第 2 章

JSP 动态页面开发技术

动态网页是指在服务器端运行的程序或者网页，它们会随不同客户、不同时间，返回不同的网页内容。动态网页需要用到服务器端脚本语言，JSP 就是目前流行的一种动态网页技术。动态网页的内容一般存储在数据库中，用户访问动态网页时，JSP 通过读取数据库中的存储数据来动态生成网页内容。本章对 JSP 相关技术进行介绍。

2.1 JSP 技术基础

JSP 是 Java Server Pages 的简称，它是由 Sun 公司倡导，多家公司共同参与建立起来的一种动态网页技术标准。它在动态网页中有着强大而特别的功能，具有跨平台性、易维护性、易管理性等特点。

2.1.1 JSP 简介

1．为什么需要 JSP

静态网页的显示内容是保持不变的，静态网页既不能实现与用户的交互，又不利于系统的扩展。所以，我们需要基于 B/S 技术的动态网页。

使用动态网页，可以动态地输出网页内容、同用户进行交互、对网页内容进行在线更新。这是由 B/S 技术的特点所决定的，如图 2-1 所示。

图 2-1 B/S 技术的特点

在 B/S 结构中，浏览器端与服务器端采用请求/响应模式进行交互，这个过程可以分解为如下步骤。

(1) 客户端浏览器接受用户的输入。一个用户在 IE 窗口中输入用户名、密码，单击“登录”按钮，发送对系统的访问请求。

(2) 客户端浏览器向应用服务器端发送请求。客户端把请求消息(包括用户名、密码等信息)发送到应用服务器端，等待服务器端的响应。

(3) 数据处理。应用服务器端通常使用服务器端脚本语言，如 JSP 等，来访问数据库服务器，查询该用户有无访问权限，并获得查询结果。

(4) 发送响应。应用服务器端向客户端浏览器发送响应消息(一般是动态生成的 HTML 页面)，并由访问者的浏览器端解释 HTML 文件，呈现用户界面。

实现动态网页的关键在于运行在应用服务器端的服务器端脚本语言，它可以根据不同用户的请求输出相应的 HTML 页面，然后应用服务器再把这个 HTML 页面返回给客户端。

2．JSP 的执行过程

实际上，JSP 就是指在 HTML 中嵌入 Java 脚本语言，当用户通过浏览器请求访问 Web 应

用时，Web 服务器会使用 JSP 引擎对请求的 JSP 进行编译和执行，然后将生成的页面返回给客户端浏览器进行显示。当 JSP 请求提交到服务器时，Web 服务器会通过三个阶段实现处理，执行过程如下。

(1) 翻译阶段。当 Web 服务器接收到 JSP 请求时，首先会对 JSP 文件进行翻译，将编写好的 JSP 文件通过 JSP 引擎转换成可识别的 Java 源代码。

(2) 编译阶段。经过翻译的 JSP 文件相当于编写好的 Java 源文件，必须将 Java 源文件编译成可执行的字节码文件。

(3) 执行阶段。Web 容器接收客户端的请求后，经过翻译和编译两个阶段，生成了可以被执行的字节码文件，此时就进入执行阶段。当执行完成后，会得到请求的处理结果，Web 容器再把生成的结果页面返回到客户端显示。

Web 容器处理 JSP 文件请求的三个阶段，如图 2-2 所示。

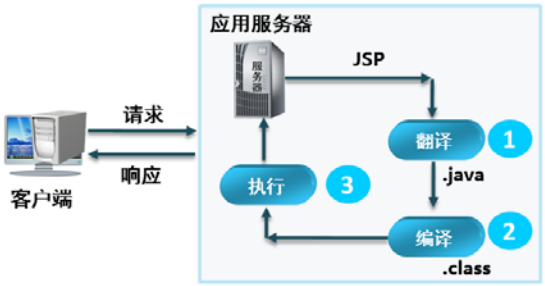

图 2-2 JSP 的执行过程

Web 容器将 JSP 文件翻译和编译完成后，会将编译好的字节码文件放在内存中，当客户端再一次请求时，就可以重用这个编译好的字节码文件，而无须重新翻译和编译，这样就大大提高了 Web 应用系统的性能。如果对 JSP 文件进行了修改，Web 容器会及时发现，此时 Web 容器就会重新执行翻译和编译过程。因此，JSP 在第一次请求时会比较慢，后续访问速度就会很快。当然，如果 JSP 文件发生了变化，同样需要重新进行编译。

2.1.2 JSP 页面组成

了解 JSP 的工作原理和执行过程，是学习 JSP 的基础，而使用 JSP 进行动态网页开发，还需要掌握 JSP 页面中包括哪些元素，不同元素具备什么功能。前面谈到，JSP 是通过在 HTML 中嵌入 Java 脚本语言来响应页面动态请求的。下面通过在页面上显示相应日期的示例，展示几个比较常用的 JSP 页面元素。在第 1 章的 restaurant 项目的 WebRoot 下新建 ch02 文件夹，并在该文件夹中新建 showDate.jsp 文件，代码如下：

```
<%@ page language="java" import="java.util.*"
contentType="text/html;charset=utf-8" %>
    <%@ page import="java.text.*" %>
    <html>
        <head>
           <title>输出当前系统日期</title>
        </head>
          <!-- 这是 HTML 注释(客户端可以看到源代码) -->
          <%-- 这是 JSP 注释(客户端不能看到源代码) --%>
        <body>
           你好，今天日期是：
          <%  //使用预定格式将日期转换为字符串
              SimpleDateFormat f = new SimpleDateFormat("yyyy 年 MM 月 dd 日");
```

```
            String currentTime = f.format(new Date());
        %>
        <%=currentTime %>
        <%! String declare = "这是声明";%>
        <%= declare %>
      </body>
</html>
```

部署项目，通过浏览器访问 http://localhost:8080/restaurant/ch02/showDate.jsp，该示例在浏览器上的运行结果如图 2-3 所示。该示例产生的网页源代码如图 2-4 所示。

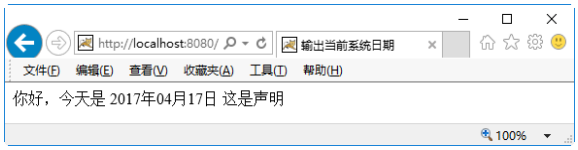

图 2-3　在浏览器上查看日期显示

```
1
2
3  <html>
4      <head>
5          <title>输出当前系统日期</title>
6      </head>
7        <!-- 这是HTML注释(客户端可以看到源代码) -->
8
9        <body>
10       你好，今天是
11
12       2017年04月17日
13
14       这是声明
15     </body>
16 </html>
```

图 2-4　查看示例网页源代码

在该示例中，一共展示了 5 种页面元素，包含静态内容、指令、小脚本、表达式、声明和注释，下面来一一介绍。

1．静态内容

静态内容是指 JSP 页面中的静态文本，它基本上是 HTML 文本，与 Java 和 JSP 无关。

2．JSP 注释

在编程中，添加注释是非常好的习惯，合理详细的注释有利于后期代码的维护以及团队成员的阅读。在 JSP 文件中有三种注释方法。

(1) HTML 注释方法，其格式为<!-- HTML 注释 -->，其中的注释内容在客户端浏览器里是可以看见的。这种注释方法不安全，而且会增加网络负担。

(2) JSP 注释标记，其格式为<%-- JSP 注释 --%>，在客户端查看源代码时看不到注释中的内容，安全性较高。

(3) 在 JSP 脚本中使用注释，和在 Java 类中进行注释的方式是一样的，使用的格式为<% //单行注释%>、<% /* 多行注释 */ %>。

3．JSP 脚本元素

在 JSP 中，将表达式、小脚本、声明统称为 JSP 脚本元素，用于在 JSP 页面中嵌入 Java 代码，实现页面的动态请求。

1) 小脚本

小脚本可以包含任意 Java 片段，形式比较灵活，通过在 JSP 页面中编写小脚本可以执行

复杂的操作和业务处理，编写方法就是将 Java 代码片段插入“<%　%>”标记中，在前面的示例中，属于小脚本的代码如下：

```
<%
      //使用预定格式将日期转换为字符串
   SimpleDateFormat f = new SimpleDateFormat("yyyy年MM月dd日");
   String currentTime = f.format(new Date());
%>
```

下面通过小脚本在页面上按行循环输出数组中的值，创建 showArray.jsp 页面，页面部分的小脚本代码如下：

```
<%
   char[] array={'A','B','C','D'};
   for(int i=0;i<array.length;i++){
      out.println(array[i]);
%>
   <br>
<%
   }
%>
```

由于小脚本和 HTML 静态页面混合编写，很容易忘记转行的标签，又不容易被发现。因此大家在编写 JSP 代码时不仅要注意代码的缩进，而且要编写必要的注释。

2) 表达式

表达式是对数据的表示，系统将其作为一个值进行计算和显示，当需要在页面中获取一个 Java 变量或者表达式值时，使用表达式是非常方便的。其语法是：<%=Java 表达式 %>。需要注意的是，使用表达式输出数据时，不能在表达式结尾添加分号来代表语句结束。

在 Java 开发过程中，使用 System.out.println()和 System.out.print()向控制台输出信息。在 JSP 网页上，可以使用内置对象 out 把结果输出到页面上，out 对象是 JSP 开发过程中使用最为频繁的对象，使用也是最简单的。

3) JSP 声明

在编写 JSP 页面程序时，有时需要为 Java 脚本定义变量和方法，这时就需要对所使用的变量和方法进行声明。声明一般没有输出，通常与表达式、小脚本一起综合运用。将输出日期的示例进行修改，在同一 JSP 页面中，如果需要在多个地方格式化日期，在 Java 代码中可以增加一个方法来解决，在 JSP 文件中，同样可以声明方法来解决类似问题。在 ch02 的文件夹中新建 show.jsp 文件，具体示例代码如下：

```
<body>
   <%!
      String formatDate(Date date){
         SimpleDateFormat f = new SimpleDateFormat("yyyy年MM月dd日");
         return f.format(date);
      }
   %>
   第一次显示时间：今天是<%=formatDate(new Date()) %> <br>
   第二次显示时间：今天是<% out.print(formatDate(new Date())); %>
</body>
```

部署项目，访问 http://localhost:8080/restaurant/ch02/show.jsp，运行效果如图 2-5 所示。

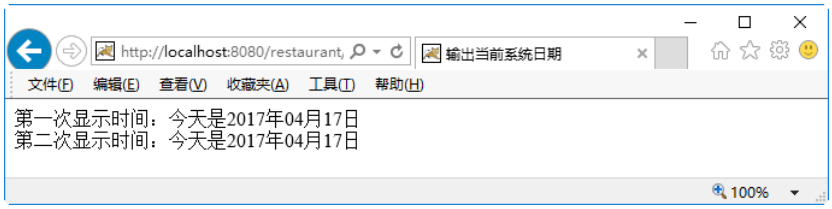

图 2-5　显示时间的运行结果

4．JSP 指令元素

JSP 指令用来设置与整个 JSP 页面相关的属性，在 JSP 运行时，控制 JSP 页面的某些特性，如网页的编码方式和脚本语言。JSP 指令一般以“<%@”开始，以“%>”结束。

JSP 指令有多种类型，主要有三种：page、include 和 taglib，下面分别进行介绍。

1) page 指令

在 Java 文件中，可以通过两种方式引入其他包中的类。第一种是使用 import 关键字，优点在于：一次引入，处处使用；另外一种方式就是使用完全限定的类名，即类名前必须加上完整的包名。在 JSP 文件中，同样可以采用以上两种方式。通常情况下，我们使用 import 关键字引入 Java 类文件，好处在于：一旦引入，这个 Java 类文件在整个 JSP 文件范围内都可用。

page 指令就是通过设置内部的多个属性来定义 JSP 文件中的全局特性。page 指令只能对当前自身页面进行设置，即每个页面都有自身的 page 指令，如果没有对某些属性进行设置，JSP 容器将使用默认指令属性值。

page 指令一般放在 JSP 页面的第一行，其语法格式如下：

```
<%@ page 属性名 1="属性值 1" 属性名 2="属性值 2,属性值 3" …… %>
```

示例：

```
<%@ page language="java" import="java.util.*,java.text.*" contentType=
"text/html; charset=utf-8" %>
```

在对同一个属性设置多个属性值时，其间以逗号相互隔开。

page 指令的属性共有 13 个，其中最常用的几个属性的含义如表 2-1 所示。

表 2-1　page 指令常用属性

属　性	描　述
language	指定 JSP 页面使用的脚本语言，默认为 Java
import	通过该属性引用脚本语言中使用的类文件
contentType	指定 JSP 页面采用的编码方式，默认为 text/html,charset=ISO-8859-1
pageEncoding	指定页面使用的字符编码，默认为 ISO-8859-1

(1) language 属性。

language 属性用来指定当前 JSP 页面所采用的脚本语言，当前 JSP 版本只能使用 Java 语言。该属性可以不设置，因为 JSP 默认就是采用 Java 作为脚本。language 属性的设置方法如下：

```
<%@ page language="java" %>
```

(2) import 属性。

import 属性在实际开发中使用非常频繁。通过 import 属性可以在 JSP 文件的脚本片段中引用外在的类文件。如果一个 import 属性引入多个类文件，则多个类文件之间要用逗号隔开。import 属性的设置方法如下：

```
<%@ page import="java.util.*,java.text.*" %>
```

(3) contentType 属性。

contentType 属性的设置在开发过程中是非常重要的，且经常被用到，中文乱码一直是困扰开发者的一个问题，而该属性就是用来设置编码格式的，告诉 Web 容器在客户端浏览器上以何种格式显示 JSP 文件以及使用何种编码格式。contentType 属性的设置方法如下：

```
<%@ page contentType="text/html;charset=utf-8" %>。
```

text/html 和 charset=utf-8 之间用分号隔开，它们同属于 contextType 属性值。当设置为 text/html 时，表示该页面以 HTML 页面的格式进行显示。这里设置的编码格式为 utf-8，这样 JSP 页面中的中文就可以正常显示了。

(4) pageEncoding 属性。

该属性用来指定页面所使用的字符编码，如果设置了这个属性，则 JSP 页面的字符编码就是指定的字符集；如果未设置此属性，则使用 contextType 属性的值；如果两个属性均未设置，页面默认使用 ISO-8859-1。

在什么位置插入 page 指令并不重要，因为 page 指令和其他指令一样，只在 JSP 页面编译的时候起作用。page 指令的参数除了以上介绍的 4 个外，还有 extends、session、buffer、autoFlush、isThreadSafe、info、errorPage、isErrorPage、isELIgnored，这些参数的名称是区分大小写的。

2) include 指令

include 指令用于在 JSP 页面中包含一个文件，该文件可以是 JSP 文件、HTML 网页、文本文件或 Java 代码，使用 include 指令可以简化页面代码，提高代码的重用性，语法如下：

```
<%@ include file="被包含文件的 url 地址" %>
```

例如，在页面中包含 index.jsp 的代码为<%@ include file="../index.jsp"%>。

被包含的文件中，最好不要使用<html>、</html>、<body>、</body>等标签，否则会影响 JSP 网页中的相同标签，从而导致不必要的错误。

例如，许多网站的每个页面都有一个小小的导航条，导航条往往用页面顶端或左边的一

个表格制作，同一份 HTML 代码重复出现在整个网站的每个页面上。include 指令是实现该功能非常理想的方法。使用 include 指令，开发者不必再把导航的 HTML 代码复制到每个文件中，从而可以更轻松地完成维护工作。

3) taglib 指令

taglib 指令用于通知 JSP 容器某个页面依赖于自定义标签库，标签库是可用于扩展 JSP 功能的自定义标签的集合，taglib 指令的语法如下：

```
<%@ taglib uri="标签名称空间" prefix="前缀" %>
```

uri 属性指定了 JSP 要在 web.xml 文件中查找的标签库描述符，该描述符是一个标签描述文件(*.tld)的映射。另外，通过 uri 属性直接指定标签描述文件的路径，而无须在 web.xml 文件中进行配置，同样可以使用指定的标记。prefix 属性指定了一个在页面中使用由 uri 属性指定的标签库的前缀，示例如下：

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

2.2 JSP 内置对象

在当今的 Web 程序页面中，用户交互是必不可少的，JSP 的内置对象就是用于处理浏览器请求的对象，可以直接使用，而不需要自己创建。

2.2.1 什么是 JSP 内置对象

JSP 内置对象，就是当你编写 JSP 页面时，无须做任何声明就可以直接使用的对象。例如，在 2.1.2 小节的示例中出现了如下的代码片段：

```
<%
   char[] array={'A','B','C','D'};
   for(int i=0;i<array.length;i++){
      out.println(array[i]);
   }
%>
```

代码 out.println()可以实现页面的输出显示，但是代码中并没有任何地方声明或者创建这个 out 对象，没有创建就可以直接使用的原因，就是 out 对象是 JSP 的内置对象之一。除了 out 对象之外，在 JSP 中还有其他一些内置对象，如图 2-6 所示。

图 2-6 JSP 常用的内置对象

所谓内置对象就是由 Web 容器加载的一组类的实例，它不像一般的 Java 对象在创建类的实例时，必须要用 new 关键字去构造对象，而是可以直接在 JSP 中使用的对象。

JSP 的内置对象名称均是 JSP 的保留字，不得随便使用。

2.2.2 out 内置对象

out 内置对象是 JSP 在开发过程中使用最为频繁的对象，是 JspWriter 类的实例，用来向客户端输出内容。out 对象常用的方法是 print()和 println()，这个方法用于在页面中打印字符串信息。out 对象除了输出方法外，还有一些其他的方法，这些方法主要用来管理缓冲流或者输出流，如表 2-2 所示。

表 2-2 out 对象的其他方法

方 法	说 明
void clear()	清除缓冲区的内容
void clearBuffer()	清除缓冲区的当前内容
void flush()	清除数据流
void close()	关闭输出流
int getBufferSize()	返回缓冲区字节数大小，如不设缓冲区则为 0
int getRemaining()	返回缓冲区还剩余多少可用
boolean isAutoFlush()	返回缓冲区满时，是自动清空还是抛出异常

2.2.3 request 内置对象

request 内置对象是最常用的对象之一，主要用于处理客户端浏览器的请求，其工作原理如图 2-7 所示。

request 对象中包含请求的相关信息，可以在 JSP 页面中通过调用 request 对象的方法获取请求的相关数据。request 对象的常用方法如表 2-3 所示。

图 2-7 request 内置对象的工作原理

表 2-3 request 对象的常用方法

方 法	说 明
String getParameter(String name)	根据表单组件名称获取提交数据
String[ ] getParameterValues(String name)	获取表单组件对应多个值时的请求数据
void setCharacterEncoding(String charset)	指定每个请求的编码

续表

方　法	说　明
request.getRequestDispatcher(String path)	返回一个 java.servlet.RequestDispatcher 对象，该对象的 forward( )方法用于转发请求

在 restaurant 项目的 WebRoot/ch02/目录下，创建 reg.jsp 注册页面和 reginfo.jsp 注册提交页面，编程实现用户的注册功能。

注册页面 reg.jsp 的主要代码如下：

```
<body>
  请输入注册信息<br>
  <form name="form1" method="post" action="ch02/reginfo.jsp">
       用户名：<input type="text" name="username"><br>
       密    码：
              <input type="password" name="password"><br>
       业余爱好：
         <input type="checkbox" name="habit" value="看书">看书
         <input type="checkbox" name="habit" value="玩游戏">玩游戏
         <input type="checkbox" name="habit" value="旅游">旅游
         <input type="checkbox" name="habit" value="看电视">看电视<br>
         <input type="submit" value="提交">  
         <input type="reset" value="取消">
  </form>
</body>
```

注册提交页面 reginfo.jsp 的主要代码如下：

```
  <%
       request.setCharacterEncoding("utf-8"); //设置读取字符编码为 utf-8
       String username=request.getParameter("username");    //读取用户名
       String password=request.getParameter("password");    //读取密码
       String[] habits=request.getParameterValues("habit");//读取兴趣爱好
  %>
<body>
     您输入的注册信息如下：<br>
     用户名为：<%=username %>   <br>
  <%
      out.print("密码为："+password+"<br>");
       out.print("您的兴趣爱好：");
       if(habits!=null){
          for(int i=0;i<habits.length;i++){
             out.print(habits[i]+"  ");
          }
       }
  %>
</body>
```

注册页面信息包括用户名、密码、兴趣爱好，如图 2-8 所示；页面提交后，显示用户输入的数据，如图 2-9 所示。

图 2-8 输入注册信息

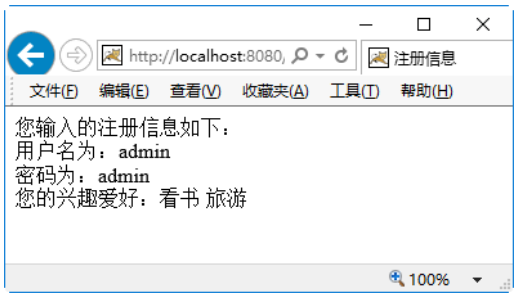

图 2-9 显示注册信息

2.2.4 response 内置对象

response 对象用于响应客户请求并向客户端输出信息，与内置 request 对象相对应，其工作原理如图 2-10 所示。

图 2-10 response 内置对象的工作原理

response 对象也提供了多个方法用来处理 HTTP 响应，response 对象的常用方法如表 2-4 所示。

表 2-4 response 对象的几个常用方法

方 法	说 明
void addCookie()	在客户端添加 Cookie
void setContentType(String type)	设置 HTTP 响应的 contentType 类型
void setCharacterEncoding(String charset)	设置响应所采用的字符编码类型
void sendRedirect(String path)	将请求重新定位到一个不同的 URL 上

在 restaurant 项目的 WebRoot/ch02/目录下，创建登录页面 login.jsp、登录处理页面 control.jsp 和欢迎页面 welcome.jsp。编程实现用户的登录处理，并跳转到欢迎页面。

登录页面 login.jsp 的表单部分代码如下：

```
<form name="form1" method="post" action="ch02/control.jsp">
    用户名：<input type="text" name="username">
    密码：<input type="password" name="password">
    <input type="submit" value="登录">
</form>
```

登录处理页面 control.jsp 接收参数处理请求部分的代码如下：

```
<%
    request.setCharacterEncoding("utf-8");
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    if(username.equals("admin") && password.equals("admin")){
        //此处暂不访问数据库，用户名和密码都为 admin
        response.sendRedirect("welcome.jsp");
    }
%>
```

欢迎页面 welcome.jsp 的主要代码如下：

```
<body>
     欢迎 来到本页面!
</body>
```

在登录页面输入用户名和密码，如图 2-11 所示。单击“登录”按钮，提交至处理页面，并接收用户名和密码参数，进行逻辑判断。符合条件则跳转到欢迎页面，客户端重新建立链接，URL 地址发生了变化，如图 2-12 所示。

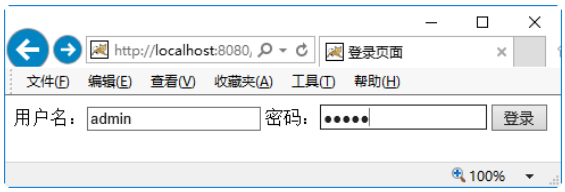

图 2-11　登录页面

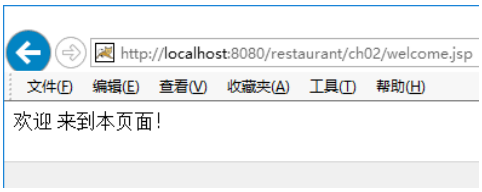

图 2-12　欢迎页面

如果要在欢迎页面显示登录的用户名，可以使用 request 对象获取用户请求的数据。在 welcome.jsp 页面中显示该用户名，修改 welcome.jsp 的代码，具体如下：

```
<%
    String username=request.getParameter("username");
%>
欢迎<%=username %>，来到本页面!
```

重新部署项目，运行程序，登录后进入欢迎页面，并未显示用户名，如图 2-13 所示。

图 2-13　显示用户信息

重定向是在客户端发挥作用的，客户端重新向服务器请求一个地址链接，由于是新发送的请求，因而上次请求的数据将随之丢失，在地址栏中可以显示转向后的地址。由于服务器重新定向了新的 URL 地址，所以重定向可以理解为是浏览器至少提交了两次请求。

修改登录处理页面 control.jsp，将跳转部分的代码进行修改，代码如下：

```
if(username.equals("admin") && password.equals("admin")){
```

```
    request.getRequestDispatcher("welcome.jsp").forward(request, response);
}
```

重新运行程序，再次显示运行结果，如图 2-14 所示。

图 2-14　显示用户的欢迎页面

转发是在服务器端发挥作用的，通过 forward()将提交信息在多个页面间进行传递，整个过程都是在一个 Web 容器内完成的，因而可以共享 request 范围内的数据。而对应到客户端，不管服务器内部如何处理，作为浏览器都只是提交了一个请求，客户端浏览器的 URL 地址栏不会显示转向后的地址。

2.2.5　session 内置对象

我们在上网时，一定有过这样的经历：好不容易找到一个下载地址，可是单击下载时，系统会自动转入登录页面，提示登录。如果是已登录用户，就不会面临这样的问题，系统如何判断用户是否已经登录过该网站呢？JSP 中提供了会话跟踪机制，该机制可以保持每个用户的会话信息，为不同的用户保存自己的数据。

对 Web 开发来说，一个会话就是用户通过浏览器和服务器之间进行的一次通话，它可以包含浏览器与服务器之间的多次请求和响应。当用户向服务器发出第一次请求时，服务器会为该用户创建唯一的会话，会话将一直持续到用户访问结束(即浏览器的关闭)，JSP 提供了一个可以在多个请求之间持续有效的会话对象 session，如图 2-15 所示。

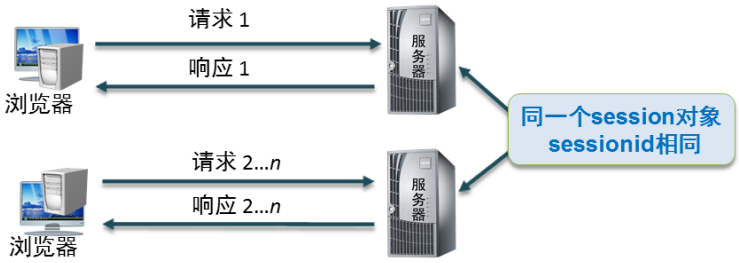

图 2-15　会话过程

session 机制是一种服务器端的机制，在服务器端使用，类似于散列表的结构来保存信息，当程序接收到客户端的请求时，服务器首先检查这个客户端是否已经创建了 session。

在 JSP 中，session 对象用来存储相关用户会话的所有信息。一个用户对应一个 session，并且随着用户的离开，session 中的信息也随之消失。session 对象的常用方法如表 2-5 所示。

表 2-5　session 对象的常用方法

方　法	说　明
void setAttribute(String key,Object value)	以 key/value 的形式保存对象值
Object getAttribute(String key)	通过 key 获取对象值

续表

方　法	说　明
void invalidate()	设置 session 对象失效
String getId()	获取 sessionid
void setMaxInactiveInterval(int interval)	设定 session 的非活动时间
int getMaxInactiveInterval()	获取 session 的有效非活动时间(秒为单位)
void removeAttribute(String key)	从 session 中删除指定名称(key)所对应的对象

在 restaurant 项目的 WebRoot/ch02/目录下，创建普通的首页面 index.jsp，访问控制流程如图 2-16 所示。

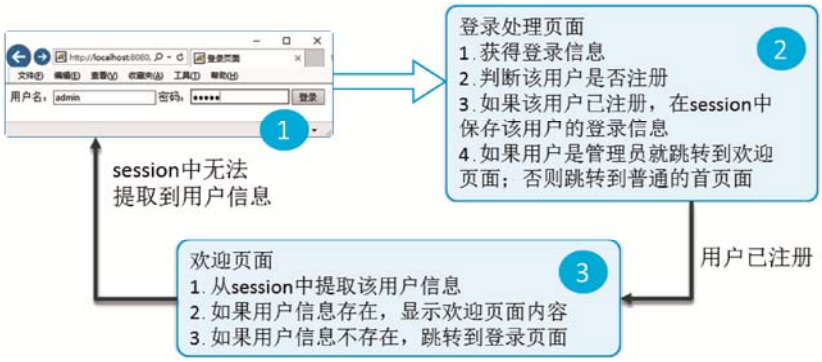

图 2-16　session 对象的访问控制流程

根据访问控制流程，修改 2.2.4 小节中的登录处理页面 control.jsp，在会话中，保存用户信息，如果用户登录成功则跳转到欢迎页面，代码如下：

```
<%
   request.setCharacterEncoding("utf-8");
   String username = request.getParameter("username");
   String password = request.getParameter("password");
   if(username.equals("admin") && password.equals("admin")){
      //此处暂不访问数据库，用户名和密码默认为 admin
      session.setAttribute("LOGINED_NAME", username);//设置登录信息
      session.setMaxInactiveInterval(10*60);//设置 session 的过期时间
      response.sendRedirect("welcome.jsp");
   }else{
      response.sendRedirect("index.jsp");
   }
%>
```

修改 2.2.4 小节中的欢迎页面 welcome.jsp，在欢迎页面中读取会话中的用户信息，并进行校验，校验失败，则返回登录页面，代码如下：

```
<%
     String loginedName=(String)session.getAttribute("LOGINED_NAME");
     if(loginedName==null){
        response.sendRedirect("login.jsp");
     }
```

```
    out.print("欢迎"+loginedName+"，来到本欢迎页面！");
%>
```

重新部署项目，运行登录页面，输入用户名和密码，则跳转到欢迎页面，并显示用户名，地址栏中的地址显示的是 welcome.jsp，如图 2-17 所示。

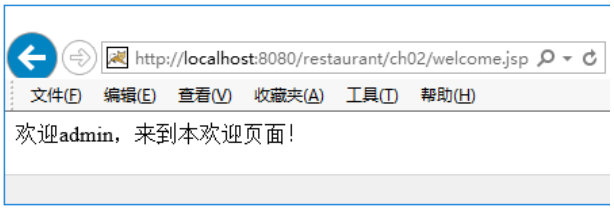

图 2-17　登录后的欢迎页面

只要当前浏览器不关闭，在设置的 session 期限内，session 都是有效的，如果已达到期限或者打开一个新的浏览器，则 session 中存储的对象会被释放。

2.2.6　application 内置对象

application 对象类似于系统的“全局变量”，可跨越多个浏览器，用于实现用户之间的数据共享。application 对象的常用方法如表 2-6 所示。

表 2-6　application 对象的常用方法

方　法	说　明
void setAttribute(String key,Object value)	以 key/value 的形式保存对象值
Object getAttribute(String key)	通过 key 获取对象值
String getRealPath(String path)	返回相对路径的真实路径

在 restaurant 项目的 WebRoot/ch02/目录下，新建统计显示页面 showCount.jsp，用来实现在网站系统中统计并显示已访问过的人数，具体实现代码如下：

```
<body>
  <%
     Integer count=(Integer)application.getAttribute("count");
     if(count==null){
        count=1;
     }else{
        count=count+1;
     }
     application.setAttribute("count", count);
  %>
  统计访问量：目前有<%=application.getAttribute("count") %>个人访问过本网站！
</body>
```

重新部署项目，运行统计显示页面，后显示有多少个人访问过网站，刷新页面后，或重新打开浏览器访问页面，访问过的人数会增加，运行效果如图 2-18 所示。

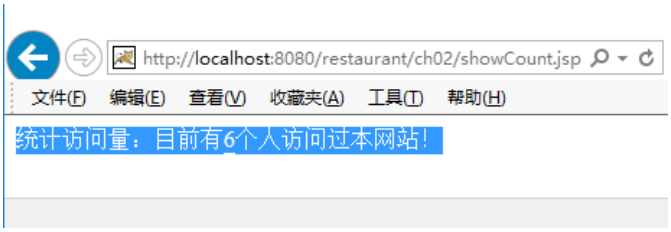

图 2-18　访问人数统计页面

2.2.7　其他内置对象

在 JSP 中，除了 out、request、response、session、application 对象外，还有 page、config、exception、pageContext 四个对象。

1．page 对象

page 对象表示当前页面，page 对象就是页面实例的引用，指向当前 JSP 页面本身，类似于 Java 中的 this 关键字。在 JSP 页面中，page 对象使用得较少。

2．config 对象

config 对象用于存放 JSP 页面编译后的初始数据。config 对象代表当前 JSP 配置信息，但 JSP 页面通常无须配置，因此也就不存在配置信息。与 page 对象一样，config 对象在 JSP 页面中也很少使用。

3．exception 对象

exception 对象表示 JSP 页面运行时产生的异常，该对象只有在错误页面(page 指令中设定 isErrorPage 为 true 的页面)中才能使用。

4．pageContext 对象

pageContext 对象代表页面上下文，提供了对 JSP 页面内所有对象及命名空间的访问，并提供访问其他隐含对象的方法，即 request 对象、response 对象、application 对象，config 对象、session 对象、out 对象可以通过访问这个对象的属性来导出，相当于页面中所有功能的集成。pageContext 对象的常用方法如表 2-7 所示。

表 2-7　pageContext 对象的常用方法

方　法	说　明
ServletRequest getRequest()	获得 request 对象
ServletResponse getResponse()	获得 response 对象
HttpSession getSession()	获得 session 对象
JspWriter getOut()	获得 out 对象
void setAttribute(String name,Object attribute)	设置 name 属性及属性值

续表

方　法	说　明
void getAttribute(String name)	获取 page 范围内属性的值
void getAttribute(String name,int scope)	获取指定范围内的 name 属性
void include(String relativeUrlPath)	在当前位置包含另一文件
Object getPage()	返回当前页面的 Object 对象

2.3　对象的范围

在 JSP 页面中的对象，无论是用户创建的对象，还是 JSP 的内置对象，都有一个范围，这个范围定义了在什么时间内，在哪些 JSP 页面中可以访问这些对象。在 JSP 中，对象有四种范围：page、request、session 和 application，它们都能借助 setAttribute()方法和 getAttribute()方法来设置和取得其属性，也可通过 removeAttribute()来删除属性。

2.3.1　page 范围

所谓的 page 范围，是指单一 JSP 页面的范围。page 范围内的对象只能在创建对象的页面中访问。在 page 范围内，将数据存入和取出，可以使用 pageContext 对象的 setAttribute()和 getAttribute()方法。page 范围内的对象在客户端每次请求 JSP 页面时创建，在服务器发送响应或请求转发到其他页面或资源后失效。

在 restaurant 项目的 WebRoot/ch02/目录下，创建 rangeOne.jsp 页面和 rangeTwo.jsp 页面。在第一个页面中，可以调用 pageContext 的 setAttribute()方法将一个字符串类型对象保存为 page 范围，然后分别在本页面和另一页面调用 pageContext 的 getAttribute()方法访问具有 page 范围的这个对象。

rangeOne.jsp 页面的代码如下：

```
<%
      String name="pageRange";
      pageContext.setAttribute("name", name);
%>
<h3>rangeOne：<%=pageContext.getAttribute("name") %></h3>
<% pageContext.include("rangeTwo.jsp"); %>
```

rangeTwo.jsp 页面的代码如下：

```
<h3>rangeTwo：
<%=pageContext.getAttribute("name") %>
</h3>
```

部署项目，运行访问 rangeOne.jsp 页面，效果如图 2-19 所示。

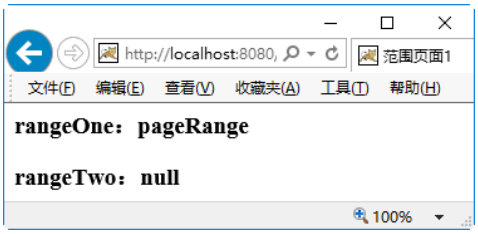

图 2-19　page 范围演示

pageContext 对象本身也属于 page 范围，具有 page 范围的对象被绑定到 pageContext 对象中。

2.3.2 request 范围

相对于在 page 范围内的对象与 pageContext 绑定在一起，request 范围内的对象则与客户端用户的请求绑定在一起，即 request 范围内的对象在页面转发或包含中有效。在该范围内的对象同样可以通过调用 request 对象的 setAttribute()与 getAttribute()方法找到，同时在调用 forward()方法转向的页面或者调用 include()方法包含的页面时，都可以访问 request 范围内的对象。

修改 rangeOne.jsp 页面，代码如下：

```
<%
      String name="requestRange";
      request.setAttribute("name", name);
%>
<h3>rangeOne：<%=request.getAttribute("name") %></h3>
<% pageContext.include("rangeTwo.jsp"); %>
```

修改 rangeTwo.jsp 页面，代码如下：

```
<h3>rangeTwo：<%=request.getAttribute("name") %></h3>
```

部署项目，运行访问 rangeOne.jsp 页面，效果如图 2-20 所示。

因为请求对象对于客户端的每次用户请求都是不同的，所以对于任何一个新的请求，都要重新创建该范围内的对象。而当请求结束后，创建的对象也就随之消失。

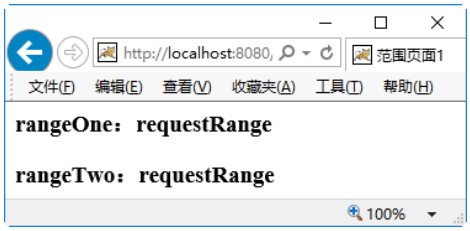

图 2-20 request 范围演示

2.3.3 session 范围

JSP 容器为每一次会话创建一个 session 对象，在会话期间，只要将对象绑定到 session 中，对象的范围就为 session。在会话有效期间，都可以访问 session 范围内的对象。

修改 rangeOne.jsp 页面，代码如下：

```
<%
      String req="requestRange";
      String ses="sessionRange";
      request.setAttribute("req", req);
      session.setAttribute("ses", ses);
      response.sendRedirect("rangeTwo.jsp");
  %>
```

修改 rangeTwo.jsp 页面，代码如下：

```
<h3>request：<%=request.getAttribute("req") %></h3>
<h3>session：<%=session.getAttribute("ses") %></h3>
```

运行访问 rangeOne.jsp 页面，效果如图 2-21 所示。

使用 response 对象将页面重定向到 rangeTwo.jsp，在 rangeTwo.jsp 中能够读取 session 对象，由此可见，session 范围内的对象在会话有效期内可以访问，使用 response.sendRedirect()方法重定向到另外一个页面时，相当于重新发起一次请求，而上一次请求中的 request 对象则随之消失。

图 2-21　session 范围演示

2.3.4　application 范围

相对于 session 范围针对一个会话，application 范围则面对整个 Web 应用程序，即当服务器启动后就会创建一个 application 对象，被所有用户所共享。当具有 application 范围的对象被设置值后，在 Web 应用程序的运行期间，所有的页面都可以访问 application 范围内的对象，其范围最大。与前面几个对象类似，application 对象也具有 setAttribute()方法和 getAttribute()方法，用于对该范围内的对象进行存储访问。

修改 rangeOne.jsp 页面，代码如下：

```
<%
      String ses="sessionRange";
      String app="applicationRange";
      session.setAttribute("ses", ses);
      application.setAttribute("app",app);
      response.sendRedirect("rangeTwo.jsp");
%>
```

修改 rangeTwo.jsp 页面，代码如下：

```
<h3>session：<%=session.getAttribute("ses") %></h3>
<h3>application：<%=application.getAttribute("app") %></h3>
```

先运行 rangeOne.jsp 页面，再运行 rangeTwo.jsp 页面，效果如图 2-22 所示。

这时，关闭浏览器再次运行 rangeTwo.jsp，效果如图 2-23 所示。

图 2-22　application 范围演示(1)

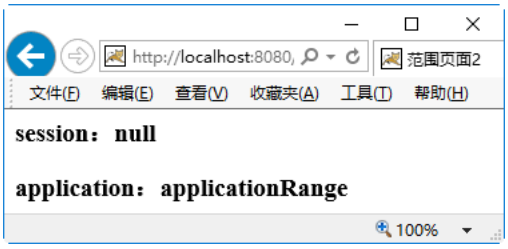

图 2-23　application 范围演示(2)

由于 session 范围针对一个会话，当浏览器关闭后会话也随之结束，所以无法读取，而 application 范围针对整个系统的服务，因而数据可以被再次读取。

2.4 在 JSP 中使用 JavaBean

JavaBean 既是一种基于 Java 平台的软件组件，也是一种独立于平台和结构的应用程序编程接口(API)。JSP 搭配 JavaBean 的组合已经成为常见的 JSP 程序标准，广泛应用于各类 JSP 应用程序中。

2.4.1 为什么需要 JavaBean

Java 企业应用是基于组件开发的，就好像我们用积木可以搭建不同的造型。在程序中，Java 是一种面向对象的编程语言，在设计和解决问题时，都是以面向对象的思想进行的。比如，数据库连接类，在这个类中定义了连接方法和关闭方法，对这个类来说，它的使命就是建立连接和关闭连接，是程序的一个组成部分。在 JSP 中调用 JavaBean，有如下两个优点。

(1) 提高代码的可复用性。

对于通常使用的业务逻辑代码，如数据运算和处理、数据库操作等，可以封装到 JavaBean 中。在 JSP 文件中可以多次调用 JavaBean 中的方法来实现快速的程序开发。

(2) 将 HTML 代码和 Java 代码分离，使程序有利于开发维护。

将业务逻辑进行封装，使得业务逻辑代码和显示代码相分离，不会互相干扰，避免了代码又多又复杂的问题，方便了日后的维护。

2.4.2 什么是 JavaBean

JavaBean 是 Java 中开发的可以跨平台的重要组件。JavaBean 在服务器端的应用中表现出强大的生命力，在 JSP 程序中常用来封装业务逻辑、数据库操作等。JavaBean 的本质就是一个 Java 类，只不过这个 Java 类要遵循一些编码的约定。

JavaBean 实际上就是 Java 类，这个类可以重用，从功能上可以分为以下两类：封装数据和封装业务。JavaBean 一般情况下须满足以下要求。

(1) JavaBean 是一个公有类，提供无参的公有的构造方法。

(2) JavaBean 的属性是私有的。

(3) 提供具有公有的访问属性的 getter 和 setter 方法。

符合上述条件的类，我们都可以将其看成 JavaBean 组件。在程序中，开发人员所要处理的无非是业务逻辑和数据，而这两种操作都可以使用 JavaBean 组件。一个应用程序中会使用很多 JavaBean。由此可见，JavaBean 组件是应用程序的重要组成部分。

2.4.3 封装数据和业务

1．封装数据

来看一个简单的 JavaBean，通过 MyEclipse 集成开发工具，在 restaurant 项目的 src 下创建 com.restaurant.bean 包，在包中创建 Admin 类，类的属性如下：

```
package com.restaurant.bean;
public class Admin {
    private Integer id;              //管理员 id
    private String loginName;        //登录名称
    private String loginPwd;         //登录密码
}
```

在 Admin 类中，可以添加相应的无参数的构造方法。MyEclipse 提供了一个方便快捷的生成 getter 和 setter 的方法，在相应的代码区，选择 source→ Generate Getters and Setters…，在对话框中，选择相应的属性，单击 OK 按钮即可自动添加相应的 getter 和 setter 方法，代码如下：

```
package com.restaurant.bean;
public class Admin {
    private Integer id;          //管理员 id
    private String loginName;    //登录名
    private String loginPwd;     //登录密码
    //无参的构造方法
    public Admin() {
    }
    //添加相应属性的 setter 方法和 getter 方法
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getLoginName() {
        return loginName;
    }
    public void setLoginName(String loginName) {
        this.loginName = loginName;
    }
    public String getLoginPwd() {
        return loginPwd;
    }
    public void setLoginPwd(String loginPwd) {
        this.loginPwd = loginPwd;
    }
}
```

这是一个典型的封装数据的 JavaBean。这个 JavaBean 封装了管理员用户的数据，如 id、loginName、LoginPwd 等属性，外部通过 getter/setter 方法可以对这些属性进行操作。

2．封装业务

在编写程序时，一个封装数据的 JavaBean 一般情况下对应着数据库内的一张表(或视图)，JavaBean 的属性与表(或视图)内字段的属性一一对应。同样，相对于一个封装数据的 JavaBean，一般都会有一个封装该类的业务逻辑和业务操作的 JavaBean 相对应。

若与封装数据的 Admin.java 相对应的封装业务 JavaBean 是 AdminControl.java，那么可以在 AdminControl.java 中编写有关管理员表操作的方法。例如，获取 admin 表中的最大编号，

结构代码如下。

```
package com.restaurant.bean;
public class AdminControl {
    // 获取最大的 Id 号，此方法并未实现
    public int getMaxId(){
        int num=0;
        String sql="select max(id) from admin";
        // 省略访问数据库的实现，num 得到最大 ID 号
        return num;
    }
}
```

2.4.4 JSP 与 JavaBean

现在已经掌握了如何创建封装数据的 JavaBean 和封装业务逻辑的 JavaBean，那么在 JSP 页面中如何使用 JavaBean？在 JSP 页面中，可以像使用普通类一样实例化一个 JavaBean 对象，调用它的方法。在项目的 WebRoot/ch02/目录下，新建 showJavaBean.jsp 页面，在 JSP 中引入并使用 JavaBean，代码如下：

```
<%@ page import="com.restaurant.bean.*" %>
<%  //使用 JavaBean
AdminControl ac=new AdminControl();
Admin admin=new Admin();
admin.setId(ac.getMaxId()+1);
admin.setLoginName("yzpc");
admin.setLoginPwd("yzpc");
%>
```

在 JSP 中使用 JavaBean 就像在 Java 程序中编写类一样，实例化 JavaBean 后，就可以使用其中的方法了。

2.5 EL 表达式

在 JSP 页面中，为了实现与用户的动态交互，或者控制页面输出，需要在 JSP 页面中嵌入很多 Java 代码，这样不利于对页面的维护和更新，因此 JSP 2.0 引入了 EL 表达式。

2.5.1 EL 表达式概述

EL 的全称是 Expression Language，它是借鉴了 JavaScript 和 XPath 的表达式语言。EL 定义了一系列隐含对象和操作符，使开发人员能够很方便地访问页面的上下文，以及不同作用域内的对象，而无须在 JSP 页面中嵌入 Java 代码，从而使开发人员即使不懂 Java 也能轻松编写 JSP 程序。

EL 表达式提供了在 Java 代码之外，访问和处理应用程序数据的功能，通常用于在某个作用域(page、request、session、application 等)内取得属性值，或者做简单的运算和判断。EL 表达式有如下特点。

(1) 自动类型转换。EL 借鉴了 JavaScript 多类型转换无关性的特点，在使用 EL 得到某个数据时可以自动进行类型转换，因此对于类型的限制更加轻松。

(2) 使用简单。与在 JSP 页面中嵌入 Java 代码相比，EL 表达式使用起来非常简单。

2.5.2 EL 表达式的使用

EL 表达式以“${”开始，以“}”结束，语法如下：

```
${ EL expression }
```

在 showJavaBean.jsp 页面的小脚本中，使用 session 封装管理员对象，代码如下：

```
<%  session.setAttribute("admin", admin);
    response.sendRedirect("showEL.jsp");
%>
```

在 restaurant 项目的 WebRoot/ch02/目录下，新建 showEL.jsp 页面。在 showEL.jsp 页面中使用传统方法取得管理员的名称，代码如下：

```
<%  Admin admin = (Admin)session.getAttribute("admin");
    String loginName = admin.getLoginName();
%>  用户名：<%=loginName %><br>
```

也可在 showEL.jsp 页面中使用 EL 表达式取得管理员的名称，等价的写法如下：

```
${sessionScope.admin.loginName}
```

两者相比，可以发现，EL 的语法比传统的 JSP 代码更为方便、简洁。EL 提供“.”和“[]”两种操作符来存取数据。

(1) 点操作符。

EL 表达式通常由两个部分组成：对象和属性。就像在 Java 代码中一样，在 EL 表达式中也可以用点操作符访问对象的某个属性，例如，通过${sessionScope.admin.loginName}可以访问 admin 对象的 loginName 属性。

(2) “[]”操作符。

与点操作符类似，“[]”操作符也可以访问对象的某个属性，例如，${sessionScope["admin"] ["loginName"] }可以访问管理员对象的登录名称属性。

除此之外，“[]”操作符还提供了更加强大的功能。

- 在属性名中包含特殊字符如“.”或“—”等的情况下，就不能使用点操作符来访问，而只能使用“[]”操作符。
- 访问数组，如果有一个对象名为 array 的数组，那么我们可以根据索引值来访问其中的元素，如${array[0]}、${array[1]}等。

在 showEL.jsp 页面中，分别调用 EL 表达式的两种操作符进行国家和城市的输出显示，代码如下：

```
使用 EL 的"."取得用户名：${sessionScope.ADMIN.loginName } <br>
使用 EL 的"[]"取得用户名：${sessionScope["ADMIN"]["loginName"] }<br>
<%
   Map countries=new HashMap();       // 定义集合 Map
```

```
    countries.put("CN", "China");
    countries.put("RU", "Russia");
    request.setAttribute("countries", countries);
    List cities=new ArrayList();       // 定义集合 List
    cities.add(0,"BeiJing");
    cities.add("ShangHai");
    request.setAttribute("cities", cities);
 %>
国家：${countries.CN }<br>
      --城市：${cities[0] }<br>
      --城市：${cities[1] }<br>
国家：${countries.RU}<br>
```

重新部署项目，在浏览器中访问 http://localhost:8080/restaurant/ch02/showJavaBean.jsp，然后跳转到 showEL.jsp 页面，效果如图 2-24 所示。

图 2-24　EL 表达式操作符的使用

2.5.3　EL 隐式对象

JSP 提供了 page、request、session、application、pageContext 等若干隐式对象。这些隐式对象无须声明，就可以很方便地在 JSP 页面脚本中使用。EL 隐式对象可以分为五类，如表 2-8 所示。

表 2-8　EL 隐式对象的分类

类　别	对象标识符	作　用
JSP 隐式对象	pageContext	提供对页面信息和 JSP 内置对象的访问
作用域访问对象	pageScope	与页面作用域 page 中的属性相关联的 Map 类
	requestScope	与请求作用域 request 中的属性相关联的 Map 类
	sessionScope	与会话作用域 session 中的属性相关联的 Map 类
	applicationScope	与应用程序作用域 application 中的属性相关联的 Map 类
参数访问对象	param	按照参数名称访问单一请求值的 Map 对象
	paramValues	按照参数名称访问数组请求值的 Map 对象
请求头访问对象	header	与请求头名称相对应的字符串的 Map 集合
	headerValues	与请求头名称相对应的字符串数组的 Map 集合
	cookie	所有 cookie 组成的 Map 集合
初始化参数对象	initParam	Web 应用程序上下文初始化参数的 Map 集合

1．JSP 隐式对象

为了能够方便地访问 JSP 隐式对象，EL 表达式引入了 pageContext，它是 JSP 和 EL 的一个公共对象，通过 pageContext 可以访问其他 JSP 内置对象(request、response 等)，这也是 EL 表达式语言把它作为内置对象的一个主要原因。

2．作用域访问对象

在 JSP 页面中定义和设置一个变量，同时指定该变量的作用域，作用域共有四个选项：page、request、session 和 application。在 EL 表达式中，为了访问这四个作用域内的变量和属性，提供了 pageScope、requestScope、sessionScope、applicationScope 这四个作用域访问对象。当使用 EL 表达式访问某个属性时，应该指定查找的范围，如${requestScope.admin}，即在请求(request)范围内查找属性 admin 的值。如果程序中不指定查找范围，则系统会按照 page→request→session→application 的顺序查找。

3．参数访问对象

参数访问对象是与页面输入参数有关的隐式对象，包含 param 和 paramValues 两个对象，通过它可以得到用户的请求参数。两者的不同之处在于，param 对象用于得到请求中单一名称的参数，而 paramValues 对象用于得到请求中的多个值。例如，用户注册时，通常只要填写一个用户名，但可以选择多个兴趣爱好，用户名可以通过 ${param.username}来访问用户名；通过 ${paramValues.habits }可以得到用户所选择的兴趣爱好。

4．请求头访问对象

请求头访问对象用于访问 HTTP 请求头，header 储存用户浏览器和服务器端用来沟通的数据。例如，要取得用户浏览器的版本，可以使用${header["User-Agent"]}。另外在极少情况下，有可能同一标头名称拥有不同的值，此时必须改为使用 headerValues 来取得这些值。要取得 cookie 中设定名称为 userCountry 的值，可以使用${cookie.userCountry}。

5．初始化参数对象

初始化参数对象 initParam，这个映射可用于访问初始化参数的值，初始化参数的值一般都在 web.xml 中设置。例如，一般的方法“String userid = (String)application.getInitParameter ("userid");”也可以使用 ${initParam.userid}来取得 userid。

2.6 JSTL 标签

通过 EL 表达式，在一定程度上简化了 JSP 页面开发的复杂度。但 EL 表达式不能实现复杂业务逻辑的处理，业务逻辑的处理还是要通过脚本的方式来实现。JSTL 标签可以不用在 JSP 页面中嵌入 Java 代码，又能在 JSP 中控制程序流程。JSTL 主要提供了五大类标签库：核心标签库、格式标签库、SQL 标签库、XML 标签库和函数标签库。

2.6.1 JSTL 标签概述

JSTL(JavaServer Pages Standard Tag Library，JSP 标准标签库)包含在 JSP 开发中经常用到的一组标签库，这些标签为我们提供了一种不用嵌入 Java 代码，就可以开发复杂 JSP 页面的途径。使用 JSTL 标签库是为了弥补 html 标签的不足和规范自定义标签。使用 JSLT 标签的目的就是不希望在 JSP 页面中出现 Java 逻辑代码。

JSTL 是由 Sun 公司推出、由 Apache Jakarta 组织负责维护的用于编写和开发 JSP 页面的一组标准标签。作为开源的标准技术，它一直在不断地完善。JSTL 标签库包含各种标签，如通用标签、条件判断标签、迭代标签等。

2.6.2 JSTL 标签的使用

1. 在工程中引用 JSTL 的 jar 包

在工程中引用 1.1 版本的 JSTL，要添加 jstl.jar 和 standard.jar 两个包，放置到项目的 WebRoot/WEB-INF/lib 目录下即可。

在 MyEclipse 集成开发环境中已经集成了 JSTL，选择 File→New→Web Project 命令，在弹出的 New Web Project 对话框中，在 JSTL Version 下拉列表框中选择 1.2.2 最高版本，如图 2-25 所示。

其余步骤与第 1 章的项目创建一致，最后完成后，MyEclipse 会自动在项目中添加相应版本所需的 jar 包和标签库描述文件。

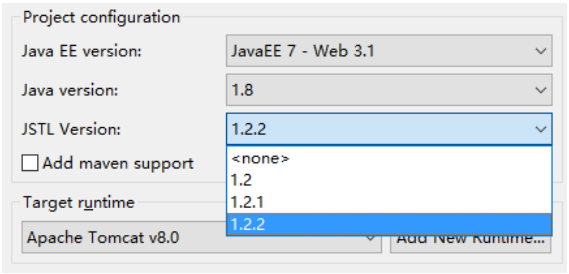

图 2-25 项目中添加 JSTL

2. 在需要使用 JSTL 的 JSP 页面中使用 taglib 指令导入标签库描述文件

如果需要使用 JSTL 核心标签库，需要在页面上方增加如下一行指令：

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

如果需要添加 SQL 标签库、格式标签库、XML 标签库和函数标签库，使用的指令如下：

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
```

完成以上步骤，就可以用 JSTL 方便地开发 JSP 页面，而无须嵌入 Java 代码了。

2.6.3 JSTL 核心标签库

核心标签库在 JSTL 中占有十分重要的地位，该标签库的工作是对 JSP 页面一般处理的封装。使用这些标签能够实现 JSP 页面的基本功能，减少编码工作。核心标签库按功能的不同又分为通用标签库、条件标签库、迭代标签库等。

1. 通用标签库

通用标签用于在 JSP 页面内设置、删除和显示变量，包含三个标签：<c:set>、<c:out>和<c:remove>。

1) <c:set>标签

<c:set>标签用于定义变量，并将变量存储在 JSP 范围中或者 JavaBean 属性中，其语法格式有如下两种。

(1) 将 value 值存储到范围为 scope 的变量 variable 中，语法格式如下：

```
<c:set var="variable" value="v" scope="scope" />
```

- var 属性的值是设置的变量名。
- value 属性的值是赋予变量的值。
- scope 属性对应的是变量的作用域，可选值有 page、request、session、application。

(2) 将 value 值存储到 target 对象的属性中，语法格式如下：

```
<c:set value="value" target="target" property="property" />
```

- target 属性是操作的对象，可以使用 EL 表达式表示。
- property 属性对应对象的属性名。
- value 属性是赋予对象属性的值。

2) <c:out>标签

<c:out>标签用来显示数据的内容，类似 JSP 中的表达式。但功能更加强大，代码也更加简洁，方便页面维护。语法格式分为指定默认值和不指定默认值两种形式。

(1) 不指定默认值，语法格式如下：

```
<c:out value="value" />
```

value 属性是指需要输出的值，可以用 EL 表达式输出某个变量。

(2) 指定默认值，语法格式如下：

```
<c:out value="value" default="default" />
```

default 属性是 value 属性的值为空时，输出默认的值。

3) <c:remove>标签

<c:remove>标签用于移除指定范围内的变量，作用与<c:set>标签相反，语法格式如下：

```
<c:remove var="value" scope="scope" />
```

- var 属性是指待删除的变量的名称。
- scope 属性是指删除的变量所在的范围，可选值有 page、request、session、application，如果没有指定，则默认为 page。

下面通过示例，从语法的角度看一下如何在 JSP 中应用 JSTL 通用标签。在 restaurant 项目的 WebRoot/ch02/目录下，新建 jstlGeneral.jsp 页面，代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
     <title>JSTL 通用标签库使用</title>
  </head>
  <body>
```

```
    设置变量之前的 message 值：<c:out value="${message }" default="null" /> <br>
    <c:set var="message" value="Hello World!" scope="page"></c:set>
    设置新值以后的 message 值：<c:out value="${message }"></c:out> <br>
    <c:remove var="message" scope="page" />
    移除变量 message 以后的值：<c:out value="${message }" default="null" />
  </body>
</html>
```

在该示例中，首先使用<c:set>标签在 page 范围内设置一个变量的值，通过<c:out>标签把该变量显示在页面上，然后用<c:remove>标签在 page 范围内删除该变量，并使用<c:out>标签检查该变量是否已经删除。运行页面，显示效果如图 2-26 所示。

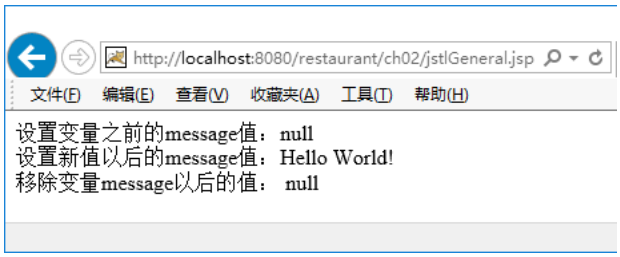

图 2-26　使用 JSTL 设置变量

2．条件标签库

对于包含动态内容的 Web 页面，若希望不同类别的用户看到不同形式的内容，就需要用到 JSTL 的另外一个常用标签，即条件标签。

1) <c:if>标签

<c:if>标签用来执行流程的控制，其功能和语言中的 if 完全相同，其语法格式如下：

```
<c:if test="condition" var="varName" scope="scope">
   // 本部分的内容
</c:if>
```

- test 属性是此条件标签的判断条件，当 test 中表达式的结果为 true 时，会执行本部分的内容，如果为 false 则不会执行。
- var 属性定义变量，该变量存放判断以后的结果，该属性可以省略。
- scope 属性是指 var 定义变量的存储范围，可选值有 page、request、session、application，该属性可以省略。

在 restaurant 项目的 WebRoot/ch02/目录下，新建 jstlCondition.jsp 和 doJstlCondition.jsp 页面，实现登录和处理。

jstlCondition.jsp 页面主要是用户登录表单，使用 jstl 的条件标签，其代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>JSTL 条件标签库使用</title>
  </head>
  <body>
    <c:set var="isLogin" value="${not empty sessionScope.LOGIN_NAME}"/>
    <c:if test="${not isLogin }">
    <form name="form1" method="post" action="doJstlCondition.jsp">
        用户名：<input type="text" name="username" id="username">
        密码：<input type="password" name="password" id="password">
```

```
        <input type="submit" value="登录">
    </form>
    </c:if>
    <c:if test="${isLogin }">
        <c:out value="${sessionScope.LOGIN_NAME}"/>已经登录!
    </c:if>
  </body>
</html>
```

doJstlCondition.jsp 页面主要使用登录请求处理并使用 session 对象设置封装用户名，其代码如下：

```
<%
    request.setCharacterEncoding("utf-8");
    String username = request.getParameter("username");
    String password = request.getParameter("password");
    if(username.equals("admin") && password.equals("admin")){
        //此处暂不访问数据库
        session.setAttribute("LOGIN_NAME", username);//设置登录信息
        response.sendRedirect("jstlCondition.jsp");
    }else{
        out.print("用户名或密码错误! ");
    }
%>
```

重新部署项目，访问 jstlCondition.jsp 页面，如果用户尚未登录，则显示登录页面，如图 2-27 所示。

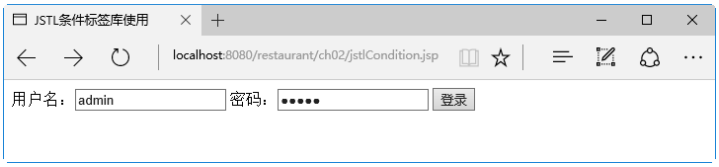

图 2-27　使用<c:if>判断是否登录(1)

如果用户已登录，则显示登录的用户名，如图 2-28 所示。

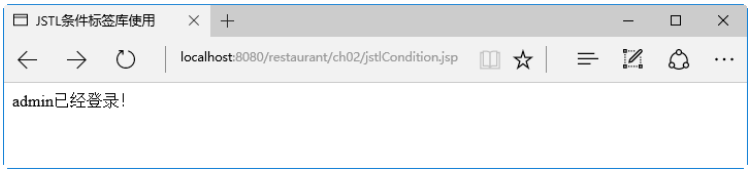

图 2-28　使用<c:if>判断是否登录(2)

2)　<c:choose>标签

<c:choose>和<c:when>、<c:otherwise>一起实现互斥条件执行，类似于 Java 中的 if-else。<c:choose>一般作为<c:when>、<c:otherwise>的父标签，示例代码如下：

```
<c:choose>
    <c:when test="${row.v_money<10000 }">
        初级者
    </c:when>
```

```
   <c:when test="${row.v_money>=10000 && row.v_money<20000 }">
      中级者
   </c:when>
   <c:otherwise>
      高级者
   </c:otherwise>
</c:choose>
```

3．迭代标签库

在 JSP 中，迭代是经常需要用到的操作，主要有两种：<c:forEach>和<c:forTokens>。

1) <c:forEach>标签

通过 JSTL 的<c:forEach>标签，能在很大程度上简化迭代操作，<c:forEach>标签的语法格式如下：

```
<c:forEach var="varName" items="collection" varStatus="statusName"
begin="beginIndex" end="endIndex" step="step">
       // …
</c:forEach>
```

- var 属性是对当前成员的引用。即如果当前循环到第一个成员，那么 var 就引用第一个成员；如果当前循环到第二个成员，它就引用第二个成员；以此类推。
- items 指被迭代的集合对象。
- varStatus 属性用于存放 var 引用的成员的相关信息，如索引等。
- begin 属性表示开始位置，默认为 0，该属性可以省略。
- end 属性表示结束位置，该属性可以省略。
- step 属性表示循环的步长，默认为 1，该属性可以省略。

下面以一个管理员信息的显示为例，来体会一下迭代标签给我们带来的便利之处。在 restaurant 项目的 WebRoot/ch02/目录下，创建 jstlIterate.jsp 页面，代码如下：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@page import="com.restaurant.bean.Admin"%>
<%   List adminList=new ArrayList();
   Admin admin=null;       // 2.4.3 小节已创建 Admin 类，此处直接使用
   for (int i = 0; i < 10; i++) {
      admin=new Admin(i+1,"管理员"+(i+1),"密码"+(i+1));
      adminList.add(admin);
   }
     request.setAttribute("ADMINLIST", adminList);
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>  <title>JSTL 迭代标签的使用</title>  </head>
  <body>
    <table border="1" width="80%" align="center">
       <tr>
          <td>ID 号</td>  <td>用户名</td>  <td>密码</td>
       </tr>
       <c:forEach var="admin" items="${requestScope.ADMINLIST }"
varStatus="status">  <!-- 循环输出管理员信息 -->
```

```
          <tr <c:if test="${status.index % 2 ==1 }">style=
"background-color:yellow;"</c:if>> <!-- 若为偶数行，更换背景颜色 -->
              <td>${admin.id }</td>
              <td>${admin.loginName }</td>
              <td>${admin.loginPwd }</td>
          </tr>
        </c:forEach>
    </table>
  </body>
</html>
```

该示例的运行效果如图 2-29 所示。

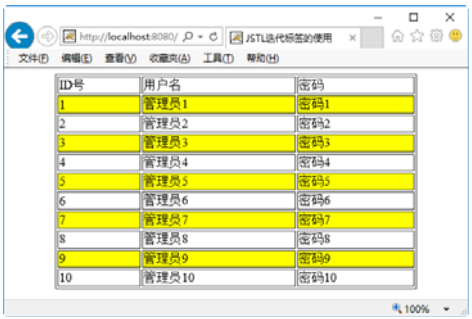

图 2-29 <c:forEach>迭代标签示例

2) <c:forTokens>标签

<c:forTokens>标签专门用于处理字符串的迭代，可以指定一个或多个分隔符，语法格式如下：

```
    <c:forTokens items="stringOfTokens" delims="delimiters" var="varName"
begin="begin" end="end" step="step"  varStatus="varStatusName">
        相应内容
    </c:forTokens>
```

在 jstlIterate.jsp 页面后面添加示例代码，具体如下：

```
<c:forTokens items="China,Russia,France" delims="," var="item">
    <c:out value="${item }"/> <br>
</c:forTokens>
```

2.7 小　　结

本章主要讲解了 JSP 技术，包括 JSP 页面组成，JSP 内置对象，对象范围，在 JSP 中使用 JavaBean、EL 和 JSTL 等。通过学习和掌握 JSP 的主要内容，读者能够掌握动态网站的相应开发技术。