第 5 章　JavaBean 技术

JavaBean 是一个遵循特定写法的 Java 类。在 Java 模型中，通过 JavaBean 可以无限扩充 Java 程序的功能，通过 JavaBean 的组合可以快速生成新的应用程序。JavaBean 的产生使 JSP 页面中的业务逻辑变得更加清晰，程序之中的实体对象及业务逻辑可以单独封装到 Java 类之中。这样不仅提高了程序的可读性和易维护性，而且还提高了代码的重用性。

本章将主要介绍 JavaBean 的构成，以及不同类型属性的使用和 JavaBean 的应用，并详细介绍了不同作用域中 JavaBean 的生命周期。

本章学习要点：

- 熟练掌握 JavaBean 的构成
- 掌握 JavaBean 中不同类型属性的使用
- 掌握 JavaBean 的编写和部署
- 熟练掌握 JavaBean 在 JSP 页面中的应用
- 熟练掌握 JavaBean 不同作用域的应用

5.1 JavaBean 概述

JavaBean 实质上是一个 Java 类，一个遵循特定规则的类。当用在 Web 程序中时，会以组件的形式出现，并完成特定的逻辑处理功能。

5.1.1 JavaBean 技术介绍

使用 JavaBean 的最大优点就在于它可以提高代码的重用性。编写一个成功的 JavaBean，宗旨为“一次性编写，任何地方执行，任何地方重用”。

1．一次性编写

一个成功的 JavaBean 组件重用时不需要重新编写，开发者只需要根据需求修改和升级代码即可。

2．任何地方执行

一个成功的 JavaBean 组件可以在任何平台上运行，JavaBean 是基于 Java 语言编写的，所以它可以轻易移植到各种运行平台上。

3．任何地方重用

一个成功的 JavaBean 组件能够被在多种方案中使用，包括应用程序、其他组件、Web 应用等。

5.1.2 JavaBean 的分类

JavaBean 按功能可分为可视化 JavaBean 和不可视 JavaBean 两类。可视化 JavaBean 就是具有 GUI 图形用户界面的 JavaBean；不可视 JavaBean 就是没有 GUI 图形用户界面的 JavaBean，最终对用户是不可见的，它更多地是被应用到 JSP 中。

不可视 JavaBean 又分为值 JavaBean 和工具 JavaBean。

（1）值 JavaBean：严格遵循了 JavaBean 的命名规范，通常用来封装表单数据，作为信息的容器，如下面的 JavaBean 类。

```
public class User {
    private String username;    //用户名
    private String password;    //密码
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}
```

（2）工具 JavaBean：可以不遵循 JavaBean 规范，通常用于封装业务逻辑、数据操作等，例如，连接数据库，对数据库进行增、删、改、查和解决中文乱码等操作。工具 JavaBean 可以实现业务逻辑与页面显示的分离，提高了代码的可读性与易维护性，如下面的代码。

```
public class MyTools {
    public String change(String source) {
        source=source.replace("<","&lt;");
        source=source.replace(">","&gt;");
        return  source;
    }
}
```

5.1.3 JavaBean 规范

通常一个标准的 JavaBean 类需要遵循以下规范。

1．实现可序列接口

JavaBean 应该直接或间接实现 java.io.Serializable 接口，以支持序列化机制。

2．公共的无参构造方法

一个 JavaBean 对象必须拥有一个公共类型、默认的无参构造方法，从而可以通过 new 关键字直接对其进行实例化。

3．类的声明是非 final 类型的

当一个类声明为 final 类型时，它是不可以更改的，所以 JavaBean 对象的声明应该是非 final 类型的。

4．为属性声明访问器

JavaBean 中的属性应该设置为私有类型（private），可以防止外部直接访问，它需要提供对应的 set×××()和 get×××()方法来存取类中的属性，方法中的“××××”为属性名称，属性的第一个字母应大写。若属性为布尔类型，则可使用 is×××()方法代替 get×××()方法。

JavaBean 的属性是内部核心的重要信息，当 JavaBean 被实例化为一个对象时，改变它的属性值也就等于改变了这个 Bean 的状态。这种状态的改变常常也伴随着许多数据处理动作，使得其他相关的属性值也跟着发生变化。

实现 java.io.Serializable 接口的类实例化的对象被 JVM（Java 虚拟机）转化为一个字节序列，并且能够将这个字节序列完全恢复为原来的对象，序列化机制可以弥补网络传输中不同操作系统的差异问题。作为 JavaBean，对象的序列化也是必需的。使用一个 JavaBean 时，一般情况下是在设计阶段对它的状态信息进行配置，并在程序启动后期恢复，这种具体工作是由序列化完成的。

【练习 1】

创建一个简单的 JavaBean 类 Student，该类中包含属性 name、age、sex，分别表示学生的姓名、年龄和性别。具体的 Student 类的实现如下。

```
import java.io.Serializable;
public class Student implements Serializable {
    public Student() {                          //无参数的构造函数
        super();
    }
    private String name;                        //学生姓名
    private String sex;                         //学生性别
    private int age;                            //学生年龄
```

```
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getSex() {
        return sex;
    }
    public void setSex(String sex) {
        this.sex = sex;
    }
    public int getAge() {
        return age;
    }
    public void setAge(int age) {
        this.age = age;
    }
}
```

5.2 JavaBean 属性

JavaBean 的属性与一般 Java 程序中所指的属性，或者与面向对象的程序设计语言中对象的属性是一个概念，在程序中的具体体现就是类中的变量。在 JavaBean 设计中，按照属性的不同作用又可分为 4 类：Simple（简单）属性、Indexed（索引）属性、Bound（关联）属性和 Constrained（限制）属性。

5.2.1 Simple 属性

Simple 属性就是在 JavaBean 中对应了简单的 set×××()和 get×××()方法的变量，在创建 JavaBean 时，简单属性最为常用。

在 JavaBean 中，简单属性的 get×××()与 set×××()方法形式如下。

```
public void set×××(type value);
public type get×××();
```

而对于 Boolean 类型的属性，则应使用 is×××()和 set×××()方法，其形式如下。

```
public void set×××(boolean value){…}
public boolean is××× (){…}
```

【练习 2】

创建一个 JavaBean 类，在该类中分别定义一个 String 类型的 name 属性和一个 Boolean 类型的 role 属性，分别表示用户的姓名和角色。当 role 属性的值为 True 时，表示为管理员角色，否则为普通用户。该 JavaBean 的定义如下。

```
public class User {
    private String name;                          //用户的姓名
    private boolean role;                         //用户的角色
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public boolean isRole() {
        return role;
    }
    public void setRole(boolean role) {
        this.role = role;
    }
}
```

一般将属性的访问权限设为 private，这样可以避免使用者直接通过访问属性修改其值。如果为属性提供了对应的 get×××()方法，表示该属性是可读的；如果提供了对应的 set×××()方法，则表示该属性是可修改的。如果某个属性是不可修改的，则不提供该属性的 set×××()方法即可。

5.2.2 Indexed 属性

一个 Indexed 属性表示一个数组值，需要通过索引访问的属性通常称为索引属性。如存在一个大小为 3 的字符串数组，若要获取该字符串数组中指定位置中的元素，需要得知该元素的索引。

在 JavaBean 中，索引属性的 get×××()与 set×××()方法形式如下。

```
public void set×××(type[ ] value);
public type[ ] get×××();
public void set×××(int index,type value);
public type get×××(int index);
```

【练习 3】

对于一个班级来说，可能有多个名称、多个学生。下面就来创建一个班级的 JavaBean 类，并在该类中分别定义一个数组类型和一个 List 类型的属性，同时要为其提供对应的 get×××()和 set×××()方法，代码如下。

```
import java.util.ArrayList;
import java.util.List;
public class Classes {
    private String[] names = new String[3];     //定义 String 类型的数组
```

```
    private List<Student> students = new ArrayList<Student>();
    //定义 List 型数组
    public String[] getNames() {                //获取一个数组
        return names;
    }
    public void setNames(String[] names) {      //为数组赋值
        this.names = names;
    }
    public String getNames(int index){ //根据索引，获取数组中的某个元素
        return names[index];
    }
    public void setNames(int index , String name){
    //为数组中的某个元素赋值
        this.names[index] = name;
    }
    public List<Student> getStudents() {        //获取一个集合
        return students;
    }
    public void setStudents(List<Student> students) {  //为集合赋值
        this.students = students;
    }
    public Student getStudents(int index){ //根据索引，获取集合中的某个元素
        return students.get(index);
    }
    public void setStudents(int index , Student student){
    //为集合中的某个元素赋值
        this.students.set(index, student);
    }
}
```


5.2.3 Bound 属性

如果在 Simple 或 Indexed 属性上添加一种监听机制，即当某个属性值发生改变时通知监听器，则这个属性属于 Bound 属性。监听器需要实现 java.beans.PropertyChangeListener 接口，负责接收由 JavaBean 组件产生的 java.beans.PropertyChangeEvent 对象，在该对象中包含发生改变的属性名、改变前后的值，以及每个监听器可能要访问的新属性值。

JavaBean 还需要实现 addPropertyChangeListener()方法和 removePropertyChangeListener()方法，以添加和取消属性变化监听器。这两个方法的定义如下。

```
void addPropertyChangeListener(PropertyChangeListener listener);
void removePropertyChangeListener(PropertyChangeListener listener);
```

除此之外，还可以通过 java.beans.PropertyChangeSupport 类来管理监听器。通常情况下，使用该类的实例作为 JavaBean 的成员字段，并将各种工作委托给它。

PropertyChangeSupport 类的构造方法及主要方法如下。

```
public PropertyChangeSupport(Object paramObject)
public void addPropertyChangeListener(PropertyChangeListener
paramPropertyChangeListener)
public void removePropertyChangeListener(PropertyChangeListener
paramPropertyChangeListener)
public void firePropertyChange(String paramString, Object paramObject1,
Object paramObject2)
```

如上述代码所示，在 PropertyChangeSupport 类中主要有三个方法，其中，addPropertyChangeListener()方法表示在监听者列表中加入一个 PropertyChangeListener 监听器；removePropertyChangeListener() 方法表示从监听者列表中删除一个 PropertyChangeListener 监听器；firePropertyChange()方法表示通知用于更新任何注册监听者的一个绑定属性，若改变前的值和改变后的值相等且为非空，则不激发事件。

Bound 属性通常情况下在实现 Java 图形编程的 JavaBean 中大量使用，在开发 JSP 的过程中很少用到。

5.2.4 Constrained 属性

Constrained 属性是在 Bound 属性的基础上添加了一个约束条件，即当某个监听器检测到某个属性值发生改变后，需要由所有的监听器验证通过才能够修改该属性值。只要有一个监听器否决了该属性的变化，值不能被修改。监听器需要实现 java.beans.VetoableChangeListener 接口，该接口负责接收由 JavaBean 组件产生的 java.beans.PropertyChangeEvent 对象，JavaBean 组件可以通过 java.beans. VetoableChangeSupport 类激活由监听器接收的实际事件。

JavaBean 还需要实现 addVetoableChangeListener() 方法和 removeVetoableChangeListener()方法，以便添加和取消可否决属性变化的监听器。这两个方法的一般定义如下。

```
void addVetoableChangeListener(VetoableChangeListener listener);
void removeVetoableChangeListener(VetoableChangeListener listener);
```

除此之外，还可以通过 java.beans.VetoableChangeSupport 类的 fireVetoableChange() 方法传递属性名称、改变前的值和改变后的值等信息。

Constrained 属性通常情况下在实现 Java 图形编程的 JavaBean 中大量使用，在开发 JSP 的过程中很少能用到。

5.3 实验指导 5–1：邮箱验证

Java 是纯面向对象的编程语言，JSP 是以 Java 为脚本语言的动态技术。它也具有面向对象开发模式的先天条件，所以在 JSP 程序设计之中更应该融入面向对象的思维。本节实验指导将通过一个案例，简单介绍在 JSP 页面中使用 JavaBean 对象的应用。

本实验指导通过非可视化 JavaBean 封装邮箱地址对象，通过 JSP 页面调用此对象来验证邮箱是否合法。其步骤如下。

（1）创建 com.itzcn.bean.Email 类，定义两个属性，代码如下。

```
import java.io.Serializable;
public class Email implements Serializable {
    private static final long serialVersionUID = 1L;
    private String mail = null;             //Email 地址
    private boolean isMail = false;         //是否为一个标准的 Email 地址
    public Email() {                        //默认无参数的构造函数
        super();
    }
    public Email(String mail) {             //参数为 mail 的构造方法
        this.mail = mail;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
        this.mail = mail;
    }
    public boolean isMail() {
        String regex =  "^([a-z0-9A-Z]+[-|\\.]?)+[a-z0-9A-Z]@
        ([a-z0-9A-Z]+(-[a-z0-9A-Z]+)?\\.)+[a-zA-Z]{2,}$";//正则表达式
        if (mail.matches(regex)) {
            isMail = true;
        }
        return isMail;
    }
    public void setMail(boolean isMail) {
        this.isMail = isMail;
    }
}
```

（2）创建 index.jsp 页面，用于放置验证邮箱的表单。此表单的提交地址为 result.jsp，主要代码如下。

```
<form method="post" id="searchform" action="result.jsp">
    <fieldset>
        <input type="text" name="mail" id="searchtext" />
        <input type="submit" id="searchsubmit" value="验证" />
    </fieldset>
</form>
```

（3）创建 result.jsp 页面，处理 index.jsp 页面中提交来的表单，在其中实例化 Email 对象，验证邮箱地址。并将验证结果输出到页面中，其主要代码如下。

```
<h2>邮箱认证系统</h2>
<%
    String mail = request.getParameter("mail");
    Email email = new Email(mail);
    if(email.isMail()){
     out.print(mail + "<br>是一个标准的邮箱地址<br>");
    }else{
     out.print(mail + "<br>不是一个标准的邮箱地址<br>");
    }
%>
<a href="index.jsp">[返回]</a>
```


（4）运行 index.jsp 文件，邮箱认证页面运行效果如图 5-1 所示。

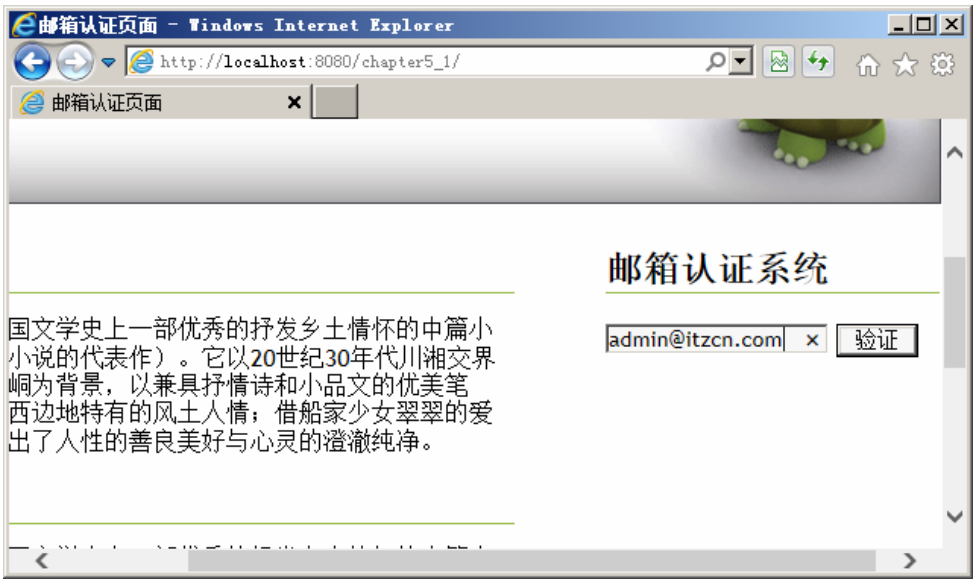

图 5-1 邮箱认证页面

输入标准的邮箱地址，如“admin@itzcn.com”，单击验证，其邮箱认证结果如图 5-2 所示。

输入不标准的邮箱地址，如“admin@itzcn”，单击验证，其邮箱认证结果如图 5-3 所示。

图 5-2 标准的邮箱地址的验证结果

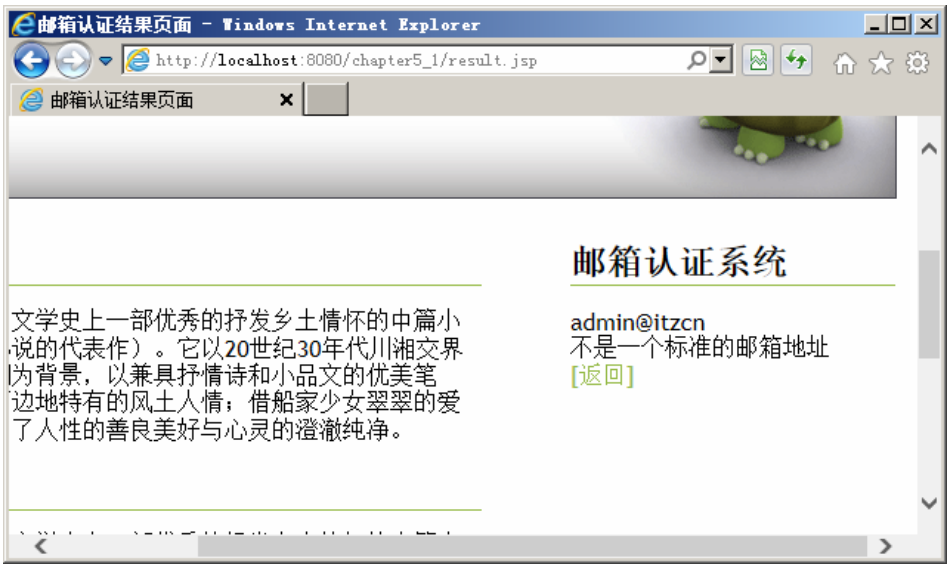

图 5-3 不标准的邮箱地址的验证结果

提示

在该实验指导中所讲述的在 JSP 页面中使用 JavaBean 时，使用的是脚本语言，即在<%与%>之间编写代码，实例化 JavaBean 类，并调用该 JavaBean 类中的属性。其实，JSP 对于在 Web 应用中集成 JavaBean 组件提供了完善的支持，即提供了<jsp:useBean>、<jsp:getProperty>和<jsp:setProperty>标签，使在 JSP 页面中使用 JavaBean 不再需要编写大量的代码。这种支持不仅能缩短开发时间，也为 JSP 应用带来了更多的可扩展性。这在第 2 章中已经介绍了，这里就不再赘述了。

5.4 JavaBean 作用域范围

JavaBean 的功能很强大，不仅可以通过 JavaBean 组件封装许多信息，而且还可以将一些数据处理的逻辑代码隐藏在 JavaBean 内部。为使 JavaBean 更好地服务，需要设置它的作用域，通常使用<jsp:useBean>标签中的 scope 关键字来设定 JSP 页面的生命周期和使用范围。

5.4.1 JavaBean 的作用域简介

在 JSP 页面中有 4 种范围：Page、Request、Session 和 Application。在这里同样可以设定 JavaBean 的作用域，它与 JSP 页面的范围名称相同且意义相同，这 4 种作用域如表 5-1 所示。

表 5-1 JavaBean 在 JSP 中的作用域

作用域	说明
Page	与当前页面相对应，JavaBean 的生命周期存在于一个页面之中，当页面关闭时 JavaBean 被销毁
Request	与 JSP 的 Request 生命周期相对应，JavaBean 的生命周期存在于 request 对象之中，当 request 对象销毁时 JavaBean 也被销毁
Session	与 JSP 的 session 生命周期相对应，JavaBean 的生命周期存在于 session 会话之中，当 session 超时或会话结束时 JavaBean 被销毁
Application	与 JSP 的 application 生命周期相对应，在各个用户与服务器之间共享，只有当服务器关闭时 JavaBean 才被销毁

这 4 种作用范围与 JavaBean 的生命周期是息息相关的。当 JavaBean 被创建后，通过<jsp:setProperty>标签与<jsp:getProperty>标签调用时，将会按照 Page、Request、Session 和 Application 的顺序来查找这个 JavaBean 实例，直至找到一个实例对象为止。如果在这 4 个范围内都找不到 JavaBean 实例，则抛出异常。

5.4.2 Page 作用域

设定一个 JavaBean 的作用域为当前 JSP 页面，通常使用如下的代码。

```
<jsp:useBean id="myBean" class="com.itzcn.bean.MyBean" scope="page"/>
```

如果一个 JavaBean 的 scope 属性设定为 Page，那么它的作用域在这 4 种类型中范围是最小的，客户端每次请求访问时都会创建一个 JavaBean 对象。JavaBean 对象的有效范围是客户请求访问的当前页面文件，当客户执行当前的页面文件完毕后，JavaBean 对象的生命周期结束。

在 Page 范围内，每次访问页面文件时都会生成新的 JavaBean 对象，原有的 JavaBean 对象已经结束生命周期。

【练习 4】

下面以一个登录计数器为例，说明 Page 在 JSP 页面中的作用域。操作步骤如下。

（1）创建一个名称为 MyBean 的 JavaBean 组件，主要代码如下。

```
public class MyBean {
    private String name = null;//用户名
    private String pass = null;//密码
    private int count = 0;      //登录次数
//省略 getter、setter 方法
}
```

（2）创建登录页面 login1.jsp，在该页面中使用 JavaBean 组件，设置其作用域为 Page，代码如下。

```
<jsp:useBean id="myBean" class="com.itzcn.bean.MyBean" scope="page">
</jsp:useBean>
```

（3）在 login1.jsp 页面中添加一个表单，用于登录，方法为 post，地址为 login1.jsp，主要代码如下。

```
<FORM name="form1" action="login1.jsp" method="post">
<UL>
  <LI><LABEL>用户名：<INPUT id=UserName onblur="this.className=
  'input_onBlur'"
  onfocus="this.className='input_onFocus'" name="name"><INPUT id=act
  type=hidden value=cool name=act> </LABEL>
  <LI><LABEL>密　码：<INPUT id=Password onblur="this.className=
  'input_onBlur'"
  onfocus="this.className='input_onFocus'" type=password name="pass">
  </LABEL>
  </LABEL>
  <LI class=CookieDate><LABEL for=CookieDate><INPUT id=CookieDate type=
  checkbox
  value=3 name=CookieDate>保存我的登录信息</LABEL>
  <LI><INPUT type=hidden name=fromurl><INPUT id=Submit onclick="return
  CheckForm();" type=submit value=登　录 name=Submit><A
  href="http://www.itzcn.com">忘记密码？</A>
  <LI class=hr>
  <LI>如果你不是本站会员，请注册
  <LI class=regbt><A href="http://www.itzcn.com"><IMG  src="css/reg.
  jpg"></A> </LI></UL></FORM>
```

（4）在表单中加入用于显示登录次数的代码，代码如下。

```
<%
    if (request.getParameter("name") != null) {
%>
```

```
        <jsp:setProperty property="*" name="myBean" />
        【<jsp:getProperty property="name" name="myBean" />】
            用户已登录
    <jsp:getProperty property="count" name="myBean" />次
    <%
        }
    %>
```

（5）运行程序，访问 login1.jsp 页面，输入登录信息，单击【登录】按钮，在该页面中显示登录的次数，如图 5-4 所示。

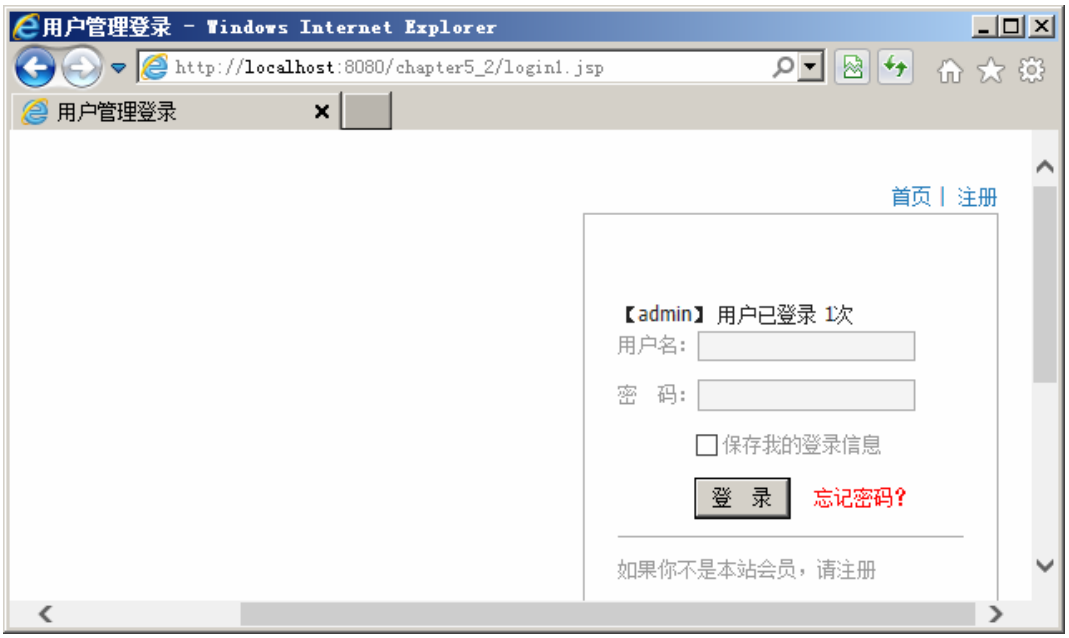

图 5-4 **JavaBean 作用域为 Page 的运行结果**

注意

无论刷新多少次该页面，页面中的登录次数显示值永远为 1，不会递增，这是因为在 login.jsp 页面中设置了 JavaBean 的作用域为 Page，每当用户执行一次刷新操作，JSP 容器会将 Page 范围内的 JavaBean 删除掉，然后再产生一个新的 JavaBean，因此 count 的值会永远保持为 1。

5.4.3 Request 作用域

设定一个 JavaBean 的作用域为 Request，通常使用如下代码。

```
<jsp:useBean id="myBean" class="com.itzcn.bean.MyBean" scope=
"request"/>
```

当 scope 属性为 request 时，JavaBean 对象被创建后，它将存在于整个 Request 的生命周期内。request 对象是一个内置对象，使用它的 getParameter()方法可以获取表单中的数据信息。

Request 范围的 JavaBean 与 request 对象有着很大的关系。它的存取范围除了 page 外，还包括使用动作元素<jsp:forward>和<jsp:include>包含的页面，所有通过这两个操作指令连接在一起的 JSP 程序都可以共享一个 Request 范围的 JavaBean。该 JavaBean 对象使得 JSP 程序之间传递信息更为容易，不过美中不足的是这种 JavaBean 不能用于客户端与服务端之间传递信息，因为客户端没有办法执行 JSP 程序和创建新的 JavaBean 对象。

【练习 5】

下面仍以一个登录计数器为例，说明 Request 在 JSP 页面中的作用域。操作步骤如下。

（1）创建一个名称为 MyBean 的 JavaBean 组件，代码与练习 4 中步骤（1）的代码一致。

（2）创建登录页面 login2.jsp，在该页面中使用 JavaBean 组件，设置其作用域为 Request，代码如下。

```
<jsp:useBean id="myBean" class="com.itzcn.bean.MyBean" scope="request">
</jsp:useBean>
```


（3）在 login2.jsp 页面中添加一个表单，用于登录，方法为 post，地址为 login2.jsp，代码与练习 4 中步骤（3）的代码类似，这里就省略了。

（4）在表单中加入用于显示登录次数的代码，这里代码与练习 4 中的步骤（4）代码一致。

（5）创建页面 success.jsp，主要代码如下。

```
<%
    if (request.getParameter("name") != null) {
%>
<jsp:setProperty property="*" name="myBean" />
<font color="red">【<jsp:getProperty property="name"  name="myBean" />】
用户已登录<jsp:getProperty property="count" name="myBean" />次</font>
 <% }%>
```

（6）在 login2.jsp 中引用 success.jsp，代码如下。

```
<jsp:include page="success.jsp" flush="true"/>
```

（7）运行程序，访问 login2.jsp 页面，输入登录信息，单击【登录】按钮，在该页面中显示登录的次数，如图 5-5 所示。

注 意

如果只运行 success.jsp 页面，页面将不打印任何文字，这是因为在 success.jsp 页面中并没有接收到 login2.jsp 共享的 MyBean 对象，那么 JSP 容器会创建新的 MyBean 对象。

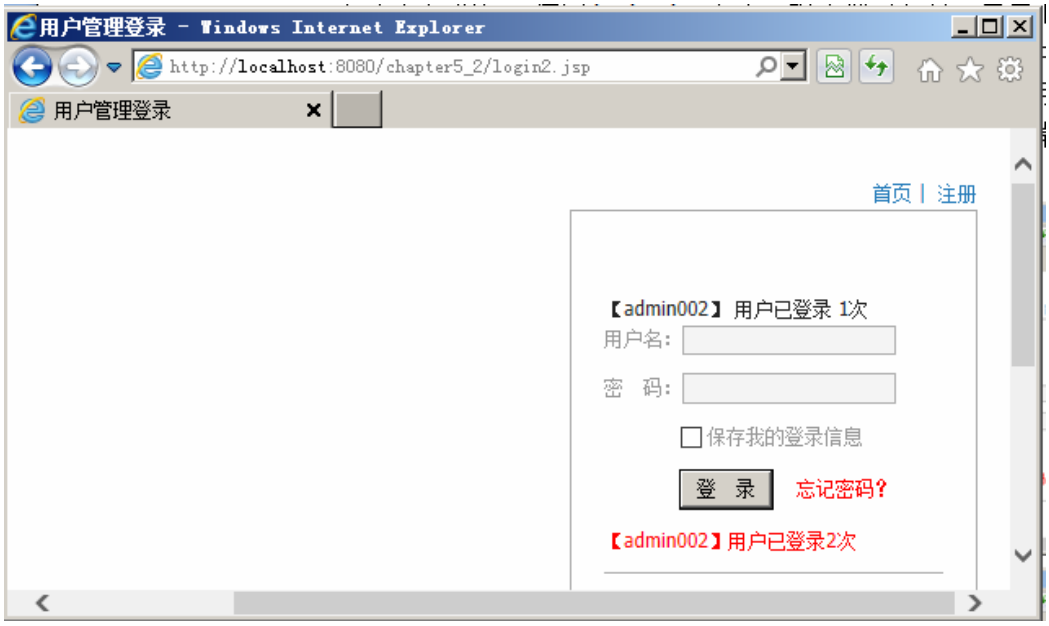

图 5-5 JavaBean 作用域为 Request 时的运行结果

5.4.4 Session 作用域

设定一个 JavaBean 的作用域为 Session，通常使用如下代码。

```
<jsp:useBean id="myBean" scope="session" class="com.itzcn.bean.
MyBean" />
```

当 scope 为 session 时，JavaBean 对象被创建后，它将存在于整个 session 的生命周期内，session 对象是一个内置对象，当用户使用浏览器访问某个页面时，就创建了一个代表该链接的 session 对象，同一个 session 中的文件共享这个 JavaBean 对象。客户对应的 session 生命周期结束时，JavaBean 对象的生命周期也结束。在同一个浏览器中，JavaBean 对象就存在于一个 session 中。当重新打开新的浏览器时，就会开始一个新的 session。每个 session 中拥有各自的 JavaBean 对象。

【练习 6】

下面仍以一个登录计数器为例，说明 Session 在 JSP 页面中的作用域。操作步骤如下。

（1）创建一个名称为 MyBean 的 JavaBean 组件，代码与练习 4 中步骤（1）的代码一致。

（2）创建登录页面 login3.jsp，在该页面中使用 JavaBean 组件，设置其作用域为 Session，代码如下。

```
<jsp:useBean id="myBean" class="com.itzcn.bean.MyBean" scope="session">
</jsp:useBean>
```

（3）在 login3.jsp 页面中添加一个表单，用于登录，方法为 post，地址为 login3.jsp，代码与练习 4 中步骤（3）的代码类似，这里就省略了。

（4）在表单中加入用于显示登录次数的代码，代码如下。

```
<%
    if (request.getParameter("name") != null) {
%>
    <jsp:setProperty property="*" name="myBean" />
        已登录
<jsp:getProperty property="count" name="myBean" />次
<%
    }
%>
```

（5）访问 login3.jsp 页面，多次登录或刷新当前页面，运行结果如图 5-6 所示。

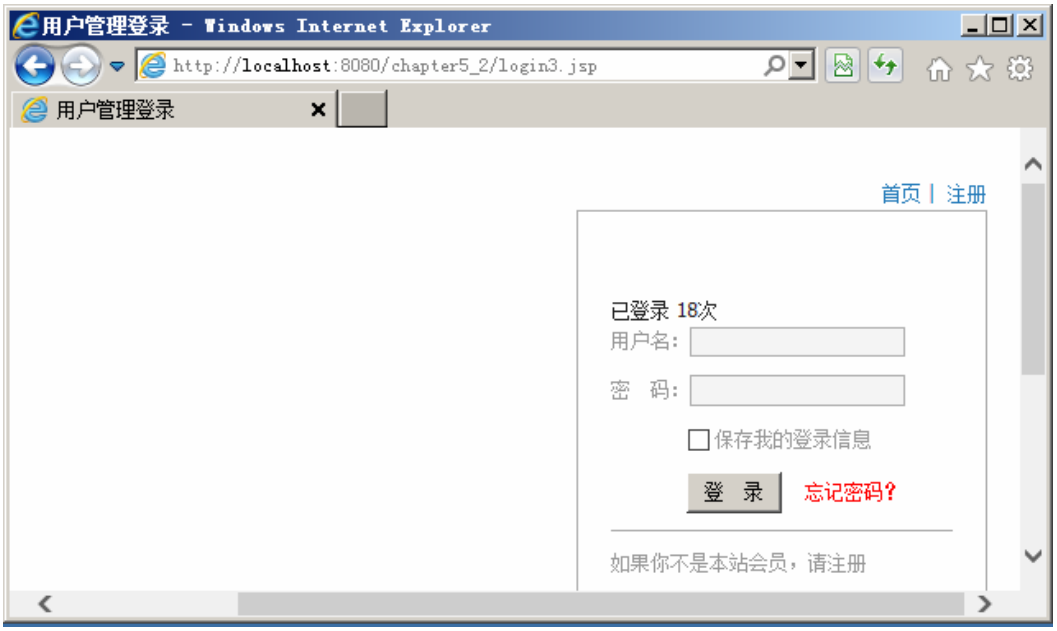

图 5-6 JavaBean 作用域为 Session 时的运行结果

注意

每当单击【登录】按钮（或刷新当前页面）时，登录次数会增加 1，这是因为在 login3.jsp 文件中赋予 JavaBean 的作用域为 Session。但当开启另外一个窗口时，登录次数又会重新开始从 1 递增，由此可知，作用域为 Session 的 JavaBean 对象在浏览器关闭后消亡。

5.4.5 Application 作用域

设定一个 JavaBean 的作用域为 Application，通常使用如下代码。

```
<jsp:useBean id="myBean" class="com.itzcn.bean.MyBean" scope=
"application" />
```

当 scope 为 application 时，JavaBean 对象被创建后，它将存在于整个主机或虚拟主机的生命周期内，Application 范围是 JavaBean 最长的一个生命周期。同一个主机或虚拟主机中的所有文件共享这个 JavaBean 对象。如果服务器不重新启动，scope 为 application 的 JavaBean 对象会一直存放在内存中，随时处理客户的请求，直到服务器关闭，它在内存中占用的资源才会被释放。在此期间，服务器并不会创建新的 JavaBean 组件，而是创建源对象的一个同步备份，任何备份对象发生改变都会使源对象随之改变，不过这个改变不会影响其他已经存在的备份对象。

【练习 7】

下面仍以一个登录计数器为例，说明 Application 在 JSP 页面中的作用域。操作步骤如下。

（1）创建一个名称为 MyBean 的 JavaBean 组件，代码与练习 4 中步骤（1）的代码一致。

（2）创建登录页面 login4.jsp，在该页面中使用 JavaBean 组件，设置其作用域为 Application，代码如下。


```
<jsp:useBean id="myBean" class="com.itzcn.bean.MyBean" scope=
"application"></jsp:useBean>
```

（3）在 login4.jsp 页面中添加一个表单，用于登录，方法为 post，地址为 login4.jsp，代码与练习 4 中步骤（3）的代码类似，这里就省略了。

（4）在表单中加入用于显示登录次数的代码，代码与练习 6 中的步骤（4）代码一致。

（5）访问 login4.jsp 页面，多次登录、刷新当前页面或者将浏览器关闭重新访问 login4.jsp，运行结果如图 5-7 所示。

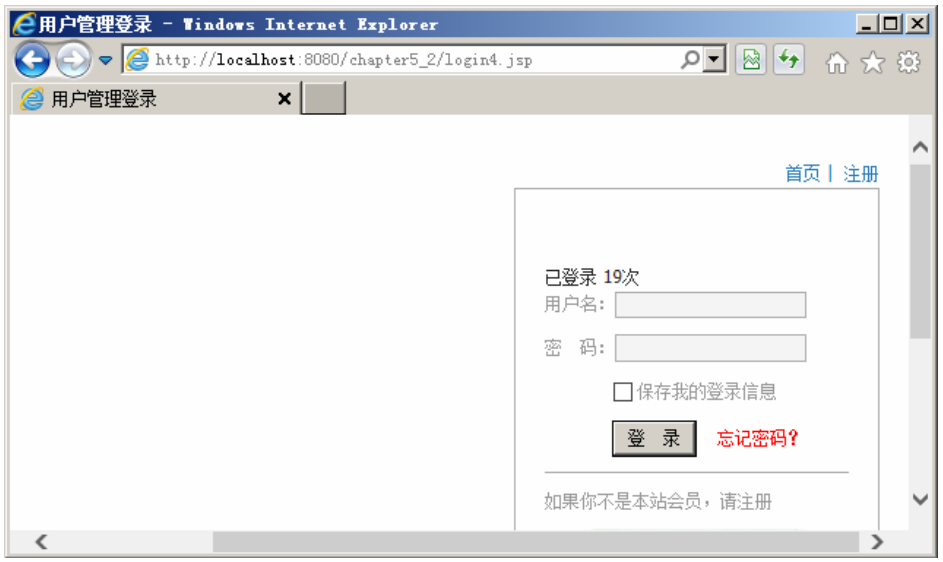

图 5-7 JavaBean 作用域为 Application 时的运行结果

每当单击【登录】按钮（或刷新当前页面）时，登录次数会增加数值 1。如果将浏览器窗口关闭，再次运行实例的 login4.jsp 页面，登录次数依然根据上一次的数值累加。将 JSP 容器（Tomcat）重新启动，再次运行 login4.jsp 文件，结果登录会从 1 开始累加。

5.5 实验指导 5–2：统计登录用户数量

通过把 JavaBean 对象放到 Application 作用域中，巧妙地实现统计登录用户位数。

（1）创建一个 MyBean 实体类，代码如下。

```
public class MyBean {
    private String name = null;     //用户名
    private String pass = null;     //密码
    private int count = -1;         //登录次数
//省略 getter、setter 方法
}
```

（2）新建一个 JSP 页面 index.jsp，引入 MyBean 对象，代码如下。

```
<jsp:useBean id="myBean" scope="application" class="com.itzcn.bean.
MyBean"/>
```

（3）在页面中，定义一个用户登录表单，代码如下。

```
<form action="index.jsp" method="post">
   <ul>
     <p class="tit">会员登录</p>
     <li style="padding-top:10px;">用户名:
       <input class="sr" type="text" name="userName" />
       <p> <input type="checkbox" class="remberme"  name="cookietime"
       value="315360000" />  记住我</p>
     <li>密    码:
       <input class="sr" type="password" name="userPass" id="userpass" />
       <br />
       <p style="top:80px">
    <input type="hidden" name="referer" value="./default.html" />
         <input type="button" id="submit" value="登录"/>
         <input type="hidden" name="loginsubmit" value="true"/>
       </p>
     </li>
  <li>
</li>
     <div class="member"></div>
   </ul>
  </form>
```

（4）获取用户名称，显示欢迎用户登录，第几位登录的用户信息，代码如下。

```
<font color="red"><%
request.setCharacterEncoding("utf-8");
String name = request.getParameter("userName");
if(request.getParameter("userName")!=null&&!request.getParameter
("userName").toString().trim().equals("")){
%>
<jsp:setProperty name="myBean" property="*"/>
欢迎
<%=name %>
登录，您是第
<jsp:getProperty name="myBean" property="count"/>
位登录的用户。
<%} %><br></font>
```

（5）运行 index.jsp，输入用户名与密码，单击【登录】按钮，执行效果如图 5-8 所示。

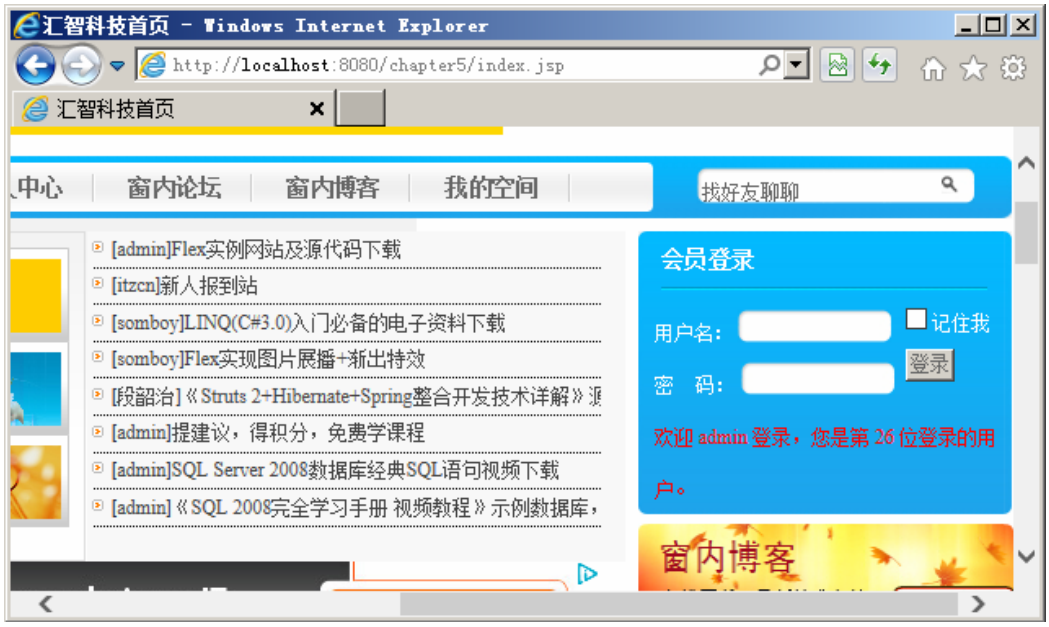

图 5-8 统计登录用户数量

思考与练习

一、填空题

1. _____的产生使 JSP 页面中的业务逻辑变得更加清晰，程序之中的实体对象及业务逻辑可以单独封装到 Java 类之中。

2. 使用 JavaBean 的最大优点就在于它可以提高代码的重用性。编写一个成功的 JavaBean，宗旨为“_____”。

3. 可视化 JavaBean 就是具有_____的 JavaBean。

4．JavaBean 中的属性应该设置为_____，可以防止外部直接访问。

5．一个 Bean 由三部分组成：实现 java.io.serializable 接口、提供无参数的构造方法、_____。

6．JavaBean 应该直接或间接实现_____接口，以支持序列化机制。

二、选择题

1．在 JavaBean 规范中类的属性需要使用_____修饰符来定义。

A．public

B．private

C．protected

D．friendly

2．下列选项中不属于 JavaBean 的属性的是_____。

A．Simple

B．Indexed

C．Bound

D．Complicated

3．设置 JavaBean 属性值使用的是_____标签。

A．<jsp:useBean>

B．<jsp:setProperty>

C．<jsp:getProperty>

D．上述三个标签都可以

4．下列关于 JavaBean 的 4 种作用域范围叙述中错误的是_____。

A．page 作用域不仅是在当前 JSP 页面内有效，在整个服务器中都有效

B．request 的作用域范围的 JavaBean 对象存储在当前 ServletRequest 中，有 request 范围的 JavaBean 实例可以在处理请求所有 JSP 页面中都存在

C．session 作用域范围的 JavaBean 将 JavaBean 对象存储在 HTTP 会话中

D．application 作用域范围的 JavaBean 对所有的用户和所有页面都起作用，只需创建一次，而且将会存在于 Web 应用程序执行的整个过程中

5．<jsp:useBean>标签的 scope 属性不可以设置为_____。

A．out

B．session

C．request

D．application

6．某 JSP 程序中声明使用 JavaBean 的语句如下：

```
<jsp:useBean id="user" class=
"mypackage.User" scope="page"/>
```

要取出该 JavaBean 的 loginName 属性值，以下语句正确的是_____。

A．<jsp:setProperty name=
"user"property="loginName"/>

B．<jsp:getProperty id="User" property=
"loginName"/>

C．<%=user.getLoginName()%>

D．<%=user.getProperty("loginName")%>

三、简答题

1．谈谈什么是 JavaBean。

2．简述创建一个 JavaBean 需要遵循的约束。

3．在为 JavaBean 添加 Simple 属性和 Indexed 属性时应该注意什么？

4．如何使用 JavaBean 的 Constrained 属性？

5．简述在 JSP 中使用一个 JavaBean 的过程。

6．简述 JavaBean 的 4 个作用域，并分别说明其作用范围。