第 2 章　JSP 语法

知识目标

◎ JSP 的基本格式
◎ JSP 的注释方法
◎ JSP 的指令元素
◎ JSP 的脚本元素
◎ JSP 的动作元素

技能目标

◎ 掌握 JSP 的基本格式
◎ 掌握 JSP 的注释方法
◎ 掌握 JSP 的指令元素的应用
◎ 掌握 JSP 的脚本元素的应用
◎ 掌握 JSP 的动作元素的应用

和掌握其他编程语言一样，要掌握 JSP，需要了解 JSP 的语法。如果读者已经学过 Java 语言，那么掌握 JSP 的语法将是件非常容易的事情，因为 Java 和 JSP 的语法是非常相似的。但是 JSP 文件的构成却和 Java 文件的构成不同，因此，除了了解 JSP 语法，还需要对 JSP 文件的构成有清晰的了解，才能编写出正确的 JSP 代码，生成正确的 JSP 文件。本章将从 JSP 的构成开始，通过对 JSP 结构的分析，来逐一介绍 JSP 语法中的注释方法、指令元素、脚本元素以及动作元素，从而使读者掌握如何正确地编写 JSP 页面程序。

2.1　JSP 的基本格式

通过第 1 章的介绍可以看出，JSP 主要是通过在 HTML 标签语言中嵌入 Java 脚本语言来实现页面动态请求的。换句话说，编写 JSP，可以使用编辑 HTML 的文本工具，在编辑完成后将其保存成“*.jsp”文件即可。

JSP 的基本格式如下：

```
<html>
    <head>
        <title>JSP 页面的标题</title>
    </head>
```

```
    <body>
        …//HTML 标签语言
        <%
            //嵌入 Java 脚本语言编写的程序代码
        %>
        …//HTML 标签语言
    </body>
</html>
```

在 JSP 的基本格式中，如果去掉“<%”和“%>”之间的内容，剩下的内容也叫做 JSP 的模板元素。所谓模板元素，是指 JSP 的静态 HTML 或者 XML 内容，对于 JSP 的显示是非常必要的。因为模板元素是网页的框架，决定了 JSP 页面的组成和显示效果的美观程度。在进行 JSP 编译时，模板元素也被编译到 Servlet 里，而在客户请求 JSP 时，模板元素又一字不变地被发送到客户端显示出来。在实际的应用开发中，这部分工作往往都由专门的网页美工人员来完成。

在嵌入由 Java 脚本语言编写的程序代码时，可以通过<%! Java 声明 %>格式来声明变量或者方法；使用<%=Java 表达式%>来引用一个表达式的值。更一般的做法是通过<%Java 脚本语言%>的格式来嵌入 Java 脚本语言。

2.2　JSP 的注释方法

在编写程序时，注释是不可缺少的，这也是程序员应该养成的一个良好习惯。合理、详细的注释对于代码的阅读和后期维护是非常重要的，对于多人组成的团队开发来说就更是如此。和其他编程语言一样，JSP 中也提供了多种注释方法：HTML 的注释方法、JSP 隐藏注释和 JSP 脚本注释。不同的注释方法的语法规则和运行效果均不相同，下面将逐一讲解。

2.2.1　HTML 的注释方法

由于 JSP 是由 HTML 标签和嵌入的 Java 脚本语言组成的，所以 HTML 中的注释方法很自然地被 JSP 使用。注释格式为：

<! -- 注释内容 -->

使用这种注释方法，在客户端浏览时，用户是看不到注释的内容的。但是当用户查看源代码时，则可以看到注释的内容。所以，这种注释方法是不安全的，编程人员最好不要在这种注释中添加需要保密的内容。

虽然 JSP 沿袭使用了 HTML 的注释方法，但 JSP 中的这种注释方法还是和 HTML 注释有不同的地方，就是可以在注释中使用表达式。使用格式为：

<! -- 注释内容<%=expression%>-->

包含该注释语句的 JSP 页面被请求后，服务器能够识别注释中的 JSP 表达式，然后执行该表达式，而对注释中的其他内容不作任何操作。当服务器将执行结果返回给客户端后，客户端浏览器会识别该注释语句，所以被注释的内容不会显示在浏览器中。

对于一些对系统的响应速度要求比较高的站点，最好不要使用这种注释方法，因为这种注释方法会加大网络的传输负担。

2.2.2 JSP 隐藏注释

顾名思义，隐藏注释是写在 JSP 程序里，但不发给客户的一种注释方法。这种注释方法所注释的内容不仅在客户端浏览时看不到，即使是通过在客户端查看 HTML 源代码也不会看到，所以比 HTML 注释方法的安全性高。其注释格式为：

<%-- 注释内容 --%>

如果在 JSP 文件中包含如下代码：

```
<%-- 获取当前时间 --%>
<p>当前时间为：<%=(new java.util.Date()).toLocaleString()%></p>
```

访问该页面后，将会在客户端浏览器中生成如下内容：

当前时间为：2010-7-24 15:50:15

通过查看 HTML 源代码将会看到如下内容：

```
<p>当前时间为：2010-7-24 15:50:15</p>
```

但是“获取当前时间”字样的内容却被隐藏起来，用户无法看到。

用隐藏注释标记的字符在 JSP 编译时会被忽略。

2.2.3 JSP 脚本注释

JSP 脚本是嵌入到“<%”和“%>”标记对之间的程序代码。由于脚本中包含的是 Java 代码，所以 Java 中的注释方法在 JSP 中也是适用的。Java 中用“//”表示单行注释，用一对“/*”和“*/”来表示多行注释，这两种注释方法都可以直接应用在 JSP 中，称为 JSP 脚本注释方法。

JSP 脚本注释的格式为：

```
<%
    Java 代码
    //单行注释
    Java 代码
    /* 多行注释
       多行注释 */
%>
```

2.2.4 混合使用多种注释

在实际的应用开发中，往往会混合使用上面介绍的注释方法。下面用一段最简单的输出“Hello World!”字样的 JSP 代码来展示如何在 JSP 中混合使用多种注释方法：

```
<html>
    <head>
        <title>Hello World</title>
    </head>
    <body>
        <! —— 这是一个 HTML 的注释 ——>
        <% —— 这是一个 JSP 隐藏注释 —— %>
        <%
            /* JSP 脚本部分,这里采用 Java 语言中的字符串输出功能
                在界面上输出"Hello World!"字样 */
            out.println("Hello World!"); //在屏幕上打印输出
        %>
    </body>
</html>
```

在上面的这个例子中,综合使用了本章介绍的 3 种注释方法,读者可以自行编制 JSP 程序,并在浏览器端查看源代码,看看哪些内容是可见的,哪些内容是不可见的。

前面提到过对于对系统的响应速度要求比较高的站点,最好不要使用 HTML 的注释方法对其中的代码进行注释,但这不是绝对的。编程人员可以在编写过程中加入注释,但在站点发布时将其中的注释去掉。

2.3　JSP 的指令元素

在一个 JSP 文件中,除了基本的模板元素,还经常会出现一些指令元素。指令元素的作用不是进行逻辑处理或产生输出代码,而是通过指令中的属性配置向 JSP 容器发出一些指示,从而控制 JSP 页面的某些特性。换句话说,指令是用来设置 JSP 的全局变量、声明类、要实现的方法和输出内容的类型等,为编译阶段提供全局信息。所有的指令都在 JSP 整个文件范围内有效。

指令的通用格式为:

<% @ 指令名称 属性="属性值"… %>

常用的指令元素有:

- page 指令:用于对 JSP 文件中的全局属性进行设置。
- include 指令:用于声明用户自定义的新标签。
- taglib 指令:用于在 JSP 页面中引用外部文件。

下面对 JSP 中的指令元素进行详细的讲解。

2.3.1　page 指令

page 指令用于定义 JSP 文件中的全局属性,可以放在 JSP 页面的任何位置,但为了便于程序的阅读,一般置于页面的首部。一个 JSP 页面可以包含多个 page 指令,在编译过程中,所有的 page 指令都被抽取出来并同时应用到一个页面里。

page 指令中包含了很多属性，其设置格式为：

```
<%@page
    [language="java"]
    [extends="package.class"]
    [import="{package.class|package.* },…"]
    [contentType="TYPE=text/html; charset=CHARSET"]
    [session="true|false"]
    [buffer="none|8kb|sizekb"]
    [autoFlush="true|false"]
    [isThreadSafe="true|false"]
    [info="text"]
    [errorPage="relativeURL"]
    [isErrorPage="true|false"]
    [pageEncoding="PEinfo"]
%>
```

需要说明的是，这些页面属性可以单独使用，也可以同时使用几个。在设置多个属性时，其间以分号隔开。

除了 import 属性外，其他属性只能出现一次。

表 2-1 列出了这些属性的详细解释。

表 2-1 page 指令的属性解释

属　性	描　述	默认值	例　子
language	设置当前页面中编写 JSP 脚本使用的语言	"java"	language="java"
extends	JSP 被编译成 Servlet 的 Java 程序，该属性设置 Servlet 继承的超类	"package.class"	—
import	设置要导入页面的 Java 包。如果有多个包，则用逗号隔开。在 JSP 中默认导入了 lang 和 servlet 包	默认忽略	import="java.io.*, javax.servlet.jsp.*"
contentType	设置页面内容的类型。内容的类型可以是一个简单的类型规范，或者一个类型规范和一个字符编码，通常被设置为 text/html，XML 风格的标签的默认值是 text/xml	TYPE=text/html CHARSET=iso8859-1	contentType="text/html; charset=gb2312"
session	表示当前页面是否支持一个会话	true	session="true"
buffer	定义输出流缓冲区的大小，与 autoFlush 一起使用	8kb	buffer="64kb"
autoFlush	设置输出流的缓冲区是否自动清除	true	autoFlush="true"

续表

属　性	描　述	默 认 值	例　子
isThreadSafe	设置 JSP 页面是否能够同时处理一个以上的页面请求	true	isThreadSafe="true"
info	指定 JSP 页面的信息	默认忽略	info="测试页面"
errorPage	定义该页面出现异常时要调用的页面，与 isErrorPage 一起使用	默认忽略	errorPage="error.jsp"
isErrorPage	表明当前页是否作为其他页面的错误处理	false	isErrorPage="false"
pageEncoding	设置当前页面的字符编码	"iso8859-1"	pageEncoding="gb2312"

2.3.2　include 指令

和 page 指令不同，include 指令是用来引用外在文件的，即在当前页面嵌入其他页面，如果被嵌入的页面中有可执行的代码，会显示代码执行后的结果。确切地说，include 指令将在 JSP 编译时插入一个包含文本或者代码的文件。当使用 include 指令时，这个包含的过程是静态的，这个被包含的文件所执行的结果将会插入到 JSP 文件中出现“<%@ include%>”的地方。一旦包含文件被执行完毕，当前的 JSP 文件的过程将会被恢复，继续执行下一行代码。

include 指令的格式为：

```
<%@ include file="文件路径" %>
```

include 指令只有一个 file 属性，用于在代码当前位置指定包含的文件名。被包含的文件可以是任何 HTML 或者 JSP 文件，甚至可以是代码片段。file 属性的值是一个 URL，但不能是一个可传入参数的 URL。

如果修改了由 include 指令嵌入的页面，则必须对整个单元重新编译。

采用 include 指令，可以很好地解决页面代码冗余问题，即将一些共性的内容（如导航栏、页脚消息等）写入到一个单独的页面中，然后通过 include 指令加入到其他 JSP 页面，从而降低页面代码的冗余问题，并且修改起来也更加方便。

除了采用 include 指令在 JSP 页面中嵌入其他页面，JSP 还提供了另一种包含其他文件的方法，即

```
<jsp:include page=""/>
```

这种方法也用到了 include 指令，确切地说，这种方法是使用了 JSP 中的动作元素。该方法是在 JSP 页面运行阶段才包含进其他页面或文件，被包含的对象可以是静态的，也可以是动态的。在后续章节中，将对这种方法的使用进行详细的阐述。

下面的程序演示了 include 指令的用法：

参考程序 2-1：

(1)建立 01.jsp 文件，将其保存在“\ch02”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<html>
    <head>
        <title>加载文件样例</title>
    </head>
    <body>
        <center>
            <h2>下面是被加载的文件内容</h2>
            <hr/>
            <font color="red"><%@ include file="./01.htm"%></font>
        </center>
    </body>
</html>
```

(2)建立 01.htm 文件，该文件表示被 01.jsp 通过 include 指令加载，其保存位置与 01.jsp 相同，其内容如下：

```
<!--/ch02/01.htm:被 include 的页面-->
<html>
    <head>
        <title>被加载文件的标题不会显示</title>
        <meta http-equiv="content-type" content="text/html";charset="utf-8">
    </head>
    <body>
        <p>This is a inserted HTML file</p>
    </body>
</html>
```

(3)在浏览器中输入 http://localhost:8080/jsplearner/ch02/01.jsp，可以查看程序的执行效果，如图 2-1 所示。

图 2-1　程序的执行效果

程序分析 2-1：

在这里，01.jsp 使用 include 指令将 01.htm 文件加载到 JSP 文件中，系统在加载文件时将 HTML 文件作为一个整体嵌入到 JSP 文件。

2.3.3　taglib 指令

标签(tag)是一种 XML 元素。在 JSP 中使用标签，一方面能够使 JSP 网页变得更加简洁，并且易于维护；另一方面可以很方便地实现同一个 JSP 文件支持多种语言版本。标签库(tag library)是由一系列功能相似、逻辑上互相联系的标签构成的集合。JSP 中使用 taglib 指令来引用一个标签库或者自定义标签，该指令能够告诉容器当前 JSP 页面将使用哪些标签库，并给引用的标签库指定一个前缀。当 JSP 页面再次用到这个标签库时，就可以使用指定的前缀来标识标签库。

taglib 指令的格式为：

```
<%@ taglib uri="tagLibraryURL" prefix="tagPrefix" %>
```

其中，属性 uri 指定了 JSP 要在 web.xml 中查找的标签库描述符，也可以是一个相对或绝对的路径；属性 prefix 指定了在页面中使用由 uri 属性指定的标签库的前缀，但不可使用 jsp、jspx、java、javax、servlet、sun 和 sunw 作为前缀。

由于标签是 XML 元素，所以它的名称和属性都是大小写敏感的。

2.4　JSP 的脚本元素

JSP 的脚本元素是 JSP 中使用最频繁的元素。那么什么是 JSP 的脚本元素呢？JSP 的脚本元素就是用 Java 写的脚本代码，用来实现页面的动态请求，主要包括声明(declaration)、表达式(expression)和脚本程序(scriptlet)。在客户端发出 JSP 请求后，JSP 页面中的脚本元素会被服务器端容器编译成 Servlet 文件，然后再动态地执行，所以，对于客户端来说这些元素都是不可见的。

2.4.1　声明

在 JSP 中，声明就是一段 Java 代码，用来定义在产生的类文件中的类的属性和方法，即用于声明或者定义一个 Java 变量或者 Java 方法，类似于在 Java 源代码中的声明。当然，声明中的变量和方法必须是有效的 Java 代码，必须符合所有 Java 语法和语义规则。

声明的格式为：

```
<%! 声明变量或方法 %>
```

声明只会出现在 JSP 页面中，而不会在客户端输出。声明后的变量和方法可以在 JSP 的任意地方使用。也就是说，声明的任何变量都会变成 JSP 页面实现类的实例变量，作用域是整个页面；声明的任何方法都会成为 JSP 页面实现类的实例方法，作用域也是整个页面。

由于实例变量不是线程安全的，所以在使用声明初始化变量时，需要考虑其安全性。

2.4.2 表达式

表达式的作用是向页面中输出某种信息。在 JSP 请求处理阶段，表达式被计算出结果，被转换成字符串后与模板数据组合在一起。表达式在页面的位置，也就是该表达式计算结果所处的位置。

表达式的使用格式为：

<%=变量或有返回值的方法或 Java 表达式%>

需要说明的是，在“<%=”和“%>”之间插入的只能是一个表达式，而不能插入一个语句，且“<%=”是一个完整的符号，中间也不可以有空格。这个表达式必须可以求值，而这个值由服务器负责计算，并将计算结果以字符串形式发送到客户端显示。由于表达式不是合法的代码语句，所以表达式不需要用分号结尾。

如果表达式要输出的是一个对象，则该对象的 toString()方法会被调用，最终输出由 toString()方法返回的内容。

下面的代码演示了如何使用表达式来输出一个变量的值：

```
<html>
    <body>
        <%
            int a=2;
            int b=3;
            int c=a*b;
        %>
        <table><tr><td>Final value:<%=c%></td></tr></table>
    </body>
</html>
```

2.4.3 Scriptlet

Scriptlet 又叫小脚本格式，或者脚本程序，是指在 HTML 中使用“<%”和“%>”嵌入 Java 程序，从而进行相应的逻辑处理。换句话说，Scriptlet 是 JSP 中页面处理请求时执行的 Java 代码。由于 Scriptlet 可以包含任意的 Java 片段，所以在应用上非常灵活，可以执行复杂的操作和控制。

Scriptlet 的使用格式为：

<%脚本代码%>

在 Scriptlet 中定义的变量在当前的整个页面内都有效，但不会被其他线程共享，也就是说一个线程对该变量的操作不会影响到其他线程。当变量所在的页面关闭后，变量也随之被销毁。

下面的代码演示了如何使用 Scriptlet：

```
<%@ page import="java.util.Date" %>
<html>
    <body>
        <%
            Date Current_Date=new Date();
            out.println("当前日期为:"+Current_Date);
        %>
    </body>
</html>
```

在这个程序中，用到了前面讲过的 page 指令，通过 import 属性来应用 Date 类。在 Scriptlet 中初始化了 Date 类的一个实例对象 Current_Date，并通过内置对象 out 中的 println()方法打印当前的日期。关于内置对象的内容，将在后续章节中进行详细的介绍。

2.5　JSP 的动作元素

JSP 定义了一系列的动作元素，使用格式是“<jsp：标记名>”，即用“jsp”作为前缀。JSP 动作元素是在请求处理阶段动态被执行的，每次有客户端请求时，都会被重新执行一次，而指令元素在转译时期被编译执行，且只会被编译一次。

JSP 规范定义了一系列的标准动作，这些标准动作不用考虑容器是如何实现的，但是每个 Web 容器都提供这些操作类型。在制作 JSP 网页时，除了使用 JSP 的标准动作元素外，还可以使用由其他第三方厂商提供的非标准动作。下面将针对常用的几种动作元素进行讲解。

2.5.1　jsp:param

jsp:param 动作主要用于指定某个参数的值，即用来以“名—值”对的形式为其他标签提供附加信息。一般来说，该动作与 jsp:include、jsp:forward 和 jsp:plugin 等动作一起使用。和不同的动作一起使用，就具有不同的功能。

jsp:param 动作的使用格式为：

```
<jsp:param name="paramName" value="paramValue"/>
```

其中，name 为与属性相关联的关键字，value 为属性的值。每个 jsp:param 动作都会创建一个既有名又有值的参数，这样通过 jsp:include 动作所包含的外在文件或者通过 jsp:forward 动作转移到的另外的页面，都可以使用这些参数。

下面的代码可以很好地演示该动作的使用方法：

```
<html>
    <body>
        <jsp:include page="example.jsp">
            <jsp:param name="userName" value="lzw"/>
        </jsp:include>
    </body>
</html>
```

这段代码表明，在所包含的 example.jsp 文件中，请求时可以使用通过 jsp:param 动作定义的 userName 参数。

2.5.2 jsp:include

jsp:include 动作元素用于在 JSP 程序中包含一个静态或动态的文件，在运行效率上比“<%@ include file="文件路径"%>”指令要低，但却可以动态地增加内容。

jsp:include 动作元素的使用格式为：

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

或者：

```
<jsp:include page="filename" flush="true">
    <jsp:param name="paramname" value="paramvalue"/>
    ...
</jsp:include>
```

其中：

(1)page="filename"：通过 page 参数来指定需要包含的文件的相对路径，或者是代表相对路径的表达式。

(2)flush="true"：通过 flush 参数来指定是否将被包含的文件的执行结果输出到客户端，这里必须使用 flush="true"，不能使用 false 值，而缺省值为 false。

(3)<jsp:param name="paramname" value="paramvalue"/>：通过 jsp:param 动作元素来传递一个或多个参数到指定的动态文件，可以在一个页面中使用多个<jsp:param>动作来传递参数。

这里需要着重说明的是：前面介绍的 include 指令是在 JSP 翻译时进行文件的合并，然后对合并的整个文件进行编译；而 jsp:include 动作则是先进行自身的翻译和编译，然后在用户请求阶段进行二进制文件的合并。

下面的例子程序演示了如何使用 jsp:include 动作。

参考程序 2-2：

(1)建立 02-1.jsp 文件，将其保存在“\ch02\”下，并输入如下所示的代码：

```
<%
    String paramvalue=request.getParameter("param");
    out.println("param value is:<br/>"+paramvalue);
%>
```

(2)建立 02-2.jsp 文件，将其保存在“\ch02\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
    <head>
        <title>加载文件样例</title>
    </head>
    <body>
        <center>
            加载 02-1.jsp
```

```
            <h2>
                <jsp:include flush="false" page="./02-1.jsp">
                    <jsp:param name="param" value="true"/>
                </jsp:include>
            </h2>
        </center>
    </body>
</html>
```

(3)在浏览器中输入 http://localhost:8080/jsplearner/ch02/02-2.jsp 并按 Enter 键,可以查看程序的运行效果,如图 2-2 所示。

图 2-2　程序的运行效果 1

程序分析 2-2:

在这里,02-2.jsp 使用了 jsp:include 动作来引用文件 02-1.jsp,并传递参数,参数名为“param”,参数值为“true”。被引用的页面可以通过 request.getParameter("param")得到参数名“param”对应的值。

语句<jsp:include flush="false" page="./02-1.jsp">中的第一个“.”可有可无,在 Windows 下,“./”表示当前路径下,“../”表示父路径,一般“/”也可以表示当前路径,但是在 UNIX 中,“/”则表示路径。

需要特别说明的是,jsp:include 动作元素的目的是把其他文件插入到这个程序中来,但过度使用会降低执行的效率。如果被引入的文件是静态 HTML 文件,那么这仅仅是把包含文件的内容加到 JSP 文件中,这个文件并不会被 JSP 编译器执行;如果被引入的文件是动态的,则可被 JSP 编译器执行。

2.5.3　jsp:forward

jsp:forward 动作主要将请求转发到另一个 JSP、Servlet 或 HTML 文件。请求被转向到的文件必须位于发送请求相同的上下文环境中,每当遇到转向动作元素时,就停止执行当前的 JSP,转到其他页面上去。

jsp:forward 动作的使用格式为:

```
<jsp:forward page="url"/>
```

或者是:

```
<jsp:forward page="url">
    <jsp:param name="paramname" value="paramvalue"/>
</jsp:forward>
```

其中，page 属性包含一个相对 URL 地址，即该动作元素要定向的文件或 URL，要定向的文件可以是 JSP 文件，也可以是程序段，还可以是其他能够处理 request 对象的文件。而在第二种格式中结合了 jsp:param 动作向一个动态文件发送一个或多个参数，其中，参数可以是一个或者多个值，而这个文件却必须是动态文件。如果要传递多个参数，可以在一个 JSP 文件中使用多个 jsp:param 动作，分别将多个参数发送到一个动态文件中。

下面的例子程序演示了如何使用 jsp:forward 动作。

参考程序 2-3：

(1)建立 03-1.jsp 文件，将其保存在“\ch02\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
    String paramvalue=request.getParameter("param");
%>
<html>
    <head>
        <title>Forward,注意标题变化</title>
    </head>
    <body>
        <center><h2>
            Param value is:<br/>
            <%=paramvalue%>
        </h2></center>
    </body>
</html>
```

(2)建立 03-2.jsp 文件，将其保存在“\ch02\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<html>
    <head>
        <title>Forward 样例</title>
    </head>
    <body>
        <jsp:forward page="03-1.jsp">
            <jsp:param name="param" value="true"/>
        </jsp:forward>
    </body>
</html>
```

(3)在浏览器中输入 http://localhost:8080/jsplearner/ch02/03-2.jsp 并按 Enter 键，可以查看程序的运行效果，如图 2-3 所示。

图 2-3　程序的运行效果 2

程序分析 2-3：

本参考程序的代码内容和参考程序 2-2 类似，只是代码中的 jsp:include 动作变成了 jsp:forward 动作。

2.5.4　jsp:plugin

jsp:plugin 动作被用来生成客户端浏览器的特别标签，可以使用该动作来插入 Applet 或者 JavaBean，即用 Java 插件在浏览器中运行 Java Applet 或 Java Bean。当 JSP 文件被编译并把结果发送到浏览器时，它会根据浏览器的版本替换成<object>或者<applet>标签。也就是说，它们随着页面程序一起传输到客户端，并且在客户端运行。

jsp:plugin 动作的使用格式为：

```
<jsp:plugin type="bean|applet" code="classname" codebase="classFileDirectoryName">
    <jsp:param name="paramname" value="paramvalue">
    </jsp:param>
    <jsp:fallback>text message for user</jsp:fallback>
</jsp:plugin>
```

其中：

(1)type="bean|applet"。该属性指定了插件执行的对象的类型。用户必须在 Bean 或 Applet 中指定一个，因为这个属性没有缺省值。

(2)code="classname"。该属性指定了插件将执行的 Java 类文件的名称，在名称中必须包含扩展名，且此文件必须在用 codebase 属性指明的目录下。

(3)codebase="classFileDirectoryName"。该属性用来指定包含插件将运行的 Java 类的目录或指向这个目录的路径，缺省为此 JSP 文件的路径。

(4)<jsp:param name="paramname" value="paramvalue"/>。jsp:param 动作在这里用于向 Applet 或 Bean 传送参数和参数值。

(5)<jsp:fallback>text message for user</jsp:fallback>。“text message for user”是在 Java 插件能正确启动而 Applet 或 Bean 的程序代码不能找到并被执行时，浏览器为用户显示的出错信息。

习 题 2

1. JSP 中都有哪些注释方式？为了提高站点的安全性，应该采用哪种注释方法？

2. include 指令和 jsp:include 动作有什么共同点和不同点？

3. JSP 中的指令元素和动作元素分别是在什么时候被执行的？

4. 在 2.5 节中所介绍的参考程序的基础上，自行编写用户登录验证的 login.jsp 页面，如果验证成功，将页面 forward 到一个名为 success.jsp 的页面，否则转回到 login.jsp 页面。

第 3 章　JSP 的内置对象

知识目标

◎ request 对象
◎ response 对象
◎ session 对象
◎ out 对象
◎ application 对象
◎ page 对象
◎ config 对象
◎ exception 对象
◎ pageContext 对象

技能目标

◎ 掌握 request、response、session 和 out 的使用方法
◎ 掌握 application、page、exception 和 pageContext 的知识

除了各种动作元素，JSP 还提供了一些内置对象来简化页面的开发。所谓内置对象，就是可以在 JSP 页面中直接调用，而不需要事先定义的对象。这些内置对象实际上都对应着某个 Servlet 类，在 JSP 被翻译成 Servlet 之后，这些内置对象就会被相应地转换为对应的类实例。熟练掌握这些内置对象的使用，会使得 JSP 网页的开发更加简单。本章将通过内置对象的定义、使用方法两个方面，对 JSP 中经常使用到的 request 对象、response 对象、session 对象、out 对象、application 对象、page 对象、config 对象、exception 对象和 pageContext 对象等内置对象逐一进行详细的介绍。

3.1　request 对象

request 对象是 JSP 中最常用的对象之一，代表的是 java. servlet. HttpServletRequest 类的对象。该对象中包含了有关浏览器请求的信息，并提供了多个用于获取 cookie、header 以及 session 内数据的方法。例如，可以查看请求参数的配置情况（调用 getParameter 来实现）、请求的类型（如 get、post、head 等）和已经请求的 HTTP 头（如 cookie、referer 等）。

一般来说，来自客户端的请求经 Servlet 容器处理后，都由 request 对象进行封装，然后作为 jspService()方法的一个参数由容器传递给 JSP 页面。

3.1.1 request 对象的常用方法

request 对象的常用方法有：

• getAttribute(String name)。

该方法用于返回 name 指定的属性的值，如果不存在指定的属性，则返回空值(null)。调用的语法如下：

```
request.getAttribute(String name)
```

其中，参数 name 是指定的属性的名称。

• setAttribute(String name,java.lang.Object obj)。

该方法可以设置名称为 name 的 request 参数的值为 obj。调用的语法如下：

```
request.setAttributeName()
```

• getCookies()。

该方法用于返回客户端的 Cookie 对象，结果是一个 Cookie 数组。调用的语法如下：

```
request.getCookies()
```

• getHeader(String name)。

该方法可以获得 HTTP 协议定义的传送文件头信息，如 request.getHeader("User-Agent")将返回一个用户所用浏览器的版本号和类型。调用的语法如下：

```
request.getHeader(String name)
```

其中，参数 name 是指定的头名称。

• getHeaderNames()。

该方法用于返回所有 request header 的名称，结果保存在一个 Enumeration 类的实例中。调用的语法如下：

```
request.getHeaderNames()
```

• getServerName()。

该方法可以获得服务器的名称。调用的语法如下：

```
request.getServerName()
```

• getServerPort()。

该方法可以获得服务器的端口号。调用的语法如下：

```
request.getServerPort()
```

• getRemoteAddr()。

该方法可以获得服务器的客户端的 IP 地址。调用的语法如下：

```
request.getRemoteAddr()
```

• getRemoteHost()。

该方法可以获得客户端计算机的名称。如果该方法失败，则返回客户端计算机的 IP 地址。调用的语法如下：

```
request.getRemoteHost()
```

• getProtocol()。

该方法可以获得客户端向服务器端传送数据所依据的协议名称。调用的语法如下：

```
request.getProtocol()
```

• getMethod()。

该方法可以获得客户端向服务器端传送数据的方法。调用的语法如下：

request.getMethod()

• getServletPath()。

该方法可以获得客户端所请求的脚本文件的文件路径。调用的语法如下：

request.getServletPath()

• getCharacterEncoding()。

该方法可以获得请求中的字符编码方式。调用的语法如下：

request.getCharacterEncoding()

• getSession([Boolean create])。

该方法用于返回和请求相关的 Session。其中 create 参数是可选的。当有参数 create 且这个值为 true 时，如果客户还没有创建 Session，那么可以建立一个新的 Session。调用的语法如下：

request.getSession()

• getParameter(String name)。

该方法可以获得客户端传送给服务器端的参数值，该参数由 name 指定。调用的语法如下：

request.getParameter(String name)

其中，参数 name 是指定的参数的名称。

• getParameterValues(String name)。

该方法可以获得指定参数的值。调用的语法如下：

request.getParameterValues(String name)

• getQueryString()。

该方法可以获得查询字符串，该字符串由客户端以 GET 方法向服务器传送。调用的语法如下：

request.getQueryString()

• getRequestURI()。

该方法可以获得发送请求字符串的客户端地址。调用的语法如下：

request.getRequestURI()

3.1.2　request 对象使用实例

为了使读者更好地了解 request 对象中的常用方法，下面通过一个例子程序来演示如何调用这些常用方法。

参考程序 3-1：

(1)建立 01.jsp 文件，将其保存在“\ch03\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<html>
    <head>
        <title>request 内置对象中常用方法的使用实例</title>
    </head>
    <body>
        <form action="01.jsp">
```

```
        <br>
        Get request results:<br>
        <input type="text" name="requestParam"><br>
        <input type="submit" value="提交得到查询内容">
    </form>
    返回 HTTP 请求信息中使用的方法名称:<%=request.getMethod()%><br>
    返回请求信息中调用 Servlet 的 URL 部分:
                                <%=request.getServletPath()%><br>
    返回 HTTP GET 请求信息中 URL 之后的查询字符串:
                                <%=request.getQueryString()%><br>
    返回请求实体的 MIME 类型:<%=request.getContentType()%><br>
    返回请求信息中的协议名称和版本号:
                                <%=request.getProtocol()%><br>
    返回有关路径的信息:<%=request.getPathInfo()%><br>
    返回接受请求的服务器主机:<%=request.getServerName()%><br>
    返回服务器的端口号:<%=request.getServerPort()%><br>
    返回提交请求的客户端的名字:<%=request.getRemoteHost()%><br>
    返回请求提交的客户端的 IP 地址:<%=request.getRemoteAddr()%><br>
    返回请求中使用的模式名称:<%=request.getScheme()%><br>
    返回由 request 对象提交的值:
                                <%=request.getParameter("requestParam")%>
  </body>
</html>
```

(2)在浏览器中输入 http://localhost:8080/jsplearner/ch03/01.jsp? requestParam 并按 Enter 键,在输入框内输入 HelloJsp,然后单击“提交得到查询内容”按钮,即可查看程序的运行效果,如图 3-1 所示。

图 3-1　程序的运行效果 1

程序分析 3-1：

在该程序中，页面首先通过＜form＞提交了一个 requestParam 参数，并调用 request.getParameter("requestParam")以获得这个参数的值，同时调用其他 request 方法获取各种请求信息。

3.2　response 对象

response 是和应答相关的 javax.servlet.HttpServletResponse 类的一个对象，它封装了 JSP 产生的响应，然后被发送到客户端以响应客户的请求。换句话说，response 对象和 request 对象的作用恰恰相反，用于响应客户请求并向客户端输出信息。

3.2.1　response 对象的常用方法

response 对象的常用方法有：

• addheader(String name,String value)。

该方法可以添加 HTTP 文件头，并将该 header 传送到客户端，如果同名的 header 存在，那么原来的 header 会被覆盖。调用的语法如下：

```
response.addHeader(String name,String value)
```

其中，name 是头信息的名称，value 是对应的头信息的值。

• setHeader(String name,String value)。

该方法用于设置指定名字的 HTTP 文件头值。调用的语法如下：

```
response.setHeader(String name,String value)
```

其中，name 是头信息的名称，value 是对应的头信息的值。

• containsHeader(String name)。

该方法可以判断指定名字的 HTTP 文件头是否存在。返回值为布尔型变量。调用的语法如下：

```
response.containsHeader(String name)
```

其中，参数 name 是指定的头信息的名称。

• addCookie(Cookie cook)。

该方法用于添加一个 Cookie 对象，用来保存客户端的用户信息，可以通过 request 对象的 getCookie()方法获得这个 Cookie。调用的语法如下：

```
response.addCookie(Cookie cook)
```

其中，参数 cook 是一个 Cookie 对象的名称。

• encodeURL()。

该方法使用 sessionId 来封装 URL，如果没有必要封装 URL，则返回原来的值。调用的语法如下：

```
response.encodeURL()
```

• flushBuffer()。

该方法可以强制将当前缓冲区的内容发送到客户端。调用的语法如下：

```
response.flushBuffer()
```

• getBufferSize()。

该方法用于返回缓冲区的大小。调用的语法如下：

```
response.getBufferSize()
```

• sendError(int sc)。

该方法向客户端发送错误信息。例如，505 为服务器内部错误，404 为网页找不到的错误或者页面无效。调用的语法如下：

```
response.sendError(int sc)
```

• sendRedirect(String location)。

该方法把响应发送到另一个指定的位置进行处理。调用的语法如下：

```
response.sendRedirect(String location)
```

其中，参数 location 可以是合法的 URL 地址，也可以是文件路径。

• getOutputStream()。

该方法用于返回到客户端的输出流对象。调用的语法如下：

```
response.getOutputStream()
```

• setContentType()。

该方法可以设置响应的 MIME 类型。调用的语法如下：

```
response.setContentType()
```

3.2.2 response 对象使用实例

response 对象的 sendRedirect(String location)方法在实际开发中用得比较频繁，因为实际开发中经常遇到页面跳转和重定向的问题。下面的例子程序将演示该方法的应用。

参考程序 3-2：

(1)建立 02-1.jsp 文件，将其保存在“\ch03\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<html>
    <head>
        <title>注意标题地址栏变化 - 跳转页面</title>
    </head>
    <body>
        <center>
            和 jsp:forward 进行对比<br/>
            <%
                out.println("这是跳转页面");
            %>
        </center>
    </body>
</html>
```

(2)建立 02-2.jsp 文件，将其保存在“\ch03\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<html>
```

```
<head>
    <title>注意标题地址栏变化 - 原始页面</title>
</head>
<body>
    <center>
        <h3>response.sendRedirect()方法使用实例</h3>
    </center>
    <form action="02-2.jsp">
        <table border=1>
            <tr><td>
                <select name="selectParam">
                    <option value=0>当前页面</option>
                    <option value=1>跳转页面</option>
                </select>
            </td></tr>
            <tr><td>
                <input type="submit" value="跳转到选择的页面">
            </td></tr>
        </table>
    </form>
    <%
        String q=request.getParameter("selectParam");
        if("1".equals(q))
            response.sendRedirect("02-1.jsp");
        else
            out.println("没有进行页面重定向");
    %>
</body>
</html>
```

(3)在浏览器中输入 http://localhost:8080/jsplearner/ch03/02-1.jsp 并按 Enter 键,可以查看程序的运行效果,如图 3-2 所示。

图 3-2　程序的运行效果 2

程序分析 3-2：

在这里，判断 select 的选择结果，是使用 response. sendRedirect()跳转来进行的，需要注意浏览器地址栏和标题的变化。读者可以自行与前面章节中的 jsp:forward 动作参考程序进行对比。

3.3 session 对象

session 对象是 java. servlet. http. HttpSession 类的对象，用来表示和存储当前页面的请求信息。也就是说，session 对象用来保存每个用户的信息，这样对每个用户的操作状态进行跟踪变得非常容易。session 信息会被保存在容器里，一般情况下，用户首次登录系统时，容器会给此用户分配一个唯一标识的 session id 以区别其他用户。当用户退出系统时，这个 session 将自动消失。

3.3.1 session 对象的常用方法

session 对象的常用方法有：

• getAttribute(String name)。

该方法用于获得指定名字的属性，如果该属性不存在，将返回 null。调用的语法如下：

```
session.getAttribute(String name)
```

其中，参数 name 是指定的 session 对象的名称。

• getAttributeNames()。

该方法用于返回 session 对象中存储的每一个属性对象，结果是一个 Enumeration 类的实例。调用的语法如下：

```
session.getAttributeNames()
```

该方法用于返回一个枚举类型的对象，可以用返回对象的 nextElements()方法来遍历 session 对象中所有的数据对象。

• getCreationTime()。

该方法用于返回 session 对象的创建时间，单位为毫秒。调用的语法如下：

```
session.getCreationTime()
```

• getId()。

每生成一个 session 对象，服务器都会进行编号，而且这个编号是不会重复的。该方法就是返回当前 session 的编号。调用的语法如下：

```
session.getId()
```

• getLastAccessedTime()。

该方法用于返回当前 session 对象的最后一次被操作的时间，单位为毫秒。调用的语法如下：

```
session.getLastAccessedTime()
```

• getMaxInactiveInterval()。

该方法用于获取 session 对象的生存时间。调用的语法如下：

```
session.getMaxInactiveInterval()
```

• removeAttribute(String name)。

该方法用于删除指定属性的属性值和属性名。调用的语法如下：

```
session.removeAttribute(String name)
```

其中，参数 name 是指定的 session 对象的名称。该方法无返回值。

• setAttribute(String name,Java.lang.Object value)。

该方法可以设定指定名字的属性，并把它存储在 session 对象中。调用的语法如下：

```
session.setAttribute(String name,Java.lang.Object value)
```

其中，参数 name 是要设置的 session 对象中的数据对象的名字，value 是这个对象对应的值。如果对象不存在，则新加入一个；如果对象已经存在，则值会被修改。

• invalidate()。

该方法用于注销当前的 session。调用的语法如下：

```
session.invalidate()
```

3.3.2　session 对象使用实例

由于 session 对象一般被用来保存一些需要在与每个用户会话期间保持的数据，例如，输入的用户名和密码等信息，所以下面的例子程序通过实现一个简单的登录系统演示 session 对象的使用，用户在登录页面输入正确的用户名和密码后，页面跳转到欢迎页面，并且读出保存在 session 中的信息。在这里需要建立两个 JSP 页面，分别是登录页面 login_sessionExample.jsp 和欢迎页面 welcome.jsp。

参考程序 3-3：

(1)建立 03-1.jsp 文件，将其保存在“\ch03\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<html>
    <head>
        <title>使用 Session 进行登录</title>
    </head>
    <body>
        <h2>登录首页</h2>
        <%
            String userName=(String)session.getAttribute("Login");
            if(userName!=null)
            {
                out.println("欢迎你,"+userName);
            }
            else
            {
        %>
        <form action="03-2.jsp" method="POST">
            用户名:<input type="text" name="userName"><br>
            密码:<input type="password" name="password"><br>
```

```
            <input type="submit" value="登录"><br>
        </form>
        <%
            }
        %>
    </body>
</html>
```

(2)建立 03-2.jsp 文件,将其保存在“\ch03\”下,并输入如下所示的代码:

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
    String userName=request.getParameter("userName");
    String password=request.getParameter("password");
    if("admin".equals(userName)&&"123".equals(password))
    {
        // 对于一般的应用,数据库密码从数据库读出
        // 注意判断 userName 以及 password 是否为空
        session.setAttribute("Login",userName);
    }
    response.sendRedirect("03-1.jsp");
%>
```

(3)在浏览器中输入 http://localhost:8080/jsplearner/ch03/03-1.jsp 并按 Enter 键,可以查看程序的运行效果,如图 3-3 所示。在页面中按照提示输入“admin”和“123”并提交,页面将转至 03-1.jsp 文件所表示的页面,如图 3-4 所示。

图 3-3 程序的运行效果 3

图 3-4 页面的跳转效果

程序分析 3-3:

在这里,程序使用 03-1.jsp 进行登录,03-2.jsp 进行登录验证。由于登录验证页面(03-2.jsp)不进行显示,所以不需要 HTML 代码。使用 session.setAttribute 设置登录属性(此处是用户名),为 null 即为未登录。

3.4　out 对象

out 对象可以说是 JSP 中被使用最频繁的对象，它被封装成 javax.servlet.JspWriter 接口，表示为客户打开的输出流，被 PrintWriter 使用，向客户端发送输出流。简单地说，out 就是代表输出流的对象，主要用来向客户端输出数据。

3.4.1　out 对象的常用方法

out 对象的常用方法有：

• print()。

该方法用于在页面上打印出字符串信息，不换行。调用的语法如下：

```
out.print()
```

• println()。

该方法用于在页面上打印出字符串信息，并且换行。调用的语法如下：

```
out.println()
```

• newLine()。

该方法用于在页面上输出一个换行字符。调用的语法如下：

```
out.newLine()
```

• clear()。

该方法用于清除缓冲区中尚存的内容。调用的语法如下：

```
out.clear()
```

• clearBuffer()。

该方法用于清除当前缓冲区中尚存的内容。调用的语法如下：

```
out.clearBuffer()
```

• flush()。

该方法用于清除数据流。调用的语法如下：

```
out.flush()
```

• getBufferSize()。

该方法用于获得缓冲区的大小。调用的语法如下：

```
out.getBufferSize()
```

• getRemaining()。

该方法用于获得缓冲区剩余空间的大小。调用的语法如下：

```
out.getRemaining()
```

• IsAutoFlush()。

该方法用于检查当前缓冲区是设置为自动清空(返回 true)，还是满了就抛出异常(返回 false)。调用的语法如下：

```
out.IsAutoFlush()
```

• close()。

该方法用于关闭输出流。调用的语法如下：

```
out.close()
```

在以上方法中，print()和 println()方法是最常用的。print()方法就是把 Java 对象原始数据类型输出到客户端的缓冲区，而 println()方法除了把内容输出到客户端外，还在后面添加一个空行。但是这个空行在被浏览器解析时会被忽略，所以要想真正在页面内换行，需要通过 out.println("
")方法来实现。

3.4.2 out 对象使用实例

out 对象的使用相对简单。下面的例子程序演示了 out 对象的常用方法的使用。

参考程序 3-4：

(1)建立 04.jsp 文件，将其保存在“\ch03\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<html>
    <head>
        <title>out 对象中常用方法的使用实例</title>
    </head>
    <body>
        <%
            response.setContentType("text/html");
            out.println("out 对象的方法使用实例:<br><hr>");
            out.println("<br>out.println(boolean):");
            out.println(false);
            out.println("<br>out.println(char):");
            out.println('x');
            out.println("<br>out.println(char[]):");
            out.println(new char[]{'x','y'});
            out.println("<br>out.println(double):");
            out.println(213.63d);
            out.println("<br>out.println(float):");
            out.println(12351.7326f);
            out.println("<br>out.println(int):");
            out.println(323);
            out.println("<br>out.println(long):");
            out.println(112365128472461526 3L);
            out.println("<br>out.println(object):");
            out.println(new java.util.Date());
            out.println("<br>out.println(string):");
            out.println("string");
            out.println("<br>out.newLine():");
```

```
            out.newLine();
            out.println("<br>out.getBufferSize():");
            out.println(out.getBufferSize());
            out.println("<br>out.getRemaining():");
            out.println(out.getRemaining());
            out.println("<br>out.isAutoFlush():");
            out.println(out.isAutoFlush());
        %>
    </body>
</html>
```

(2)在浏览器中输入 http://localhost:8080/jsplearner/ch03/04.jsp 并按 Enter 键,可以查看程序的运行效果,如图 3-5 所示。

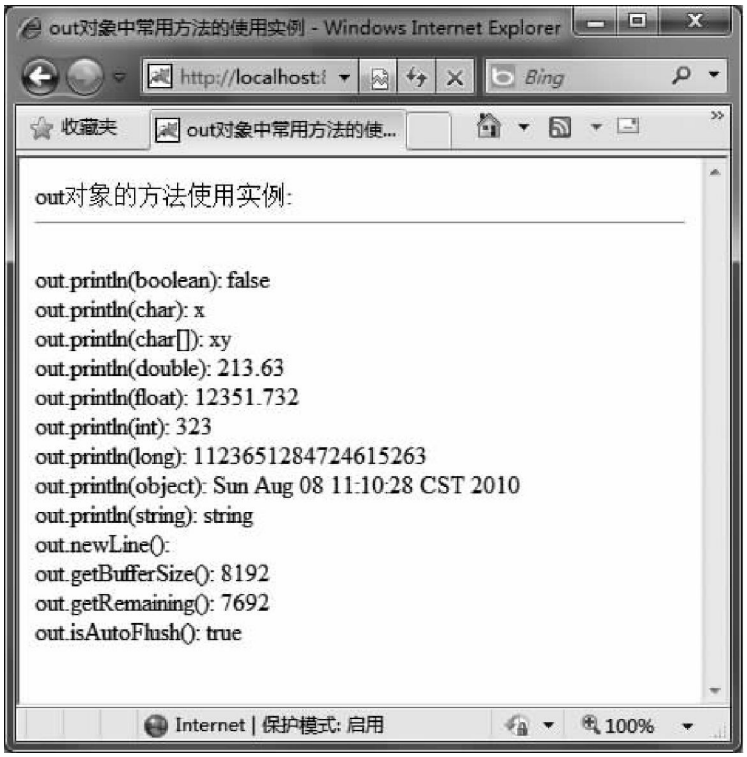

图 3-5　程序的运行效果 4

程序分析 3-4:

在这里,使用 out.println()方法输出了 out 对象的多种属性。由于 out 对象的使用方法非常简单,所以在这里不作更多介绍。

3.5　application 对象

application 对象是 javax.servlet.ServletContext 类对象的一个实例,它用于多个程序或者多个用户之间共享数据。对于一个容器而言,每个用户都共用一个 application 对象,服务器一旦启动,就会自动地创建 application 对象,这个对象会一直存在,直到服务器关闭为止。

application 对象的作用和前面介绍的 session 对象有一些类似，但是它们之间的区别是很明显的。一般来说，一个用户对应一个 session，并且随着用户的离开，session 中的信息也会消失，所以不同用户之间想要会话，就必须保持某一时刻至少有一个用户没有终止会话；而 application 对象则不同，它只有一个实例，且一直存在，直到服务器关闭，类似于系统中的“全局变量”。

3.5.1 application 对象的常用方法

application 对象的常用方法有：

• getAttribute(String name)。

该方法用于返回一个由 name 指定名字的 application 对象属性的值。这个返回值是一个 Object 对象。调用的语法如下：

```
application.getAttribute(String name)
```

其中，参数 name 是指定的 application 对象中的数据对象的名称。

• getAttributeNames()。

该方法用于返回所有 application 对象属性的名字，返回值是一个 Enumeration 类型的实例。调用的语法如下：

```
application.getAttributeNames()
```

该方法返回一个 Enumeration 类型的实例，可以用返回实例的 nextElements()方法来遍历 application 对象中所有的数据对象。

• getInitParameter(String name)。

该方法用于返回 application 某个属性的初始值。调用的语法如下：

```
application.getInitParameter(String name)
```

其中，参数 name 指定 application 对象中的数据对象的名称。

• getServerInfo()。

该方法用于获得当前版本 Servlet 编译器的信息。调用的语法如下：

```
application.getServerInfo()
```

• setAttribute(String name,Object object)。

该方法可以用 object 来初始化某个由 name 指定的属性。调用的语法如下：

```
application.setAttribute(String name,Object object)
```

• removeAttribute(String name)。

该方法用于删除一个指定的属性。调用的语法如下：

```
application.removeAttribute(String name)
```

3.5.2 application 对象使用实例

由于 application 对象是一直存在于服务器端的，所以经常用来实现用户之间的数据共享。下面的例子程序演示了如何利用 application 对象实现对网页访问次数进行计数。

参考程序 3-5：

(1)建立 05.jsp 文件，将其保存在“\ch03\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
<html>
    <head>
        <title>使用 Application 进行网页访问统计</title>
    </head>
    <body>
        <%
            Object userCountObj=application.getAttribute("userCount");
            int userCount=0;
            if (userCountObj==null)
                userCount=1;
            else
                userCount=Integer.parseInt(userCountObj.toString())+1;
            application.setAttribute("userCount",userCount);
        %>
        <h2>
            您是第<%=userCount%>位访问者
        </h2>
        <h3>注意刷新页面的效果、重启服务器的效果</h3>
    </body>
</html>
```

(2)在浏览器中输入 http://localhost:8080/jsplearner/ch03/05.jsp 并按 Enter 键，可以查看程序的运行效果，如图 3-6 所示。

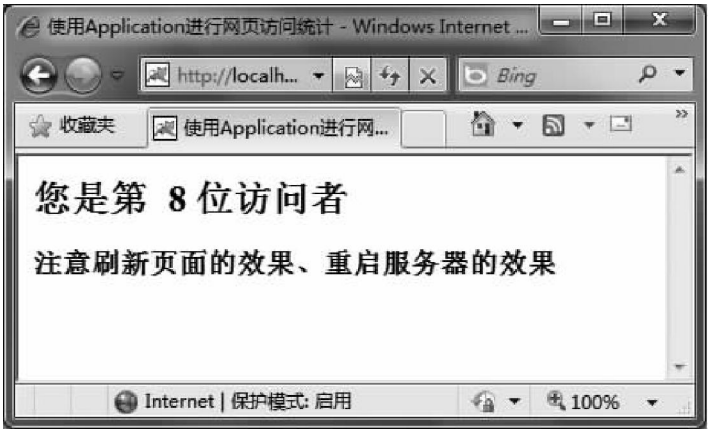

图 3-6　程序的运行效果 5

程序分析 3-5：

在这里，程序调用了 application 对象的 setAttribute()方法来存储用户名信息以及对初始值进行计数。由于 application.getAttribute("userCount")返回的是一个 Object 对象，值为 null 的 Object 进行 toString 转换会导致 NullPointerException(读者可以自己尝试)。

如果刷新这个页面，userCount 值将加 1。随着网页被不断刷新或者访问，计数器也会不断增加。而一旦服务器重启，计数值将“归 0”，调用 getAttribute 会返回 null。因此虽然 application 对象能够持久保持，但是如果需要一直进行计数，就应该将这些数据保存在数据库中。

3.6 page 对象

page 对象是可以从 JSP 脚本小程序和表达式中获得的一个内置对象，该对象是 java.lang.Object 类的一个实例。换句话说，page 对象是指当前 JSP 页面本身，有些类似于 Java 中的 this 指针。需要说明的是，相对于其他内置对象，page 对象在实际开发过程中应用得并不多。

3.6.1 page 对象的常用方法

page 对象的常用方法有：

• getClass()。

该方法用于返回当前的 Object 的类。调用的语法如下：

```
page.getClass()
```

• hashCode()。

该方法用于返回当前的 Object 类的哈希代码。调用的语法如下：

```
page.hashCode()
```

• toString()。

该方法用于把当前的 Object 类转换成字符串。调用的语法如下：

```
page.toString()
```

• equals(Object obj)。

该方法用于比较当前对象是否与指定的对象 obj 相等。调用的语法如下：

```
page.equals(Object obj)
```

• copy(Object obj)。

该方法用于把当前对象复制到指定的对象 obj 中。调用的语法如下：

```
page.copy(Object obj)
```

• clone()。

该方法用于把当前对象类进行克隆。调用的语法如下：

```
page.clone()
```

3.6.2 page 对象使用实例

下面的例子程序演示了如何使用 page 对象的常用方法。

参考程序 3-6：

(1)建立 06.jsp 文件，将其保存在“\ch03\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"%>
```

```
<html>
    <head>
        <title>page 对象的方法使用实例</title>
    </head>
    <body>
        <h3>
            <%! Object obj; %>
            objValue:<% =obj %>
            <br>getClass:<% =page.getClass() %>
            <br>hashCode:<% =page.hashCode() %>
            <br>toString:<% =page.toString() %>
            <br>equals:<% =page.equals(obj) %>
        </h3>
    </body>
</html>
```

(2)在浏览器中输入 http://localhost:8080/jsplearner/ch03/06.jsp 并按 Enter 键,可以查看程序的运行效果,如图 3-7 所示。

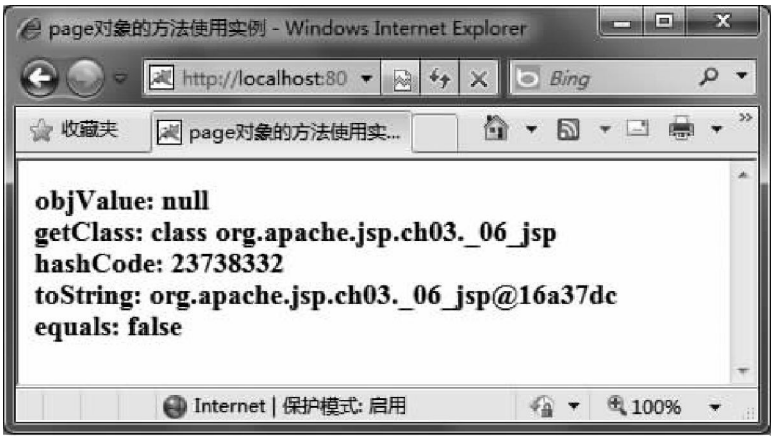

图 3-7　程序的运行效果 6

程序分析 3-6:

在这里,程序实现的是简单的 page 对象使用实例。需要注意的是,对 obj 声明不赋初值时,其默认初值为 null。

3.7　config 对象

config 对象提供了对每一个给定的服务器小程序或 JSP 页面的 javax.servlet.ServletConfig 对象的访问,封装了初始化参数以及一些实用方法。也就是说,config 对象表示了 Servlet 的配置,当一个 Servlet 被初始化时,容器会将某些信息通过 config 对象传递给 Servlet。

一般在 JSP 的实际开发中很少用到 config 对象,只有在编写 Servlet 且需要重载 Servlet

的 init()方法时才会用到，所以这里就只对 config 对象的几种常用方法进行介绍。

config 对象的常用方法有：

• getInitParameter(String name)。

该方法用于获得初始化参数。返回值为 String 类型。调用的语法如下：

```
config.getInitParameter(String name)
```

• getInitParameterNames()。

该方法用于获得所有初始化参数的名称。返回值为一个 Enumeration 类型。调用的语法如下：

```
config.getInitParameterNames()
```

• getServletName()。

该方法用于获得当前服务器小程序或 JSP 页面的名称。调用的语法如下：

```
config.getServletName()
```

• getServletContext()。

该方法用于获得当前服务器小程序或者 JSP 页面的服务器小程序环境。调用的语法如下：

```
config.getServletContext()
```

3.8 exception 对象

exception 对象是 java.lang.Throwable 类的一个实例，被用来处理页面中出现的异常错误。这种异常指的是运行时的异常，也就是被调用的错误页面的结果。一般来说，exception 对象可以配合 page 指令一起使用，通过指定某个页面为错误处理页面，把所有的错误都集中在那个页面处理，可以使得整个系统更加强壮，程序流程也更加清晰。

异常错误一般都是开发人员无法避免的，所以在编写 JSP 程序时，应该考虑到各种可能的异常，并在程序中对这些异常进行后期处理和必要的提示。

3.8.1 exception 对象的常用方法

exception 对象的常用方法有：

• getMessage()。

该方法用于返回 exception 对象的异常消息字符串。调用的语法如下：

```
exception.getMessage()
```

• getLocalizedMessage()。

该方法用于返回 exception 对象的本地化语言的异常错误。调用的语法如下：

```
exception.getLocalizedMessage()
```

• printStackTrace()。

该方法用于以标准错误的形式输出一个异常和异常的堆栈，即显示异常的栈跟踪轨迹。调用的语法如下：

```
exception.printStackTrace()
```

• toString()。

该方法用于以字符串的形式返回一个对异常的描述。调用的语法如下：

```
exception.toString()
```

3.8.2　exception 对象使用实例

下面的例子程序演示了如何在程序中使用 exception 对象。

参考程序 3-7：

(1)建立 07-1.jsp 文件，将其保存在“\ch03\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"
    errorPage="07-2.jsp" %>
<html>
    <head>
        <title>exception 对象的方法使用实例</title>
    </head>
    <body>
        <% Integer.parseInt("t"); /*error 这里会抛出异常 */ %>
    </body>
</html>
```

(2)建立 07-2.jsp 文件，将其保存在“\ch03\”下，并输入如下所示的代码：

```
<%@ page language="java" import="java.util.*" pageEncoding="utf-8"
    isErrorPage="true" %>
<html>
    <head>
        <title>exception 显示页面</title>
    </head>
    <body>
        <p>exception 信息显示，注意地址栏和标题栏变化</p>
        Exception:<%=exception %><br>
        Message:<%=exception.getMessage() %><br>
        Localized Message:<%=exception.getLocalizedMessage() %><br>
        Stack Trace:<% exception.printStackTrace(new java.io.PrintWriter(out));
                    %><br>
    </body>
</html>
```

(3)在浏览器中输入 http://localhost:8080/jsplearner/ch03/07-1.jsp 并按 Enter 键，可以查看程序的运行效果，如图 3-8 所示。

程序分析 3-7：

在这里，程序使用 errorPage 来为异常指定处理页面，在异常处理页面要设置 isErrorPage="true"显示说明“这是一个异常处理的页面”。

图 3-8　程序的运行效果 7

3.9　pageContext 对象

pageContext 对象是一个非常特殊的对象，它引用的是 javax.servlet.jsp.PageContext 对象。它提供了对 JSP 页面内所有的对象及名字空间，可以访问到本页面所在的 Session，也可以取本页面所在的 application 的某一属性值，它相当于页面中所有功能的集大成者。

JSP 容器使用 pageContext 对象的相关方法来初始化其他内置对象。内置对象都自动地被加入至 pageContext 中，程序员可以通过该对象来取得与 JSP 相关的内置对象对应的 Servlet 对象。这与 getRequest()可以取得 ServletRequest、getServletConfig()可以取得 ServletConfig、getSession()可以取得 HttpSession 等一样。

但实际上，pageContext 对象在实际的 JSP 开发过程中很少被使用，因为 request、response 等对象可以直接调用，如果使用 pageContext 对象来调用这些对象，反倒显得舍近求远。因此，在这里只对 pageContext 对象的常用方法进行简单的介绍。

pageContext 对象的主要方法有：

• forward(String relativeUrlPath)。

该方法把页面重定向到另外一个页面或者 Servlet 组件上。调用的语法如下：

```
pageContext.forward(String relativeUrlPath)
```

其中，参数 relativeUrlPath 为一个合法的 URL 地址。

• getAttribute(String name[,int scope])。

该方法可以检测一个特定的已经命名的对象的范围，并且还可以通过调用 getAttributeNamesInScope()方法，检测对某个特定范围的每个属性 String 字符串名称的枚举，scope 参数为可选参数。调用的语法如下：

```
pageContext.getAttribute(String name[,int scope])
```

• getException()。

该方法返回当前的 exception 对象。调用的语法如下：

```
pageContext.getException()
```

• getRequest()。

该方法返回当前的 request 对象。调用的语法如下：

```
pageContext.getRequest()
```

• getResponse()。

该方法返回当前的 response 对象。调用的语法如下：

```
pageContext.getResponse()
```

• getServletConfig()。

该方法返回当前页面的 ServletConfig 对象。调用的语法如下：

```
pageContext.getServletConfig()
```

• getServletContext()。

该方法返回当前页面的 ServletContext 对象。调用的语法如下：

```
pageContext.getServletContext()
```

• getSession()。

该方法返回当前页面的 session 对象。调用的语法如下：

```
pageContext.getSession()
```

• findAttribute()。

该方法可以按照页面、请求、会话以及应用程序范围的顺序实现对某个已命名属性的搜索。调用的语法如下：

```
pageContext.findAttribute()
```

• setAttribute()。

该方法用来设置默认页面范围内或特定对象范围中的已命名的对象。调用的语法如下：

```
pageContext.setAttribute()
```

习　题　3

1. session 对象和 application 对象的异同点是什么？

2. 试分析每个内置对象的作用域。

3. 改进本章例程中的计数器程序，实现对不同的 JSP 页面进行访问计数。

4. 使用 request 对象的方法实现一个简单的登录系统，在 login.jsp 页面中由用户输入用户名和密码，用户名和密码都输入正确后转到 success.jsp，否则转到 login.jsp 页面由用户重新输入。超过 3 次不能登录则打开 error.jsp 提示错误。

5. 设计并编写能够在线打分的 JSP 页面。