
        String Description = user.getDescription();
        String str = "insert into users values('"+User_id+"','"+Password+"','"+Name+"','"+Sex+"','"+Birth+"','"+Description+"')";
        try{
            DBConnect dbconnect = new DBConnect();
            dbconnect.excuteUpdate(str);
            return true;
        }catch(Exception e){
            e.printStackTrace();
            return false;
        }

    }
    public User getUser(int ID){
        User user = null;
        String str = "select * from users where USER_ID='"+ID+"'";
        try{
            DBConnect dbconnect = new DBConnect();
            dbconnect.excuteQuery(str);
            if(dbconnect.next()){
                user.setUser_id(dbconnect.getString(1));
                user.setPassword(dbconnect.getString(2));
                user.setName(dbconnect.getString(3));
                user.setSex(dbconnect.getString(4));
                user.setBirth(dbconnect.getLong(5));
```

```
                    user.setDescription(dbconnect.getString(6));
                }
            }catch(Exception e){
                e.printStackTrace();
            }
            return user;
        }
        public int checkUser(int ID, String password){
            int index = 0;
            User user = getUser(ID);
            if(user != null){
                if (user.getPassword().equals(MD5.toMD5(password)))
                    index = 0;
                else
                    index = 2;
            }else
                index = 1;
            return index;
        }
}
```

程序说明：

（1）checkRPwd()方法实现对用户输入密码和验证密码的对比操作，如果相等则返回 true，否则返回 false。

（2）saveUser()方法将一个用户的注册信息存储到数据库中，传入参数为 User 类型。如果操作成功返回 true，否则数据库操作失败，返回 false。

（3）getUser()方法通过一个用户的 ID 返回该用户的所有信息，这些信息被封装到 User 实例中。如果该 ID 用户不存在，则返回 null。

（4）checkUser()方法通过传入用户 ID 和密码，进行用户身份验证，如果该用户不存在则返回参数 1，如果密码不正确则返回 2，否则表示用户身份合法，返回 0。

（5）其中使用到了 MD5 类，调用其中的 toMD5()方法来对密码进行编码。在进行密码对比的时候，一定要记得编码之后再比较。

14.2.4　创建商品接口类Product.java和抽象类AbstractProduct.java

类似用户对象所创建的类，接下来为商品创建响应的 Product.java 接口类和 AbstractProduct.java 抽象类。

1．Product.java 接口类

该接口类的详细代码如下：

```
package cn.register.product;

public interface Product {
      public abstract void setProduct_id(int product_id);
       public abstract int getProduct_id();
       public abstract void setProduct_name(String product_name);
       public abstract String getProduct_name();
       public abstract void setPrice(float price);
```

```
    public abstract float getPrice();
    public abstract void setDescription(String description);
    public abstract String getDescription();
}
```

程序说明：类似 User.java 接口，该程序只定义了有关商品属性变量的复制和取值方法。

2．AbstractProduct.java 抽象类

继承 Product.java 接口类的抽象类 AbstractProduct.java 代码如下：

```
package cn.register.product;

public abstract class AbstractProduct implements Product {
    private int product_id;                    //商品号
    private String product_name;               //商品名称
    private float price;                       //商品价格
    private String description;                //商品描述

    public void setProduct_id(int product_id){
        this.product_id = product_id;
    }
    public int getProduct_id(){
        return product_id;
    }
    public void setProduct_name(String product_name){
        this.product_name = product_name;
    }
    public String getProduct_name(){
        return product_name;
    }
    public void setPrice(float price){
        this.price = price;
    }
    public float getPrice(){
        return price;
    }
    public void setDescription(String description){
        this.description = description;
    }
    public String getDescription(){
        return description;
    }
}
```

程序说明：类似 AbstractProduct.java 抽象类，该类定义了有关商品的所有属性变量，并实现了 Product.java 接口类中所有定义的赋值和取值方法。

14.2.5　SqlProduct.java商品操作类

类似于 SqlUser.java 类，该类继承了 AbstractProduct.java 抽象类，另外还定义了对 Product 商品进行操作的方法，其详细代码如下：

```
package cn.register.product;
import cn.register.Factory;
import java.util.ArrayList;
import java.util.Iterator;
import cn.register.db.DBConnect;

public class SqlProduct extends AbstractProduct{

	public Iterator getProducts(){
		ArrayList arraylist = new ArrayList();
		Product product = Factory.getInstance().initProduct();
		String str = "select * from products";
		try{
			DBConnect dbconnect = new DBConnect();
			dbconnect.excuteQuery(str);
			while(dbconnect.next()){
				product.setProduct_id(dbconnect.getInt(1));
				product.setProduct_name(dbconnect.getString(2));
				product.setPrice(dbconnect.getFloat(3));
				product.setDescription(dbconnect.getString(4));
				arraylist.add(product);
			}
			return arraylist.iterator();
		}catch(Exception e){
			e.printStackTrace();
			return null;
		}
	}
}
```

程序说明：该类中只定义了一个 getProducts()方法，它用来返回数据库中所有的商品信息，返回的是商品“反复器”类。

14.2.6　创建工厂类Factory.java和SqlFactory.java

这里创建的 Factory.java 和 SqlFactory.java 类，是用来对其他所有类进行统一初始化操作的。其中 Factory.java 类为抽象类，该类中除了实现初始化本身的 getInstance()，还定义了四个用于初始化其他类的方法，这些方法等待 SqlFactory.java 类实现。Factory.java 抽象类的实现代码如下：

```
package cn.register;
import cn.register.product.SqlProduct;
import cn.register.user.SqlUser;

public abstract class Factory {
	private static Factory factory = null;
	public Factory(){}
	public static Factory getInstance(){
		if(factory == null){
			try{
				Class factoryClass = Class.forName("cn.register.SqlFactory");
```

```
                    factory = (Factory)factoryClass.newInstance();
                }catch(Exception e){
                    e.printStackTrace();
                }
            }
            return factory;
        }
        public abstract User initUser();
        public abstract SqlUser initSqlUser();
        public abstract Product initProduct();
        public abstract SqlProduct initSqlProduct();}
```

SqlFactory.java 类继承了 Factory.java 类，该类的实现代码如下：

```
package cn.register;
import cn.register.product.Product;
import cn.register.product.SqlProduct;
import cn.register.user.SqlUser;
import cn.register.user.User;

public class SqlFactory extends Factory{
    public User initUser() {
        // 初始化 User 实例
        return (User) new SqlUser();
    }
    public Product initProduct() {
        // 初始化 Product 实例
        return (Product) new SqlProduct();
    }
    public SqlUser initSqlUser() {
        // 初始化 SqlUser 实例
        return new SqlUser();
    }
    public SqlProduct initSqlProduct() {
        // 初始化 SqlProduct 实例
        return new SqlProduct();
    }
}
```

程序说明：实现的四个方法依次初始化 User、Product、SqlUser 以及 SqlProduct 实例。

14.2.7　创建日期格式转换类DateFormat.java

在本实例中，需要将字符串类型的日期转换成长整数类型。调用下面类中的方法，将实现字符串日期和长整数类型日期的转换。详细代码如下：

```
package cn.register;
import java.text.ParsePosition;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateFormat {
    public static long getDate(String s)
    {
```

```
            SimpleDateFormat simpledateformat = new SimpleDateFormat("yyyy-MM-dd");
            ParsePosition parseposition = new ParsePosition(0);
            Date date = simpledateformat.parse(s, parseposition);
            return date.getTime();
        }
}
```

14.2.8　MD5.java类

MD5 的全称是 Message-Digest Algorithm 5，在 90 年代初由 MIT 的计算机科学实验室和 RSA Data Security Inc 发明，经 MD2、MD3 和 MD4 发展而来。

Message-Digest 泛指字节串(Message)的 Hash 变换，就是把一个任意长度的字节串变换成一定长的大整数。请注意我使用了“字节串”而不是“字符串”这个词，是因为这种变换只与字节的值有关，与字符集或编码方式无关。

MD5 将任意长度的“字节串”变换成一个 128bit 的大整数，并且它是一个不可逆的字符串变换算法，换句话说就是，即使你看到源程序和算法描述，也无法将一个 MD5 的值变换回原始的字符串，从数学原理上说，是因为原始的字符串有无穷多个，这有点象不存在反函数的数学函数。

MD5 算法的主要应用有两个方面：

（1）MD5 的典型应用是对一段 Message(字节串)产生 fingerprint(指纹)，以防止被“篡改”。举个例子，你将一段话写在一个叫 readme.txt 文件中，并对这个 readme.txt 产生一个 MD5 的值并记录在案，然后你可以传播这个文件给别人，别人如果修改了文件中的任何内容，对这个文件重新计算 MD5 时就会发现。如果再有一个第三方的认证机构，用 MD5 还可以防止文件作者的“抵赖”，这就是所谓的数字签名应用。

（2）MD5 还广泛用于加密和解密技术上，在很多操作系统中，用户的密码是以 MD5 值（或类似的其他算法）的方式保存的， 用户 Login 的时候，系统是把用户输入的密码计算成 MD5 值，然后再去和系统中保存的 MD5 值进行比较，而系统并不“知道”用户的密码是什么。

该类就是实现了 MD5 算法对密码进行加密，然后再保存在数据库中，这样可以防止别人盗窃密码，其实现 MD5 算法的源代码如下所示：

```
package com.util;
public class MD5 {
        /* 下面这些 S11-S44 实际上是一个 4*4 的矩阵，在原始的 C 实现中是用#define 实现的，
          这里把它们实现成为 static final 是表示了只读，且能在同一个进程空间内的多个 Instance 间共享*/
          static final int S11 = 7;
          static final int S12 = 12;
          static final int S13 = 17;
          static final int S14 = 22;
          static final int S21 = 5;
          static final int S22 = 9;
          static final int S23 = 14;
          static final int S24 = 20;
          static final int S31 = 4;
          static final int S32 = 11;
          static final int S33 = 16;
          static final int S34 = 23;
          static final int S41 = 6;
```

```
static final int S42 = 10;
static final int S43 = 15;
static final int S44 = 21;
static final byte[] PADDING = { -128, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
/*  下面的三个成员是 MD5 计算过程中用到的 3 个核心数据，在原始的 C 实现中
    被定义到 MD5_CTX 结构中*/
private long[] state = new long[4];   // state (ABCD)
private long[] count = new long[2];   // number of bits, modulo 2^64 (lsb first)
private byte[] buffer = new byte[64]; // input buffer
/* digestHexStr 是 MD5 的惟一一个公共成员，是最新一次计算结果的 16 进制 ASCII 表示. */
public String digestHexStr;
/* digest,是最新一次计算结果的 2 进制内部表示，表示 128bit 的 MD5 值. */
private byte[] digest = new byte[16];
/*getMD5ofStr 是类 MD5 最主要的公共方法，入口参数是你想要进行 MD5 变换的字符串
  返回的是变换完的结果，这个结果是从公共成员 digestHexStr 取得的.  */
public String getMD5ofStr(String inbuf) {
        md5Init();
        md5Update(inbuf.getBytes(), inbuf.length());
        md5Final();
        digestHexStr = "";
        for (int i = 0; i < 16; i++) {
                digestHexStr += byteHEX(digest[i]);
        }
        return digestHexStr;
}
// 这是 MD5 这个类的标准构造函数，JavaBean 要求有一个 public 的并且没有参数的构造函数
public MD5() {
        md5Init();
        return;
}
/* md5Init 是一个初始化函数，初始化核心变量，装入标准的幻数  */
private void md5Init() {
        count[0] = 0L;
        count[1] = 0L;
        ///* Load magic initialization constants.
        state[0] = 0x67452301L;
        state[1] = 0xefcdab89L;
        state[2] = 0x98badcfeL;
        state[3] = 0x10325476L;
        return;
}
/* F, G, H ,I  是 4 个基本的 MD5 函数，在原始的 MD5 的 C 实现中，由于它们是
简单的位运算，可能出于效率的考虑把它们实现成了宏，在 java 中，我们把它们
实现成了 private 方法，名字保持了原来 C 中的。 */
private long F(long x, long y, long z) {
        return (x & y) | ((~x) & z);
}
```

```
    private long G(long x, long y, long z) {
            return (x & z) | (y & (~z));
    }
    private long H(long x, long y, long z) {
            return x ^ y ^ z;
    }

    private long I(long x, long y, long z) {
            return y ^ (x | (~z));
    }
    /*   FF,GG,HH 和 II 将调用 F,G,H,I 进行近一步变换
       FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
       Rotation is separate from addition to prevent recomputation. */
    private long FF(long a, long b, long c, long d, long x, long s,
            long ac) {
            a += F (b, c, d) + x + ac;
            a = ((int) a << s) | ((int) a >>> (32 - s));
            a += b;
            return a;
    }
    private long GG(long a, long b, long c, long d, long x, long s,
            long ac) {
            a += G (b, c, d) + x + ac;
            a = ((int) a << s) | ((int) a >>> (32 - s));
            a += b;
            return a;
    }
    private long HH(long a, long b, long c, long d, long x, long s,
            long ac) {
            a += H (b, c, d) + x + ac;
            a = ((int) a << s) | ((int) a >>> (32 - s));
            a += b;
            return a;
    }
    private long II(long a, long b, long c, long d, long x, long s,
            long ac) {
            a += I (b, c, d) + x + ac;
            a = ((int) a << s) | ((int) a >>> (32 - s));
            a += b;
            return a;
    }
    /*md5Update 是 MD5 的主计算过程，inbuf 是要变换的字节串，inputlen 是长度，这个
      函数由 getMD5ofStr 调用，调用之前需要调用 md5init，因此把它设计成 private 的*/
    private void md5Update(byte[] inbuf, int inputLen) {
            int i, index, partLen;
            byte[] block = new byte[64];
            index = (int)(count[0] >>> 3) & 0x3F;
            // /* Update number of bits */
            if ((count[0] += (inputLen << 3)) < (inputLen << 3))
                    count[1]++;
```

```
            count[1] += (inputLen >>> 29);
            partLen = 64 - index;
            // Transform as many times as possible.
            if (inputLen >= partLen) {
                    md5Memcpy(buffer, inbuf, index, 0, partLen);
                    md5Transform(buffer);
                    for (i = partLen; i + 63 < inputLen; i += 64) {
                            md5Memcpy(block, inbuf, 0, i, 64);
                            md5Transform (block);
                    }
                    index = 0;
            } else
                    i = 0;
            ///* Buffer remaining input */
            md5Memcpy(buffer, inbuf, index, i, inputLen - i);
    }
    /*md5Final 整理和填写输出结果*/
    private void md5Final () {
            byte[] bits = new byte[8];
            int index, padLen;
            ///* Save number of bits */
            Encode (bits, count, 8);
            ///* Pad out to 56 mod 64.
            index = (int)(count[0] >>> 3) & 0x3f;
            padLen = (index < 56) ? (56 - index) : (120 - index);
            md5Update (PADDING, padLen);
            ///* Append length (before padding) */
            md5Update(bits, 8);
            ///* Store state in digest */
            Encode (digest, state, 16);
    }
    /* md5Memcpy 是一个内部使用的 byte 数组的块复制函数，从 input 的 inpos 开始把 len 长度的
       字节拷贝到 output 的 outpos 位置开始*/
    private void md5Memcpy (byte[] output, byte[] input,
            int outpos, int inpos, int len) {
            int i;
            for (i = 0; i < len; i++)
                    output[outpos + i] = input[inpos + i];
    }
    /*md5Transform 是 MD5 核心变换程序，有 md5Update 调用，block 是分块的原始字节*/
    private void md5Transform (byte block[]) {
            long a = state[0], b = state[1], c = state[2], d = state[3];
            long[] x = new long[16];
            Decode (x, block, 64);
            /* Round 1 */
            a = FF (a, b, c, d, x[0], S11, 0xd76aa478L); /* 1 */
            d = FF (d, a, b, c, x[1], S12, 0xe8c7b756L); /* 2 */
            c = FF (c, d, a, b, x[2], S13, 0x242070dbL); /* 3 */
            b = FF (b, c, d, a, x[3], S14, 0xc1bdceeeL); /* 4 */
            a = FF (a, b, c, d, x[4], S11, 0xf57c0fafL); /* 5 */
```

```
d = FF (d, a, b, c, x[5], S12, 0x4787c62aL); /* 6 */
c = FF (c, d, a, b, x[6], S13, 0xa8304613L); /* 7 */
b = FF (b, c, d, a, x[7], S14, 0xfd469501L); /* 8 */
a = FF (a, b, c, d, x[8], S11, 0x698098d8L); /* 9 */
d = FF (d, a, b, c, x[9], S12, 0x8b44f7afL); /* 10 */
c = FF (c, d, a, b, x[10], S13, 0xffff5bb1L); /* 11 */
b = FF (b, c, d, a, x[11], S14, 0x895cd7beL); /* 12 */
a = FF (a, b, c, d, x[12], S11, 0x6b901122L); /* 13 */
d = FF (d, a, b, c, x[13], S12, 0xfd987193L); /* 14 */
c = FF (c, d, a, b, x[14], S13, 0xa679438eL); /* 15 */
b = FF (b, c, d, a, x[15], S14, 0x49b40821L); /* 16 */
/* Round 2 */
a = GG (a, b, c, d, x[1], S21, 0xf61e2562L); /* 17 */
d = GG (d, a, b, c, x[6], S22, 0xc040b340L); /* 18 */
c = GG (c, d, a, b, x[11], S23, 0x265e5a51L); /* 19 */
b = GG (b, c, d, a, x[0], S24, 0xe9b6c7aaL); /* 20 */
a = GG (a, b, c, d, x[5], S21, 0xd62f105dL); /* 21 */
d = GG (d, a, b, c, x[10], S22, 0x2441453L); /* 22 */
c = GG (c, d, a, b, x[15], S23, 0xd8a1e681L); /* 23 */
b = GG (b, c, d, a, x[4], S24, 0xe7d3fbc8L); /* 24 */
a = GG (a, b, c, d, x[9], S21, 0x21e1cde6L); /* 25 */
d = GG (d, a, b, c, x[14], S22, 0xc33707d6L); /* 26 */
c = GG (c, d, a, b, x[3], S23, 0xf4d50d87L); /* 27 */
b = GG (b, c, d, a, x[8], S24, 0x455a14edL); /* 28 */
a = GG (a, b, c, d, x[13], S21, 0xa9e3e905L); /* 29 */
d = GG (d, a, b, c, x[2], S22, 0xfcefa3f8L); /* 30 */
c = GG (c, d, a, b, x[7], S23, 0x676f02d9L); /* 31 */
b = GG (b, c, d, a, x[12], S24, 0x8d2a4c8aL); /* 32 */
/* Round 3 */
a = HH (a, b, c, d, x[5], S31, 0xfffa3942L); /* 33 */
d = HH (d, a, b, c, x[8], S32, 0x8771f681L); /* 34 */
c = HH (c, d, a, b, x[11], S33, 0x6d9d6122L); /* 35 */
b = HH (b, c, d, a, x[14], S34, 0xfde5380cL); /* 36 */
a = HH (a, b, c, d, x[1], S31, 0xa4beea44L); /* 37 */
d = HH (d, a, b, c, x[4], S32, 0x4bdecfa9L); /* 38 */
c = HH (c, d, a, b, x[7], S33, 0xf6bb4b60L); /* 39 */
b = HH (b, c, d, a, x[10], S34, 0xbebfbc70L); /* 40 */
a = HH (a, b, c, d, x[13], S31, 0x289b7ec6L); /* 41 */
d = HH (d, a, b, c, x[0], S32, 0xeaa127faL); /* 42 */
c = HH (c, d, a, b, x[3], S33, 0xd4ef3085L); /* 43 */
b = HH (b, c, d, a, x[6], S34, 0x4881d05L); /* 44 */
a = HH (a, b, c, d, x[9], S31, 0xd9d4d039L); /* 45 */
d = HH (d, a, b, c, x[12], S32, 0xe6db99e5L); /* 46 */
c = HH (c, d, a, b, x[15], S33, 0x1fa27cf8L); /* 47 */
b = HH (b, c, d, a, x[2], S34, 0xc4ac5665L); /* 48 */
/* Round 4 */
a = II (a, b, c, d, x[0], S41, 0xf4292244L); /* 49 */
d = II (d, a, b, c, x[7], S42, 0x432aff97L); /* 50 */
c = II (c, d, a, b, x[14], S43, 0xab9423a7L); /* 51 */
b = II (b, c, d, a, x[5], S44, 0xfc93a039L); /* 52 */
```

```
            a = II (a, b, c, d, x[12], S41, 0x655b59c3L); /* 53 */
            d = II (d, a, b, c, x[3], S42, 0x8f0ccc92L); /* 54 */
            c = II (c, d, a, b, x[10], S43, 0xffeff47dL); /* 55 */
            b = II (b, c, d, a, x[1], S44, 0x85845dd1L); /* 56 */
            a = II (a, b, c, d, x[8], S41, 0x6fa87e4fL); /* 57 */
            d = II (d, a, b, c, x[15], S42, 0xfe2ce6e0L); /* 58 */
            c = II (c, d, a, b, x[6], S43, 0xa3014314L); /* 59 */
            b = II (b, c, d, a, x[13], S44, 0x4e0811a1L); /* 60 */
            a = II (a, b, c, d, x[4], S41, 0xf7537e82L); /* 61 */
            d = II (d, a, b, c, x[11], S42, 0xbd3af235L); /* 62 */
            c = II (c, d, a, b, x[2], S43, 0x2ad7d2bbL); /* 63 */
            b = II (b, c, d, a, x[9], S44, 0xeb86d391L); /* 64 */
            state[0] += a;
            state[1] += b;
            state[2] += c;
            state[3] += d;
    }
    /*Encode 把 long 数组按顺序拆成 byte 数组，因为 java 的 long 类型是 64bit 的，
      只拆低 32bit，以适应原始 C 实现的用途*/
    private void Encode (byte[] output, long[] input, int len) {
            int i, j;
            for (i = 0, j = 0; j < len; i++, j += 4) {
                    output[j] = (byte)(input[i] & 0xffL);
                    output[j + 1] = (byte)((input[i] >>> 8) & 0xffL);
                    output[j + 2] = (byte)((input[i] >>> 16) & 0xffL);
                    output[j + 3] = (byte)((input[i] >>> 24) & 0xffL);
            }
    }
    /*Decode 把 byte 数组按顺序合成成 long 数组，因为 java 的 long 类型是 64bit 的，
      只合成低 32bit，高 32bit 清零，以适应原始 C 实现的用途*/
    private void Decode (long[] output, byte[] input, int len) {
            int i, j;
            for (i = 0, j = 0; j < len; i++, j += 4)
                    output[i] = b2iu(input[j]) |
                            (b2iu(input[j + 1]) << 8) |
                            (b2iu(input[j + 2]) << 16) |
                            (b2iu(input[j + 3]) << 24);
            return;
    }
    /*b2iu 是我写的一个把 byte 按照不考虑正负号的原则的 " 升位 " 程序，因为 java 没有 unsigned 运算*/
    public static long b2iu(byte b) {
            return b < 0 ? b & 0x7F + 128 : b;
    }
    /*byteHEX()，用来把一个 byte 类型的数转换成十六进制的 ASCII 表示，
     因为 java 中的 byte 的 toString 无法实现这一点，又没有 C 语言中的
      sprintf(outbuf,"%02X",ib) */
    public static String byteHEX(byte ib) {
            char[] Digit = { '0','1','2','3','4','5','6','7','8','9',
            'A','B','C','D','E','F' };
            char [] ob = new char[2];
```

```
                ob[0] = Digit[(ib >>> 4) & 0X0F];
                ob[1] = Digit[ib & 0X0F];
                String s = new String(ob);
                return s;
        }
        public static String toMD5(String source){
                MD5 md5 = new MD5();
                return md5.getMD5ofStr(source);
        }
}
```

程序说明：

（1）从安全性考虑，特别是安全性要求特别高的应用系统来说，安全问题是至关重要的。所以这里也需要使用 MD5 算法先对密码进行加密，然后把加密之后的密码存储在数据库中，可以防止密码在数据库中被盗窃。

（2）使用 MD5 算法对密码加密之后，会得到一个惟一的字符串编码进行标识，并且是不可逆转的，即加密后的字符串编码是不能再转换到原有的密码，这样就可以保证了存储在数据库中的密码的安全性。

（3）由于用户注册的密码被 MD5 算法进行了加密，保存在数据库中的其实是密码的一个字符串编码标识，而不是真正的原始密码。所以在用户身份校验时，不能直接把用户登录密码和数据库中存储的密码标识符进行比较，需要把登录密码也要进行 MD5 的转换，然后再可以进行比较。由于每一个密码（或者字符串）对应着惟一的一个字符串编码标识（由 MD5 算法得到的），所以可以进行以上的比较。

（4）直接调用 MD5 类中的 toMD5(String source)方法对需要加密的字符串进行加密。

14.3　实现登录验证码

登录一些网站时，除了要求输入用户名和密码之外，系统还要求用户输入随机生成的验证码，这是为了防止有些机器人软件会自动破解密码。

这些验证码一般都是生成的图片显示出来了，加上在图片同时生成很多条不规则的线条或者图案来加强干扰，使得软件很难识别出案上的验证码。

这一小节将在用户登录页面中添加验证码功能。需要创建一个生成随机验证码图片的 JSP 文件 createMa.jsp。该文件的具体的源代码如下：

```
<%@ page contentType="image/jpeg" %>
<%@ page import="java.awt.*,java.awt.image.*,java.util.*,javax.imageio.*"%>
<%!
Color getRandColor(int fc,int bc){              //在确定的范围中获得随机颜色
        Random random = new Random();
        if(fc>255) fc=255;
        if(bc>255) bc=255;
        int r=fc+random.nextInt(bc-fc);
        int g=fc+random.nextInt(bc-fc);
        int b=fc+random.nextInt(bc-fc);
        return new Color(r,g,b);
        }
%>
<%                                              //设置页面不缓存
```

```
response.setHeader("Pragma","No-cache");
response.setHeader("Cache-Control","no-cache");
response.setDateHeader("Expires", 0);
                                        //在内存中创建图像
int width=60, height=20;
BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
                                        //获取图形上下文
Graphics g = image.getGraphics();
                                        //生成随机类
Random random = new Random();
                                        //设定背景色
g.setColor(getRandColor(200,250));
g.fillRect(0, 0, width, height);
                                        //设定字体
g.setFont(new Font("Times New Roman",Font.PLAIN,18));
                                        //随机产生 50 条干扰线，使图像中的认证码不易被其他程序探测到
g.setColor(getRandColor(160,200));
for (int i=0;i<50;i++){
      int x = random.nextInt(width);
      int y = random.nextInt(height);
  int xl = random.nextInt(width);
  int yl = random.nextInt(height);
      g.drawLine(x,y,x+xl,y+yl);
}
                                        //取随机产生的认证码(4 位数字)
String sRand="";
for (int i=0;i<4;i++){
     String rand=String.valueOf(random.nextInt(10));
     sRand+=rand;
                                        //将认证码显示到图像中
     g.setColor(new Color(20+random.nextInt(110),20+random.nextInt(110),20+random.nextInt(110)));
                                        //调用函数出来的颜色相同，可能是因为种子太接近，所以只能直接生成
     g.drawString(rand,13*i+6,16);
}
                                        //将认证码存入 SESSION
session.setAttribute("rand",sRand);
                                        //图像生效
g.dispose();
                                        //输出图像到页面
ImageIO.write(image, "JPEG", response.getOutputStream());
%>
```

程序说明：

- <%!　　%>内部进行对变量和方法的声明，这里声明了 getRandColor()方法。
- getRandColor()方法获得一个在给定范围内的颜色。
- 此处需要把页面设置不缓存。
- 首先在内存中创建一个图像对象，对图像设置背景色以及随机的 50 条干扰线，使得验证码不易被其他程序检测到，然后生成四位随机验证码并把它显示到生成的图像中。最后输出图像到页面。

- contentType 属性告诉容器这个 JSP 输出为何种格式，这里设置为"image/jpeg"，说明输出为图片格式。

14.4　修改JSP页面

创建了以上的所有 JavaBean 类之后，接下来修改在第 12 章中已经创建的 JSP 页面代码，以实现在这些 JSP 程序中对以上 JavaBean 的调用。

14.4.1　修改首页index.jsp

由于 DBConnect.java 类已经封装了对数据库的连接和基本操作，所以在该 JSP 页面中就不需要出现数据库连接代码。另外获取所有商品的信息，也可以直接调用 SqlProduct 类中的 getProducts()方法来获取。该 JSP 页面代码修改如下：

```
<%@ page import=”cn.register.product.SqlProduct” %>
<%@ page import=”cn.register.product.Product” %>
<%@ page import=”cn.register.Factory” %>
<%@ page import=”cn.register.Iterator” %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<title>注册登录系统首页</title>
<link rel="stylesheet" href="register.css" type="text/css"/>
</head>
<%
String UserId = (String)session.getAttribute("user");              //从 session 中获取用户 ID
if(UserId == null || UserId == "")                                  //判断用户 ID 是否为空，即判断是否登录
       response.sendRedirect("login.jsp");                          //未登录则跳转到 login.jsp 登录页面

Product product = null;
SqlProduct sqlProduct = Factory.getInstance().getSqlProduct();//初始化 SqlProduct 类
Iterator iterator = sqlProduct.getProducts();                       //该方法返回 Iterator“反复器”数据类型

String product_name = "";                                           //定义产品名称变量
float price;                                                        //定义产品价格变量
String description = "";                                            //定义产品描述变量

%>
<body>
<table width="80%" align="center">
  <tr>
    <td class="title"> 用户注册登录系统==<span class="title1">产品展示</span>==</td>
    <td style="width:150; height:20; vertical-align:middle; text-align:center;"><%=UserId%>  您好!  <a href="login.jsp">登录</a> | <a href="logout.jsp">注销</a></td>
  </tr>
</table>
<hr align="center" width="75%" color="#990000" size="1"/>
```

```
<table width="80%" align="center">
  <tr><td height="5" colspan="2"></td></tr>
 <%
     while(iterator.hasNext()){                                          //循环输出产品信息
     product = (Product)iterator.next();
      product_name = product.getProduct_name();
      price = product.getPrice();
      description = product.getDescription();
 %>
  <tr>
     <td> 产品名称：<%=product_name%></td><td width="40%"> 报价：<span style="color:#FF0000;
font-style:italic;"><%=price%></span></td>
  </tr>
  <tr>
     <td colspan="2"> 描述：<%=description%></td>
  </tr>
  <tr><td height="5" colspan="2"><hr size="3" width="10%"/></td></tr>
<%
   }
%>
</table>
<div align="center" class="tail">2006==本公司版权拥有</div>
</body>
</html>
```

程序说明：使用 page 指令中的 import 属性来加载使用到的 JavaBean 类文件。

14.4.2　修改用户注册处理页面do_register.jsp

该 JSP 页面修改之后的程序代码如下：

```
<%@ page import="cn.register.DateFormat" %>
<%@ page import="cn.register.user.User" %>
<%@ page import="cn.register.user.SqlUser" %>
<%@ page import=”cn.register.Factory” %>
<%@ page import=”cn.register.MD5” %>
<%
// 获取上一页面传递过来的参数
String ID = request.getParameter("ID");
String password = MD5.toMD5(request.getParameter("password"));
String rpassword = request.getParameter("rpassword");
String name = request.getParameter("name");
String sex = request.getParameter("sex");
String year = request.getParameter("year");
          if(year.length ==1) year = "0"+year;
String mouth = request.getParameter("mouth");
          if(mouth.length == 1) mouth = "0"+mouth;
String day = request.getParameter("day");
          if(day.length == 1) day =   "0"+day;
String date = year+"-"+mouth+"-"+day;
String description = request.getParameter("description");
```

```
User user = Factory.getInstance().initUser();                    //初始化 User 实例
SqlUser sqlUser = Factory.getInstance().initSqlUser();           //初始化 SqlUser 实例
                        //调用 DateFormat 类中 getStr2Long()方法实现字符串类型日期转变成 Long 类型日期
long birth = DateFormat.getStr2Long(date);
If(SqlUser.checkRPwd(password,rpassword)){                       //如果两次密码输入正确
      user.setUser_id(ID);
      user.setPassword(password);
      user.setName(name);
      user.setSex(sex);
      user.setBirth(birth);
      user.setDescription(description);
      if(sqlUser.saveUser(user)){                                //将用户注册信息保存到数据库中，如果成功
            session.setAttribute("user",ID);
            response.sendRedirect("index.jsp");
      }else{                                                     //否则数据库操作失败
            response.sendRedirect("register.jsp");
      }
}else{                                                           //两次密码不符合
      response.sendRedirect("register.jsp");
}
%>
```

程序说明：该页面中使用到了 DateFormat 类中的 getStr2Long()方法来对日期类型进行转换。SqlUser 类中的 checkRPwd()方法来对用户输入的两次密码进行校对，saveUser()方法将用户注册信息保存到后台数据库中。另外保存的用户密码需要先调用 MD5 类中的 toMD5()方法进行编码操作。

14.4.3　修改登录页面login.jsp

修改 logon.jsp 页面，在<head></head>之间添加如下代码使得页面不缓存：

```
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="0">
```

在<%　%>内部的 if else 语句中再添加如下一段代码，当验证码输入有误时打印出提示：

```
  else if(info.equals("3"))
     out.println("<font size=4 color='red'>验证码不正确，请重新登陆！</font><br><br><br>");
```

然后再在输入密码的表格行下面添加一个输入验证码的行，代码如下：

```
<tr><td width="56"><img src="../images/userym2.gif"></td><td width="113"><input type="text" name="chkcode"
size="4"> <img border=1 src="../include/image.jsp"></td></tr>
```

14.4.4　修改登录处理页面chek_login.jsp

该页面代码修改如下：

```
<%@ page import=”cn.register.User.SqlUser” %>
<%@ page import=”cn.register.Factory” %>
<%
String ID = request.getParameter("ID");
String password = request.getParameter("password");
```

```
String chkcode = request.getParameter("chkcode");          //取得 chkcode 参数值
String rand = (String)session.getAttribute("rand");        //取得 rand 参数值

SqlUser sqlUser = Factory.getInstance().initSqlUser();
String index = checkUser(ID,password);
if(chkcode.equals(rand)){
      If(index == 1){
            response.sendRedirect("login.jsp?info=1");;
      }else if(index ==2){
            response.sendRedirect("login.jsp?info=2");
      }else{
            session.setAttribute("user",ID);
            response.sendRedirect("index.jsp");
      }
}else{
      response.sendRedirect("index.jsp?info=3");
}
%>
```

程序说明：由于该实例中添加了验证码功能，需要在进行用户身份验证之前，需要检验用户输入的验证码是否正确。

至此，完成了该实例的修改，重新运行该实例，会出现十二章中运行效果。惟一不同的是多了一个验证码功能。

14.5　本章小结

本章使用 JavaBean 重新实现了第 12 章中的实例。在代码演示过程中，读者需要学会 JavaBean 类的创建以及如何在 JSP 页面进行调用。另外，该实例中添加了验证码功能，以及为了安全使用 MD5 对密码首先要进行编码。学完本章之后，读者应该对整个 JavaBean 的概念有所了解。