第 7 章　JSP基本语法

前面几章依次介绍了 JSP 的基本概念、运行原理。以及如何搭建 JSP 运行环境，并以一个简单的实例作示范来加深对 JSP 运行机制以及架构 JSP 环境的理解。另外，还介绍了 JSP 页面程序一般所包含的其他技术，例如使用到的 HTML 标签语言（基本 JSP 即由 HTML 中嵌入 Java 脚本语言组成）、JavaScript 浏览器脚本语言（由浏览器解析执行）以及层叠样式表（CSS）。

这一章节将向读者正式介绍 JSP 的基本语言，教会读者如何正确地编写 JSP 页面程序。

本章要点包括以下内容：

- JSP 页面的基本组成
- JSP 的脚本元素
- JSP 的三种注释形式
- JSP 指令元素
- JSP 的标准动作元素

7.1　JSP页面构成

前面已经有所介绍，JSP 基本是通过在 HTML 标签语言中嵌入 Java 脚本语言来实现页面动态请求。其中有关 HTML 标签的知识已经在第 4 章简单介绍了，另外 Java 脚本程序是通过<%和%>标记来嵌入到 HTML 中的。基本格式如下：

```
<html>
<head><title>JSP 页面标题</title></head>
<body>
…//HTML 标签语言
<%
  //嵌入 Java 脚本执行语言
%>
…//HTML 标签语言
</body>
</html>
```

其中可以通过<%!...Java 声明…%>来声明变量或者方法；使用<%= Java 表达式 %>格式来引用一个表达式值；更多的是通过在<% Java 脚本语言 %>中嵌入 Java 脚本语言。详细见下面介绍的脚本元素。

JavaScript 浏览器语言和 CSS 以前是对页面程序的补充，但是随着 Web2.0 的提出，这两个技术的重要性已经越来越凸现，慢慢从幕后走到了前台。CSS 可以对页面样式进行更细微的设置；JavaScript 在客户端执行，可以进行简单的输入验证，减少服务器请求次数。但是 JavaScript 在 Web2.0 中的应用，可以实现异步处理，这方面知识见第 21 章的 AJAX 介绍。

7.2　脚本元素

JSP 脚本元素是用来嵌入 Java 代码的，主要用来实现页面的动态请求。一般客户端请求 JSP 请求后，HTML 和嵌入的 Java 脚本元素会被服务端容器编译成 Servlet 文件，然后再动态执行。现在主要有三种脚本元素类型：

- 表达式格式（expression）：用来在页面中直接调用 Java 表达式，从而得到返回值。
- 小脚本格式（scriptlet）：在 HTML 中使用<%和%>来嵌入 Java 程序，从而进行相应逻辑处理。
- 声明格式（declaration）：用来定义 Java 脚本语言中使用到的变量或者方法。

接下来对这三种类型进行详细讲解。

7.2.1　JSP表达式格式

当需要在页面中获取一个 Java 变量或者表达式值时，使用表达式格式是非常方便的。使用的基本语法如下：

```
<%= Java 表达式/变量 %>
```

其中可以引用 Java 表达式值，也可以直接引用某一变量值。当 JSP 容器遇到该表达式格式时，会先计算嵌入的表达式或者变量，然后将计算的结果以字符串形式返回，并插入到相应页面中。

下面通过一个实例作示范，详细代码如下：

```
<html>
<body>
<%
//以下为 Java 脚本语言
Int I = 5;
Int res = 0;
res = i*I+i*2;
%>
<table><tr><td>Current time: <%= res %></td></tr></table>
</body>
</html>
```

代码说明：该实例中就是使用<%= res %>来直接调用 res 变量的值，使用非常方便。

最后，如果需要在 XML 引用 JSP 表达式时，也可以写成下面这种格式（这里不重点介绍）：

```
<jsp:expression>…Java 表达式…</jsp:expression>
```

注意：XML 和 HTML 不一样。XML 是大小写敏感的。想详细了解 XML 技术，请读者参考相关书籍。

7.2.2　JSP小脚本格式

小脚本更加的灵活，它可以包含任意的 Java 片段，从而可以执行更加复杂的操作和控制，这些都是 JSP 表达式格式所不能达到的。小脚本的编写方法即将 Java 程序片段插入到<% %>标记中，基本语法如下：

```
<%...任意的 Java 代码…%>
```

另外小脚本（scriptlet）的 XML 兼容语法为：

```
<jsp:scriptlet>…Java 代码…</jsp:scriptlet>
```

下面通过一个实例来具体介绍该方法的使用，详细代码如下：

```
<%@ page import=”java.util.Date” %>
<html>
<body>
<%
//以下为 Java 小脚本语言
Date now_date = new Date();
out.println(“当前日期为：”+now_date);
%>
</body>
</html>
```

代码说明：以上代码中使用到了下面将要讲到的 page 指令，通过属性 import 来应用 Date 类。在脚本中初始化了 Date 类的一个实例对象 now_date，然后通过 out 内置对象（下一章将对 JSP 中的内置对象作详细介绍）中的 println()方法（注意和 print()方法的区别）来打印出当前的日期。

7.2.3　JSP声明格式

在编写 JSP 页面程序时，有时需要为 Java 脚本定义变量和方法，可以通过下面的方法进行声明：

```
<%!...Java 声明…%>
```

注意和以上两种脚本语法的不同，另外，声明中一般不会有任何输出，它一般是和脚本表达式、小脚本一起配套使用。下面还是通过一个实例来说明：

```
<html>
<%!
    String ID=”Johnson”;                    //声明的变量
    String ReturnID()                       //声明的方法
    {    return ID; }
%>
<body>
The UserID: <%=ID%> <br>
<%
    String UserID= ReturnID();                    //调用上面声明的函数方法
    Out.println(“The UserID: “+UserID);
%>
</body>
</html>
```

代码说明：上面的代码申明了一个变量 ID 和 ReturnID()方法，然后在表达式格式和小脚本中进行调用，并将结果打印出来。如果读者对 Java 有所了解的，将对该申明不会感到陌生，在前面已经介绍过，其实 JSP 最终会被编译成 Servlet 的 Java 程序。

上面的 JSP 程序编译成的 Servlet 代码如下：

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
public final class test_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {
  String ID="Johnson";
  String ReturnID()
```

```
    {     return ID; }
  private static java.util.Vector _jspx_dependants;
  public java.util.List getDependants() {
    return _jspx_dependants;
  }
  public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {
    JspFactory _jspxFactory = null;
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
    try {
      _jspxFactory = JspFactory.getDefaultFactory();
      response.setContentType("text/html");
      pageContext = _jspxFactory.getPageContext(this, request, response,null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;
      out.write("<html>\r\n");
      out.write("\r\n");
      out.write("<body>\r\n");
      out.write("The UserID: ");
      out.print(ID);
      out.write(" <br>\r\n");
    String UserID= ReturnID();
    out.println("The UserID: "+UserID);
      out.write("\r\n");
      out.write("</body>\r\n");
      out.write("</html>\r\n");
    } catch (Throwable t) {
      if (!(t instanceof SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
          out.clearBuffer();
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
      }
    } finally {
      if (_jspxFactory != null) _jspxFactory.releasePageContext(_jspx_page_context);
    }
  }
}
```

代码说明：其实用户请求的 JSP 页面，最终都是执行该编译好的 Servlet 程序。读者对比 JSP 程序

代码和生成的 Servlet 代码，就会看出 JSP 带来的好处，编写一个 Servlet 是这样的复杂，即使只是为了完成一个简单的功能。

Servlet 已经慢慢退缩到幕后，在后面将会介绍 Servlet 作为控制器、过滤器以及监听器的使用。而页面程序的编写交给了 JSP。

如果开发者没有特别指定该编译文件 Servlet 的目录，一般会由 Tomcat 容器自动生成在 Tomcat 5.0\work\Catalina\localhost 目录（以本书安装的 Tomcat5.0 容器为例）。

7.3　JSP的三种注释

在编写程序的时候，每个程序员都要养成做注释的好习惯，这也是项目的一个规范。因为随着时间，项目代码越来越多，当再返回来看原来的代码时，即使自己编写的，也会很费力。合理、详细的注释有利于代码后期维护和阅读。一般在 JSP 文件编写过程中，有三种注释方法，详细介绍如下。

7,3.1　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，所以使用在 HTML 中注释方法当然也可以在 JSP 文件中使用。其使用格式如下：

```
<!--…add your comments here…-->
```

说明：使用该注释方法，其中的注释内容在客户端浏览中是看不见的。但是查看源代码时，客户是可以看到这些注释内容，所以说这种注释方法是不安全的，即不能带有保密内容。使用该注释方法，只是为了加强程序的可读性，但是对响应速度要求非常高度的站点，并不建议使用太多该类注释，因为会加大网络传输负担。

7.3.2　JSP注释标记

JSP 也提供了自己的标记来进行注释，其使用的格式一般如下：

```
<%--    add your comments here    --%>
```

说明：使用该注释方法的内容在客户端源代码中是不可见的，所以安全性比较高。

7.3.3　JSP脚本中使用注释

前面已经介绍，脚本就是嵌入到<%和%>标记之间的程序代码，并且脚本使用的语言是为 Java，所以在脚本中进行的注释和 Java 类中的注释方法一样。可以使用“//”来注释一行，使用“/*和*/”来注释多行内容。具体的使用格式如下：

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

说明：了解 Java 的读者应该对该方法很熟悉，下面小节通过一个综合实例来演示这三种注释方法的具体使用。

7.3.4　注释的综合实例

创建的代码如下：

```
<html>
<body>
<!--  这是一个 HTML 注释(在客户端源代码可以看到)  -->
<%--  这是一个 JSP 标记注释(在客户端源代码不可以看到)  --%>
<%
        /*这里是 JSP 脚本部分，
        使用 Java 语言，实现字符串输出*/
        out.println("这是一个注释实例");                    //打印出字符串
%>
</body>
</html>
```

代码说明：该实例综合使用了以上介绍的所有注释方法，读者可以在浏览器端查看源代码，检查哪些内容是不见的，即是安全的。

读者在以后的代码编写中，要逐步学会加写注释的好习惯。及时对那些需要响应速度非常快的页面，在编写的过程也要适当作注释，当网站发布时，再进行适当去除。

7.4　JSP指令元素

指令元素不是用来进行逻辑处理或者产生输出代码，而是通过指令中的属性配置来向 JSP 容器发出一些指示，用来控制 JSP 页面的某些特性。使用 JSP 指令元素的格式一般如下：

```
<%@ 指令名[…一个或者多个指令属性…] %>
```

其中 JSP 共有三个指令元素，如下：

- page 指令：用于对 JSP 文件中的全局属性进行设置。
- taglib 指令：该指令是用来声明用户自定义新的标签。
- include 指令：在 JSP 页面中使用该指令来引用外部文件。

接下来逐一讲解这三个指令的具体使用方法。

7.4.1　全局指令page

该指令在介绍 JSP 脚本时，已经有所提及，即使用了 page 指令中的 import 属性来引用一个类文件。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，这里需要注意的是 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。如果某些属性没有进行设置，JSP 容器将使用默认指令属性值。

下面列出 page 指令中的各类属性的设置格式：

```
<%@ page
[ language="java" ]
[ extends="package.class" ]
```

```
[ import="{package.class | package.*}, ..." ]
[ session="true | false" ]
[ buffer="none | 8kb | sizekb" ]
[ autoFlush="true | false" ]
[ isThreadSafe="true | false" ]
[ info="text" ]
[isELIgnored="true|false" ]
[ isErrorPage="true | false" ]
[ errorPage="relativeURL" ]
[ contentType="mimeType [ ;charset=characterSet ]" | "text/html ; charset=ISO-8859-1" ]
%>
```

代码说明：设置多个属性时，是以分号相互隔开的。其中各个属性的含义介绍见表 7.1 所示。

表 7.1　page指令属性

属　性	描　述	默 认 值
language	指定JSP页面使用的脚本语言	Java
extends	JSP被编译成Servlet的Java程序，该属性设置Servlet继承的超类	HttpJspBase类
import	通过该属性来引用脚本语言中使用到的类文件	无
session	设置session用来共享信息的有效性	true（有效）
buffer	定义输出流缓冲区大小	默认8KB
autoFlush	设置输出流的缓冲区是否自动清除	true（自动）
isThreadSafe	设置该JSP页面是否能够同时处理超过一个以上的页面请求	true（可以）
info	用来指定JSP页面的信息	无
isELIgnored	设置该JSP页面是否支持表达式语言（EL，在高级部分介绍）	false（支持）
isErrorPage	设置该页面是否作为其他页面的错误处理	false（不作为）
errorPage	和isErrorPage属性配套使用，设置异常处理的JSP页面URL	无
contentType	用来指定MIME类型以及JSP页面所采用的编码方式	text/html,ISO8859-1

各属性详细讲解如下：

7.4.1.1　language 属性

page 指令中的 language 属性是用来指定当前 JSP 页面所采用的脚本语言，当前 JSP 版本只能采用 Java 作为脚本语言。其实该属性可以不设置，因为 JSP 默认的就是采用 Java 作为脚本。language 属性的设置方法如下：

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

指令语言是放置在<%@和%>标签之间的，并使用字段“page”来指定设置的是 page 指令属性。其中“java”即为 language 属性设置的值，即指定该 JSP 页面所采用的脚本语言为 Java。

7.4.1.2　extends 属性

在前面几章已经多次提及，JSP 其实是一个特殊的 Servlet，它最终需要被编译成 Servlet 的 Java 类程序，然后执行用户的请求。被编译成的 Java 类程序一般继承一个父类，而默认的继承类为 HttpJspBase 类。其实开发者可以通过此处的 extends 属性来设置继承的超类。通过下面的例子来说明 extends 属性的具体设置方法：

```
<%@ page language="java" extends="com.model. sqlUser" %>
```

通过上面的介绍，可知这里使用了 language 属性设置了 Java 脚本语言。通过 extends 属性设置该 JSP 编译后的 Servlet 类程序将继承 com.model 包下的 sqlUser 超类。如果打开 Tomcat 容器所默认的输出目录 Tomcat 5.0\work\Catalina\localhost 下的 Servlet 文件，将发现代码形式如下：

```
…
public final class index_jsp extends com. model. sqlUser
    implements org.apache.jasper.runtime.JspSourceDependent {…}
```

```
...
```

代码说明：此处 com. model. sqlUser 类即为编译后的 Servlet 类文件的继承超类。通过 extends 属性设置的超类，即可以在 JSP 脚本片断中使用该超类中的所有保护成员和公共成员（这里的成员包括属性变量和方法）

注意：尽量少使用 extend 属性来指定超类，这样就会把 JSP 代码与 Java 代码进行了绑定。

7.4.1.3　import 属性

page 指令中的 import 属性将在后面的介绍中经常提及，其实在实际开发过程中也是使用非常频繁的。通过 import 属性可以在该 JSP 文件的脚本片断中引用外在的类文件。进行选择的类文件有如下多种：

- 系统环境变量中所指定目录下的类文件；
- Tomcat 容器默认 Tomcat 5.0\common\lib 目录下的零散类文件或者打包后的 Jar 文件；
- 建立的 Web 模块所在的所在 WEB-INF\classes 目录下的类文件以及 WEB-INF\lib 目录下的 jar 文件。

注意：使用 import 属性设定的类文件一定要写全名（即必须加上包名）。如果一个 import 属性引入多个类文件时，需要在多个类文件之间用逗号隔开。

该属性的具体设置格式如下：

```
<%@ page import=”cn.com.zzb.eshopping.model.*,cn.com.zzb.eshopping.model.sql.*” %>
```

代码说明：通过 import 属性将包 cn.com.zzb.eshopping.model 和 cn.com.zzb.eshopping.model.sql 下的所用类文件引入。

上面引用格式也可以分成两步写：

```
<%@ page import=”cn.com.zzb.eshopping.model.*” %>
<%@ page import=”cn.com.zzb.eshopping model.sql.*” %>
```

通过 import 属性设置的类文件，在 JSP 程序被编译成 Servlet 文件之后，会变成标准的 Java 程序形式，如下：

```
import=”cn.com.zzb.eshopping.model.*”;
import=”cn.com.zzb.eshopping model.sql.*”;
…
```

注意：import 属性在后面实例中会被经常使用，读者一定要对此属性非常了解。

7.4.1.4　session 属性

这里介绍的 session 属性要和前面已经介绍的多个属性有所差异。前面介绍的属性是在 JSP 页面处于编译阶段执行的，而 session 属性是在容器处于请求阶段时所提供的指示。

其实大部分网站都使用到 session，这里提及的 session 是内置对象（将在下面重点介绍），而 session 属性是用来设置 JSP 脚本语言中 session 内置对象是否有效：“true”值设置有效；“false”表示无效。当 Web 站点需要在多个页面之间进行交互作用时，常常需要共享一些信息，这时就需要使用到 JSP 中的 session 内置对象。例如使用 session 来保存用户登录信息，电子购物网站使用 session 来保存购物车中的商品信息。

为了使 JSP 中的 session 内置对象可用，需要设置 session 属性值为“true”，其基本格式如下：

```
<%@ page session=”true” %>
```

session 属性的默认值为“true”，即 session 内置对象可用。当设置 session 内置对象可用之后，容器就会创建和维护一个 HTTP 会话，使用 session 内置对象来存储信息。当 session 属性值设置为“false”时，容器则不会创建任何的 HTTP 会话，如果强迫使用 session 存储和提取信息，容器会报相应错误。

为了直观地介绍 session 属性以及引入 session 内置对象，下面通过一个实例来说明问题。因为 session

属性默认的为“ture”值，所以就没有必要再进行 session 属性的设置。首先创建的 JSP 文件为 submitName.jsp，详细代码：

```
<html>
<body>
<div align='center'>
<form method=post action="SaveName.jsp">
    您的姓名：<input type=text name=username size=20>
    <p><input type=submit value="提交"> <input type=reset value="重置">
</form>
</div>
</body>
</html>
```

代码说明：该 JSP 页面只负责提交用户所输入的用户名信息，然后由 savename.jsp 页面进行相应处理。

创建的 saveName.jsp 页面的详细代码如下：

```
<%
    String name = request.getParameter("username");          //获取上一页面 GetName.jsp 中输入的用户名
    session.sessAttribute("User",name);                      //把用户名信息存储在 session 中
    String sessionUserName = (String)session.getAttribute("user");      //从 session 中获取到用户信息
    Out.println("hello!"+sessionUserName);                   //打印出用户信息
%>
```

代码说明：该页面对传递过来的用户名信息进行了相应处理，使用 session 对象中的 sessAttribute 方法将用户名存入到了 session 中。然后从 session 获取到用户名信息，并打印处理。当然除了本页面可以获取 session 中的用户名信息之后，其他同一站点的 JSP 页面程序都可以共享到该用户名信息。获取 session 信息的方法为 session.getAttribute(ID)。

7.4.1.5　buffer 属性

buffer 属性在实际开发过程中，并不经常使用，它是用来设置输出缓冲区（读者应该对缓冲区的概念有所认识，计算机中普遍使用到的技术）的大小，默认值缓冲区大小为 8KB（一般默认使用）。当遇到特殊情况时，才将它设置成 8KB 以上或者 none（表示不使用输出缓冲区）。

7.4.1.6　autoFlush 属性

该属性一般需要和之上的属性陪同使用，它用来设置输出缓冲区使用可以自动刷新清除（将缓冲区中的内容刷新到页面中显示），默认值为“true”，即自动清除处理。如果属性值设置为“false”，即不自动清除，这时需要开发者在程序中编写程序进行手动清除。

7.4.1.7　isThreadSafe 属性

该属性同样在实际开发过程中不常使用，它使用来设置该 JSP 页面是否能够同时处理超过一个以上的用户请求。

7.4.1.8　info 属性

Info 属性使用非常简单，它其实不对 JSP 页面特性进行设置，它只是定义了一个字符串，并且可以使用 servlet.getServletInfo()获得 info 所定义的信息。

```
<%@ page info="这是第一个 info 属性设置的例子" %>
…
<%
    out.println(getServletInfo());                    //打印出 info 属性所定义的字符串
```

```
%>
…
```

代码说明：该页面使用 page 指令中的 info 属性定义了一个“这是第一个 info 属性设置的例子”字符串，然后调用 Servlet 中的 getServletInfo()方法来获取到该字符串的信息。运行该页面，效果如图 7.1 所示。

图 7.1　info 属性设置

7.4.1.9　isELIgnored 属性

在介绍该属性之前，需要先提及一下表达式语言（expression language，EL）概念，在本书的高级 JSP 部分将会对 EL 进行重点讲解，读者可以先翻到那一章了解一下它的大概意思。而 isELIgonred 属性就是用来设置本 JSP 页面中的 EL 是否可用。该属性的值是一个布尔类型，如果设置为“true”，表示忽视 EL，即 EL 不可用；如果设置为“false”，表示 EL 可以。该属性默认为“false”，即可用。

该属性的设置格式如下：

```
<%@ page isELIgnored=”true” %>
```

下面通过一个实例来介绍该属性的不同设置所带来的效果：例如 EL 表达式${2000%20}，当 isELIgnored 属性设置为 true 时，在 JSP 中会显示字符串${2000%20}；而当 isELIgnored 属性设置为 false 时，表达式则在 JSP 中显示 100。

注意：Web 容器默认的 isELIgnored 属性值为“false”，即 EL 可用。在 JSP2.0 编写中，建议读者尽量使用 EL，这样使得 JSP 的格式更加的一致。

7.4.1.10　isErrorPage 属性

该属性用来设置该 JSP 页面是否使用来作为其他页面的错误处理。当需要统一进行 JSP 错误处理时，就可以使用 isErrorPage 和下面即将介绍的 errorPage 属性，来设置错误处理页面。当 isErrorPage 属性值设置为“true”时，即如下格式：

```
<%@ page isErrorPage=”true” %>
```

这时，设置该属性值的 JSP 页面就可以使用隐式的 exception 内置对象（将在下面一章重点介绍各类内置对象的概念和使用方法）来处理请求异常。isErrorPage 属性的默认值为 false。因为一般情况下，一个 Web 应用只需要一个或少数几个 JSP 页面来统一处理异常信息就可以了。

7.4.1.11　errorPage 属性

该属性和以上介绍的 isErrorPage 属性配套使用。isErrorPage 属性设置能够处理异常的页面，而 errorPage 属性设置需要处理异常的页面，它的属性值是一个 URL 地址，即指定处理异常的页面（即设置 isErrorPage 属性值为“true”的页面）。

例如，进行了如下的 errorPage 设置：

```
<%@ page errorPage=”doError.jsp” %>
```

代码说明：页面设置以上属性之后，当该页面出现异常错误的时候，就会自动跳转到 doError.jsp 页面进行相应的错误处理。

注意：在实际的 Web 项目开发中，一般指定少数几个页面统一进行异常错误的处理（处理异常错误的页面必须设置 isErrorPage 属性值为“true”）。其他的 JSP 页面则可以通过 errorPage 属性来指定处理异常的页面。errorPage 属性没有默认值，当不进行 errorPage 属性设置时，则表示这个页面没有处理的错误会由容器报告出一个未捕获的异常。

7.4.1.12　contentType 属性

contentType 属性的设置在开发过程中是非常重要的，也经常使用到。读者要知道中文乱码一直是困扰开发者的头痛问题，而该属性就是用来对编码格式进行设置。这个设置的属性会最先传递给客户端。它告诉容器要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方式：

- MIME 有如下几种类型：text/plain、text/html（容器默认类型）、text/xml、image/gif、image/jpeg。
- 编码格式：默认的字符编码为 ISO8859-1。可以通过 contentType 属性，指定编码为 UTF-8 或者 GBK 等等，关于编码方面的知识将在十五章节专门讲解。

其中在以上介绍的各属性例子是有问题的，原因就是没有设置 contentType 属性，所用的中文都将显示乱码。下面通过一个实例来具体介绍该属性的设置方法，读者需要认真了解，详细代码如下：

```
<%@ page language="java" contentType="text/html; charset=GBK" %>
<html>
<head></head>
<body>
      <h1> contentType </h1>
<table border="1">
  <tr><td>Hello, contentType!</td></tr>
</table>
</body>
</html>
```

代码说明：text/html 和 charset=GBK 的设置之间是用分号隔开的，它们同属于 contentType 属性值。当设置为 text/html 时，即表示该页面以 HTML 页面格式进行显示。这里设置的编码格式为 GBK。启动 Tomcat 服务器，在浏览器中输入地址 http://localhost:8080/7/contentType.jsp，出现的页面效果如图 7.2 所示。

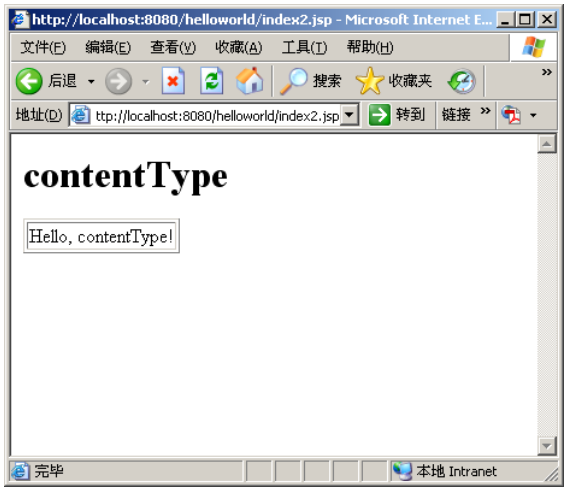

图 7.2　html 输出格式的 JSP 页面

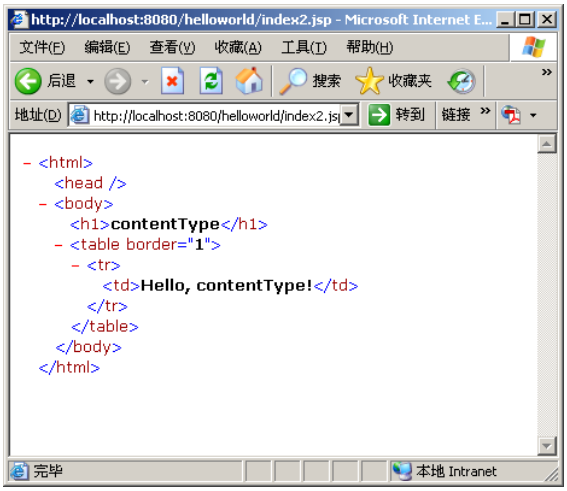

图 7.3　xml 输出格式的 JSP 页面

修改 contentType 属性值，使用“text/xml”替代“text/html”，重新运行该页面，将出现如图 7.3 所示的效果。另外，单击图 7.3 页面上的小图标可以展开和折叠 XML 文档中的元素。在后面将要介绍的

用户注册系统中，将要使用验证码功能，该验证码是以图片的格式显示的，其中就是将 contentType 中的显示格式设置为“image/jpeg”。

7.4.2　文件引用指令include

和 page 指令有所不同，该 include 属性是用来引用外在文件。即指示 Web 容器在编译 JSP 文件的时候，将 include 指定的外在文件插入到当前 JSP 文件中（也可以称为两文件进行合并）。

include 指令的重要性：因为页面代码冗余问题一直存在。现在可以将一些共性的内容（例如标题、导航栏和页脚消息等）写入到一个单独文件中，然后别人的 JSP 页面可以通过 include 指令进行引用。这样可以大大降低页面代码的冗余问题，并且修改也更加方便，这些共性内容只需要修改单独文件即可。

下面还是通过一个实例，来介绍 include 指令的具体使用和需要注意的事项。

首先创建一个 logo.jsp 页面程序，来显示导航栏，详细代码如下：

```
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<style>
a.link {color:#666666; text-decoration:underline; size:12px;}
a.active {color:#003399; text-decoration:none; size:12px;}
a.visited {color:#FF0000; text-decoration:none; size:12px;}
</style>
<table width="60%"    border="1" bordercolordark="#FFFFFF" bordercolorlight="#000000">
  <tr>
    <td><a  href="#"> 新 闻 </a> | <a  href="#"> 生 活 </a> | <a  href="#"> 购 物 区
</a> | <a href="#">论坛</a></td>
  </tr>
</table>
```

代码说明：该页面提取了共性的导航栏。为了巩固前面章节介绍的层叠样式表（CSS），这里编写了 CSS，来调整导航链接的显示格式。

然后创建调用以上文件的 JSP 程序，取名为 include.jsp，详细代码如下：

```
<%@ page language="java" contentType="text/html;charset=gb2312" %>
<html>
<head>
      <title>include 指令－调用</title>
</head>
<body>
<%@ include file="logo.jsp" %>                    <!--  引入 date.jsp 文件  -->
<table width="300">
<tr><td> <h2> JSP 大全</h2></td><tr>
<tr><td align="center">注意 include 指令的使用方法</td></tr>
</table>
</body>
```

代码说明：该页面通过 include 指令引用了外在文件 logo.jsp。这样的好处就是不需要在每个使用导航栏的页面中重写编写 logo.jsp 中的代码，这降低了代码的冗余。在浏览器上运行 include.jsp 页面程序，效果如图 7.4 所示。

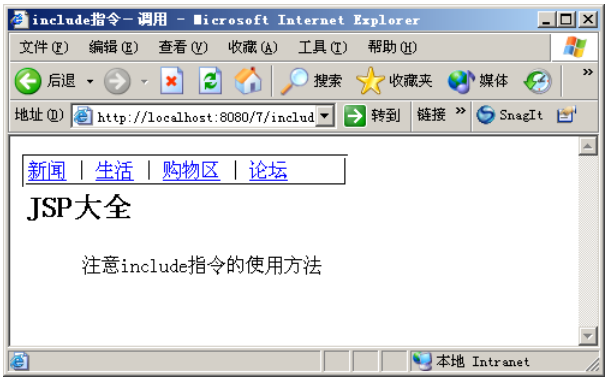

图 7.4　include 指令的使用

对 include 指令的使用，还需要注意一个问题。在第 1 章节的 1.5.2 小节中已经介绍过 JSP 需要经过三个阶段：首先是翻译成 Servlet 源代码；然后编译成二进制可执行码；然后才是处理请求阶段。这里需要重点提及的，通过 include 指令合并的两个页面是在容器翻译阶段发生的。也就是说，容器首先原封不动地将以上的 logo.jsp 页面内容插入到 include.jsp 页面的相应位置，然后才对整个合并内容进行翻译和编译。Include 指令进行的页面合并操作如图 7.5 所示。

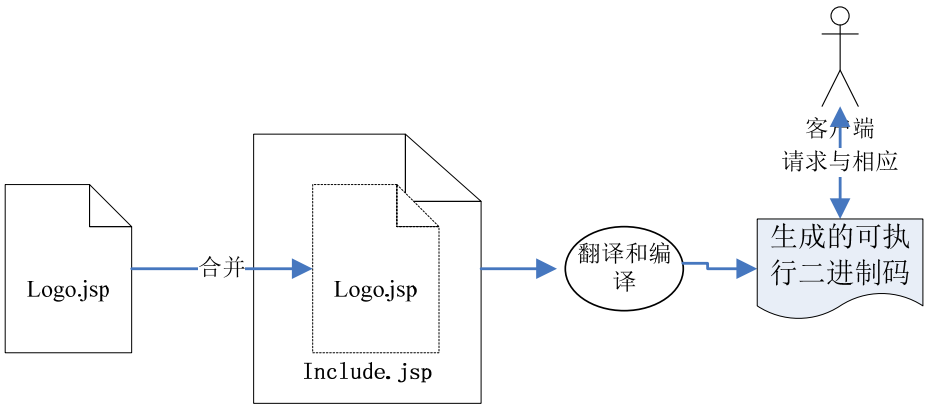

图 7.5　include 指令执行页面合并过程

下面修改以上两个 JSP 页面程序来证明以上的结论。将 logo.jsp 页面中的 contentType 属性重新设置为“text/html; charset=GBK”，然后运行 include.jsp 页面。这时将发现容器报以下错误：

```
Exception：
org.apache.jasper.JasperException: /logo.jsp(1,1) Page directive: illegal to have multiple occurrences of contentType with different values (old: text/html;charset=gb2312, new: text/html;charset=GBK)
```

错误说明：即提示两种编码格式的冲突，由于 include 指令是原封不动地将一个文件的内容插入到另一个页面中。所以在合并的文件中出现了两个不同的 contentType 属性设置。

注意：针对指令 include，如果所包含的 logo.jsp 文件有所修改，容器必须对重新合并的整个单元再次进行编译，而不是单独只编译 logo.jsp 文件。后面章节将要讲的<jsp:include>动作是在请求阶段发生的，它单独编译两个文件，所以它克服了指令 include 所存在的冲突问题。

7.4.3　taglib标签指令

Taglib 指令使用来引用一个标签库或者自定义标签。通过 taglib 指令来告诉容器此 JSP 页面将使用

哪些标签库，并可以给引用的标签库指定一个前缀。在 JSP 文件中使用到这个标签库的时候，就可以使用指定的前缀来标识标签库。

使用自定义标签的 taglib 基本语法如下。

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

以上设置的 taglib 指令，声明了此 JSP 文件使用了一个标签，同时指定了标签的前缀。如下代码使用 taglib 指令引用了标准标签库（JSTL）中的数据库标签：

```
<%@ taglib uri=”http://java.sun.com/jsp/jstl/sql” prefix=”sql” %>
<sql:query var=”books”>
    Select * from books
</sql:query>
```

读者看到这里，可以不怎么理解，在本书的高级部分，将对自定义标签以及标准标签库进行详细介绍。

7.5　JSP标准动作元素

JSP 标准动作元素的使用格式为：<jsp: 标记名>，它使用了严格的 XML 标签语法来表示。这些 JSP 标签动作元素是在用户请求阶段（JSP 执行的三阶段详细介绍见第 1 章的 1.5.2 小节）执行的。这些标准动作元素是内置在 JSP 文件中的，所以可以直接使用，不需要进行引用定义。

接下来需要介绍的标准动作元素包括如下：

- <jsp:useBean>：定义 JSP 页面使用一个 JavaBean 实例。
- <jsp:setProperty>：设置一个 JavaBean 中的属性值。
- <jsp:getProperty>：从 JavaBean 中获取到一个属性值。
- <jsp:include>：在 JSP 页面包含一个外在文件。
- <jsp:forward>：把到达的请求转交另一个页面进行处理。
- <jsp:param>：用于传递参数值。
- <jsp:plugin>：用于指定在客户端浏览器中插入插件的属性。
- <jsp:params>：用于向 HTML 页面上的插件传递参数值。
- <jsp:fallback>：指定客户端不支持插件运行的情况该如何处理。

7.5.1　JSP标准动作介绍

从标准动作元素的调用效果来看，其实每一个标准动作元素都是能够嵌入到 JSP 页面中的一个标记。在该 JSP 页面被翻译成 Servlet 源代码过程中，当容器遇到标准动作元素时就调用相对应的 Servlet 类方法来替代它。所有的标准动作元素的前面都有一个 JSP 前缀作为标记，一般形式如下：

```
<jsp:标记名...　属性　...　/>
```

有些标准动作中间还包含一个体，即一个标准动作元素中又包含了其他标准动作元素或者其他内容。包括体的标准动作使用格式如下：

```
<jsp:标记名...　属性　...　>
  <jsp:标记名...　属性以及参数　.../>
  ...
</jsp:标记名>
```

根据各个标准动作的功能不同，可以将这些标准动作分成如下 6 组：

- JSP 中使用到 JavaBean 的标准动作：<jsp:useBean>定义使用一个 JavaBean 实例，ID 属性定义了实例名称；<jsp:getProperty>从一个 JavaBean 中获取到一个属性值，并将其添加到响应中去；<jsp:setProperty>设置一个 JavaBean 中的属性值。
- 在 JSP 中包含其他 JSP 文件或者 Web 资源的标准动作：<jsp:include>在请求处理阶段包含来自一个 Servlet 或者 JSP 文件的响应，注意和 include 指令的不同。
- 将到达的请求转发给另外一个 JSP 页面或者 Web 资源以便进一步操作的标准动作：<jsp:forward>将某个请求的处理转发到另一个 Servlet 或者 JSP 页面。
- 在其他标准动作的中间指定参数的标准动作：<jsp:param>对使用<jsp:include>或者<jsp:forward>传递到另外一个 Servlet 或者 JSP 页面的请求添加一个传递的参数值。
- 在客户端的页面中嵌入 Java 对象（例如 applet，是运行在客户端的小 Java 程序）的标准动作：<jsp:plugin>根据浏览器类型为 Java 插件生成 Object 或者 Embed 标记；<jsp:params>；<jsp:fallback>。
- 仅仅用于标记文件的标准动作：<jsp:attribute>；<jsp:body>；<jsp:invoke>；<jsp:dobody>；<jsp:element>；<jsp:text>；<jsp:output>。

在实际开发过程中，使用得比较多的还是前五组标准动作，将作重点介绍。

7.5.2　处理JavaBean的标准动作

这一组所包括的标准动作元素有如下：<jsp:useBean>、<jsp:setProperty>和<jsp:getProperty>。

7.5.2.1　<jsp:useBean>标准动作

使用<jsp:userBean>标准动作来引用一个将在 JSP 页面中使用的 JavaBean 类。JavaBean 的使用，实现了逻辑处理和页面显示在一定程度上得到了分离，从而可以增加代码可重用性。将逻辑处理写在一个 JavaBean 类中，在其他所有 JSP 页面程序中即可以使用<jsp:useBean>标准动作来引用该 JavaBean 类。该标准动作的使用形式如下：

```
<jsp:useBean id=”name” class=”package.class” scope=”…” type=”…” />
```

<jsp:useBean>标准动作还可以包含一个体，例如<jsp:setProperty>动作。在第一次创建这个实例的时候，就会使用<jsp:setProperty>动作来进行参数的赋值操作，也就是说，如果该页面中已经存在了该 JavaBean 类的实例，则不会再执行<jsp:setProperty>动作所进行的参数赋值。另外，<jsp:useBean>标准动作也并不意味每次都要创建一个实例，当该页面中已经存在该 JavaBean 的实例，则直接只用这个实例。包含一个体的使用形式如下：

```
<jsp:useBean id="name" class="package.class" scope="..." type="..." >
  <jsp:标记名...  属性以及参数  .../>
  ...
</jsp:useBean>
```

代码说明：使用 class 属性来指定需要实例化的 JavaBean 类，然后使用 id 来标识该实例。Type 属性用来指定该实例化的 Javabean 类将要实现的一个接口或者一个超类；scope 属性可以用来指定该 JavaBean 实例能关联到多个页面。

<jsp:useBean>标准动作元素中所使用的属性如下：

- id：给一个类实例取名作为整个 JSP 页面的惟一标记。如果需要创建一个新 JavaBean 实例，这也是引用这个新的 JavaBean 实例的名字。
- class：这是 JSP 页面引用 JavaBean 组件的完整 Java 类名（一定要包括包名）。如果容器没有找到指定名的类实例，则会使用这个 class 属性指定的完整类名来创建一个新的 JavaBean 实例进

行引用。

- type：此属性告诉容器这里的 JavaBean 实例需要实现一个 Java 接口，或者 JavaBean 实例需要扩展一个超类。在翻译阶段会使用到这个属性。type 属性不是必须添加的，但是必须添加 class 和 type 属性中的其中之一。
- scope：指定这个 JavaBean 在哪种上下文内可用，可以取下面的四个值之一：page（默认值），request，session 和 application。page 表示该 JavaBean 只有在当前页面内可用（保存在当前页面的 PageContext 内）；request 表示该 JavaBean 在当前的客户请求内有效（保存在 ServletRequest 对象内）；session 表示该 JavaBean 对当前 httpSession 内的所有页面都有效。

7.5.2.2　<jsp:setProperty>标准动作

使用<jsp:setProperty>动作可以修改 JavaBean 实例中的属性变量，其中可以有两种使用形式：

（1）<jsp:setProperty>标准动作嵌入在<jsp:useBean>标准动作的体内，但这时只能在 JavaBean 被创建的实例执行。使用形式如下：

```
<jsp:useBean id="myName" ... >
   <jsp:setProperty name="myName" property="someProperty" ... />
</jsp:useBean>
```

（2）在<jsp:useBean>动作的后面使用<jsp:setProperty>标准动作元素，使用形式如下：

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName" property="someProperty" ... />
```

代码说明：在<jsp:useBean>动作之后使用的<jsp:setProperty>标准动作，不管指定的 JavaBean 是新创建的还是直接使用的实例，都会执行。其中<jsp:setProperty>共有四个属性可以选择设置，如下所示：

- name：这个属性是必须要设置，因为通过这个属性才能知道针对哪个 JavaBean 实例的属性值进行赋值。
- property 属性：也是必须要设置的。它告诉容器需要对 JavaBean 实例中的哪些属性进行设值。这里有个特殊的用法：就是把 property 属性值设置为“*”，这表示所有名字和 JavaBean 属性名字匹配的请求参数都将被传递给相应的属性 set 方法。
- value：这个属性是可选的，指定 JavaBean 属性的值。
- param：这个属性和 value 属性不能同时使用，只能使用其中一个。当两个属性都没有在<jsp:setProperty>动作中指定时，指定的属性变量将使用 JavaBean 中的默认值（例如类中构造函数所默认的值）。如果使用 param 属性，容器就会把 property 指定的属性变量设置为 param 指定的请求参数的值。

7.5.2.3　<jsp:getProperty>标准动作

该标准动作是和前一个<jsp:setProperty>标准动作相对应，用来提取指定的 JavaBean 属性值，然后转换成字符串输出。该动作有两个必须要设置的属性，如下：

- name：表示 JavaBean 在 JSP 中的标记。
- property：表示提取 JavaBean 中的哪个属性的值。

7.5.2.4　综合实例

为了加深对以上介绍的三个标准动作的理解，这里列举一个综合实例。创建一个名为 beanTest.jsp 页面程序，详细代码如下：

```
<%@ page contentType="text/html;charset=GBK" language="java" %>
<HTML>
<HEAD>
```

```
<TITLE>Reusing JavaBeans in JSP</TITLE>
</HEAD>
<BODY>
<CENTER>
<TABLE BORDER=1>
<P>
<!—HTML 注释：这里定义一个类实例，以 testBean 作标记，并给 message 属性变量赋"Hello"作为初始值-->
<jsp:useBean id="testBean" class="com.helloworld.SimpleBean" >
    <jsp:setProperty name="testBean" property="message" value="Hello" />
</jsp:useBean>
<H1>通过<jsp:getProperty>动作得到的 message 属性值:
    <I> <jsp:getProperty name="testBean" property="message" /> </I></H1>
<%--JSP 标记注释：下面给 testBean 实例中的 message 属性变量重新赋"Hello WWW"值 --%>
<jsp:setProperty name="testBean" property="message" value="Hello WWW" />
<H1>通过显现方式获取 testBean 实例中的 message 属性值:
    <I> ${testBean.message} </I></H1>
</BODY>
</HTML>
```

代码说明：该程序首先通过<jsp:useBean>动作来引用一个 JavaBean 类（在 com.helloworld 包中的 SimpleBean 文件），并且以 testBean 作为此 JavaBean 实例的标记。在<jsp:useBean>动作体中还包含一个<jsp:setProperty>动作，通过该动作来给 JavaBean 中的 message 属性变量赋值。然后可以通过<jsp:getProperty>动作来获取该 JavaBean 实例中的 message 变量值，这是通过 name 和 property 两个属性来指定的。在<jsp:useBean>动作后面使用<jsp:setProperty>动作，可以修改 JavaBean 中的属性变量值。最后使用了后面将要介绍的 EL 表达式来获取 JavaBean 中的属性值。

下面创建以上引用的 JavaBean 文件，容器默认的类文件地址必须在 Web 模块的 WEB-INF/classes 目录下。创建的该 JavaBean 文件代码如下：

```
package com.helloworld;
public class SimpleBean {
private String message = "No message specified";

public String getMessage() {                    //标准的 getXxxx()方法
return (message);
}

public void setMessage(String message) {        //标准的 setXxxx()方法
this.message = message;
}
}
```

代码说明：这个 JavaBean 中的属性 message 和<jsp:setProperty>或者<jsp:getProperty>动作中的 property 设置值是对应的。方法的格式为 getXxxx()和 setXxxx()，这样容器才能识别。其中 setXxxx()方法来设置，而 getXxxx()方法来获取变量值。

此时，该实例仍然不能成功运行，因为上面创建的 JavaBean 文件还是 Java 源代码，是不可执行的，还需要将它编译成二进制文件（后缀为.class）。下面介绍一下手动编译 Java 源文件的过程，如下：

（1）首先通过单击“开始”|“运行”命令，打开“运行”对话框，如图 7.6 所示。

（2）在输入框中输入 cmd 命令行，然后单击“确定”按钮，弹出如图 7.7 所示的“cmd”窗口。

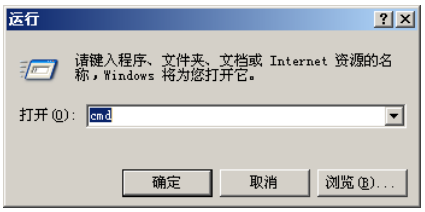

图 7.6　运行对话框

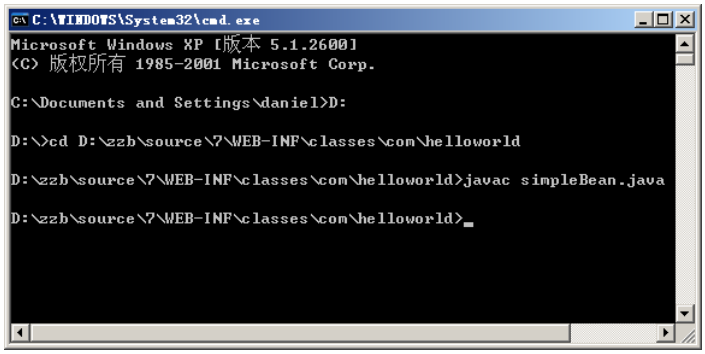

图 7.7　cmd 窗口

（3）由于 JavaBean 类文件是放在 Web 模板的 WEB-INF/classes 目录下，加上包为 com.helloworld。所以该 JavaBean 的绝对路径为 D:\zzb\source\7\WEB-INF\classes\com\helloworld（“7”为 Web 模块名，并创建在 D:\zzb\source\目录下）。在 cmd 窗口中输入“cd D:\zzb\source\7\WEB-INF\classes\com\helloworld”命令将目录指定到 JavaBean 所在的位置。

（4）然后输入“javac TestBean.java”命名将 SimpleBean.java 源文件编译成二进制码。具体操作见图 7.8 所示。操作完成后，将会发现在 D:\zzb\source\7\WEB-INF\classes\com\helloworld 目录下多了一个 Class 二进制文件。

这样就成功了对 SimpleBean.java 源代码进行了编译。其实手动编译还是显得非常麻烦的，当源文件非常多的时候，显然这样非常费时间。在后面章节中将要介绍的 Eclipse 开发工具，可以自动将源代码编译到指定目录下，并且将源代码和 class 文件进行分离（只要进行简单的设置）。

编译完 SimpleBean.java 源文件之后，就可以通过浏览器来访问之上创建的 beanTest.jsp 页面，运行结果如图 7.8 所示。

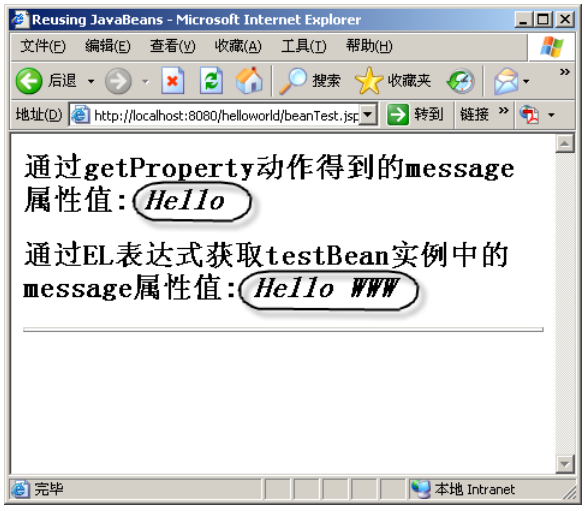

图 7.8　beanTest 页面

注意 :JavaBean 的取值和赋值方法编写格式为 getXxxx()和 setXxxx(　)。(这里的 Xxxx 为 JavaBeawn 中定义的属性) 。

7.5.3　引用外部文件的标准动作<jsp:include>

该标准动作和前面介绍的 include 指令方法非常类似，它也是将特定的外在文件插入到当前页面中，其具体的使用语法如下：

```
<jsp:include page=”…url…” flush=”true|false” />
```

该标准动作还可以包含一个体，具体形式可以如下：

```
<jsp:include page=”…url…” flush=”true|false” >
   <jsp:param ….>
</jsp:include>
```

通过在<jsp:include>动作体中的使用<jsp:param>动作，可以用来指定 JSP 页面中可用的其他请求参数，之后可以在当前 JSP 文件以及引用的外在文件中使用这些请求参数。

另外一个需要重地说明的，就是在介绍<include>指令时提及的与<jsp:include>标准动作存在的差异。Include 指令是在 JSP 翻译时进行文件的合并，然后对合并的整体文件进行编译；而<jsp:include>标准动作则首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。图 7.9 展示了使用<jsp:include>标准动作进行文件合并的过程。

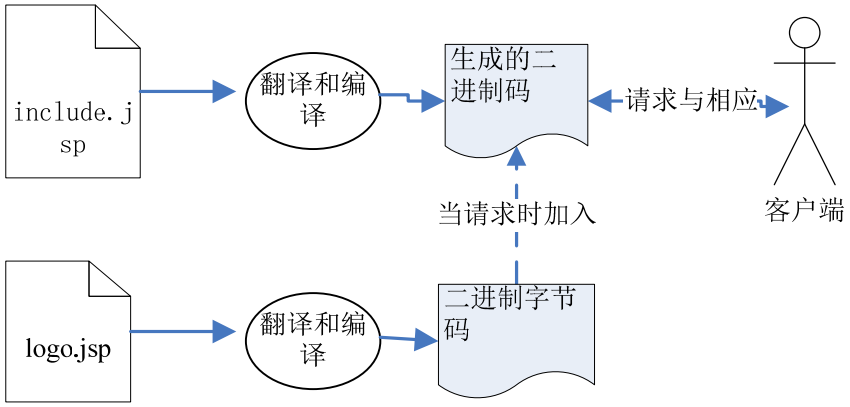

图 7.9　<jsp:include>标准动作的使用

上面的图还是以 include 指令中的实例为例。和 include 指令的图进行对比可以看出这两种文件合并方式不同。<jsp:include>标准动作是将编译之后的二进制文件进行合并，而且是发生在用户请求阶段。

再次修改 include 指令实例中的 logo.jsp 文件，将 page 指令中的 contentType 属性设置为“text/html；charset=GBK”，然后将 include.jsp 文件中的<%@ include file="logo.jsp" %>替换为<jsp:include page=”logo.jsp” flush=”true” />。

重新在 IE 浏览器中运行 include.jsp 页面，将不会出现和<include>指令所类似的错误。这就是由于<jsp:include>标准动作是先编译再引入的（当客户端请求的时候）。使用<jsp:include>标准动作来引入外在文件，当修改外在文件的时候，也就不需要像<include>指令一样对整体翻译后的文件（包括 index3.jsp 文件）重新编译。

7.5.4　进行请求转移的标准动作<jsp:forward>

<jsp:forward>标准动作把请求转移到另外一个页面。这个标准动作只有一个属性 page，page 属性包含一个相对 url 地址。例如下面的例子：

```
<jsp:forward page="/utils/errorReporter.jsp" />
<jsp:forward page="<%= someJavaExpression %>" />
```

第一行的 page 值是直接给出的，第二行的<jsp:forward>标准动作中的 page 值是在请求时动态计算的。

7.5.5　参数设置的标准动作<jsp:param>

该<jsp:param>标准动作在前面已经多次提及，它一般和<jsp:include>以及<jsp:forward>等配套使用，用来进行参数的传递，其一般使用形式如下：

```
<jsp:param name="…名称…" value="…值…" />
```

每个<jsp:param>标准动作都会创建一个有名而且有值的参数。这样通过<jsp:include>标准动作所包含的外在文件以及通过<jsp:forward>转移到的另外页面都可以使用这些参数。通过下面的例子来加深对<jsp:param>标准动作的理解：

```
<html>
<body>
  <jsp:include page="date.jsp">
    <jsp:param name="serverName" value="zzb"/>
  </jsp:include>
</body>
</html>
```

这说明在请求时所包含的 date.jsp 文件中可以使用通过<jsp:param>标准动作定义的 serverName 参数。

7.5.6　处理插件的标准动作

下面将要介绍的标准动作是用来使用相应插件（例如 applet）。主要有如下三个标准动作：

- <jsp:plugin>：用于指定在客户端运行的插件。
- <jsp:params>：用于向引用的插件传递参数值，一般包含在<jsp:plugin>动作体中。
- <jsp:fallback>：当客户端不支持 Java 插件的时候，所执行的内容。

如图 7.10 展示了插件在客户端进行调用和运行的过程。

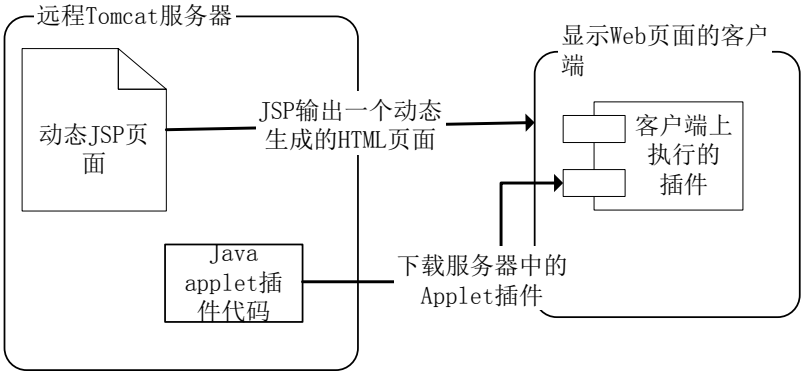

图 7.10　插件的调用

从图 7.11 可以看出，客户端浏览器首先会从服务器端将相应的插件下载下来。在运行插件的时候（一般为 Applet 插件），一般需要客户端启动一个 Java VM（Java 虚拟机）。Java 插件在客户端运行是安全的，因为它不可操作客户端机器任何其他资源。

7.5.6.1　<jsp:plugin>标准动作

使用<jsp:plugin>标准动作可以在页面程序中插入一个 Java 插件，一般为 Java Applet 小程序（但也可以是任何的 JavaBean 类），它是随着页面程序一起传输到客户端，并且在客户端运行。<jsp:plugin>标准动作的一般使用形式如下：

```
<jsp:plugin type="…applet 或者 JavaBean…" code="…" codebase="…" archive="…"…>
  …
</jsp:plugin>
```

代码说明：在该标准动作中可以设置多个属性，其中大部分是<OBJECT>或者<EMBED>标记的 HTML 属性，例如：name、code。Codebase、archive、align、width、height、jreversion 以及 title 等。其中 type 属性是用来指定该插件是 applet 小程序还是 JavaBean 类。至于这些属性具体设置方法见下面将要介绍的一个综合实例。

7.5.6.2　<jsp:params>标准动作

<jsp:params>标准动作的体中是由多个<jsp:param>动作组成，并且该标准动作只能使用在<jsp:plugin>标记的体中。其一般使用形式如下：

```
<jsp:plugin type="...applet|JavaBean..." code="..." ...>
  <jsp:params>
    <jsp:param .../>
    <jsp:param .../>
    ...
  </jsp:params>
</jsp:plugin>
```

代码说明：<jsp:params>动作体中包括多个<jsp:param>动作来给插件类中的属性变量赋值。每一个<jsp:param>向运行在客户端的插件传递一个参数。

7.5.6.3　<jsp:fallback>标准动作

该标签是和<jsp:plugin>标准动作配合使用的，它是告诉客户端浏览器，当客户端不支持该插件运行时，将要显示的 HTML 页面或者 JSP 代码。以下是该标准动作的一般使用形式：

```
<jsp:fallback>
  ...  客户端浏览器不支持插件运行时显示 html 或者 JSP 代码  ...
</jsp:fallback>
```

代码说明：体中潜入的代码只有在客户端不支持该插件运行时才执行。

7.5.6.4　实例讲解

介绍了以上三个标准动作之后，读者可能对其使用方法还是感觉很模糊。下面将通过一个实例来具体介绍这些标准动作的使用。首先在 Web 模块中创建一个 plugin.jsp 文件，在此文件中将使用到以上三个标准动作，用来引用一个 Applet 插件。该引入的 Applet 插件将在页面上画一个园，并且园的半径以及显示的颜色变量是通过<jsp:params>和<jsp:param>来传递的。该创建的 JSP 页面代码如下：

```
<%@ page language="java" contentType="text/html; charset=GBK"%>
<HTML>
<TITLE>用 plugin 加载 Applet</TITLE>
<BODY>

<CENTER> 实现画圆功能 </CENTER>
<BR><HR><BR>
<CENTER>
<!-- 用 plugin 加载 applet -->
```

```
<jsp:plugin type="applet" code=" circleApplet.class" width=”100” height=”100” align=”center”>
  <jsp:params>
    <jsp:param name="radius" value="30"/>                    <!—设置属性 radius（圆半径） -->
    <jsp:param name="color" value="0xff0000"/>               <!—设置属性 color（圆显示的颜色 -->
  </jsp:params>
  <jsp:fallback>无法加载 Applet</jsp:fallback>               <!—当无法加载时显示这段代码 -->
</jsp:plugin>
</CENTER>
</BODY>
</HTML>
```

代码说明：使用<jsp:plugin>动作中的 code 属性指定引用一个名为 circleApplet.class 的 Applet 文件（该 Applet 文件是通过继承 Applet 超类实现的，从下面的源代码可以看出），其中 type 属性指定该插件为 applet。另外，width 属性显示该插件的显示宽度为 100，height 属性指定显示高度，以及 align 属性指定显示位置。

在<jsp:params>体中，定义了多个<jsp:param>来给该 Applet 插件中的 color 和 radius 属性变量传值。<jsp:fallback>动作指定当客户端不支持该插件运行时将要显示的内容。

接下来创建 circleApplet.java 文件，详细代码如下：

```
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.Color;
public class circleApplet extends Applet{                  //编写的 applet 小程序必须要继承 Applet 类
    private Color thecolor = null;                          //定义颜色变量
    private int radius = 5;                                 //定义半径变量
    public void init() {                                    //进行初始化操作
      thecolor = Color.decode(this.getParameter("color"));      //取得页面中设置 color 参数值
      radius = Integer.parseInt(this.getParameter("radius"));   //取得页面中设置的 radius 参数值
    }
    public void paint(Graphics g){                              //进行绘图的方法
        g.setColor(thecolor);                                   //设置圆的颜色
        g.fillOval(10,10,radius*2,radius*2);                    //画圆
    }
}
```

代码说明：按照以上介绍的手动方法编译这个 Java 源文件，然后把编译好的 class 文件放在和 pluginTest.jsp 文件同一个目录下（容器是从引用 applet 的 JSP 文件目录下开始查找 applet 类）。

该编写的 Applet 必须继承超类 java.applet.Applet。在该编写的 Applet 类中定义了两个变量（这两个变量是和 plugin.jsp 文件中传递的 radius 和 color 变量对应）和两个方法。init()方法接受 JSP 页面中<jsp:param>动作传递的参数，paint()方法根据给定的 radius 半径和 color 颜色画一个圆。

然后在浏览器中输入 plugin.jsp 文件的 URL 地址，可以查看到如图 7.11 所示的效果。

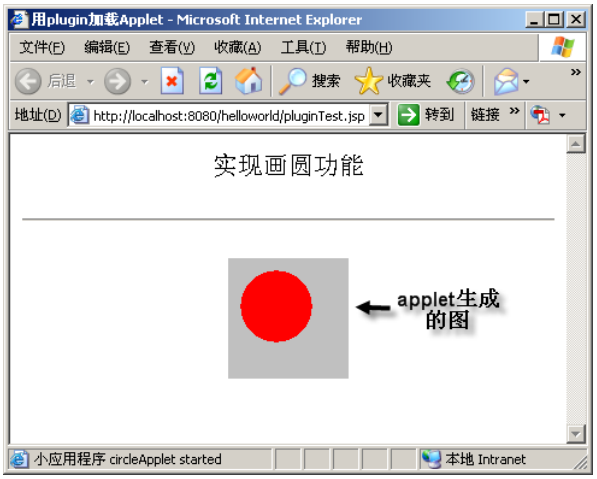

图 7.11　applet 画圆

至此就介绍完了 JSP 中的用标准动作。对于其他不常见的标准动作，在此就不作详细介绍。当读者在实际开发过程使用到时，再查阅相关书籍。

7.6　本章小结

如果说第 1 章是让读者认识 JSP，第二、三章是让读者了解 JSP 运行机制和原理，那么从第 4 章开始就具体向读者介绍 JSP 页面的编写规则和方法。前三章主要介绍了 JSP 页面页面开发过程中常使用到的 HTML 标签、JavaScript 浏览器脚本语言以及 CSS 层叠样式，这些技术都是倾向于在前台使用。这一章主要向读者介绍 JSP 页面编写的语法，包含 JSP 指令和标准动作的介绍。下一章将介绍 JSP 中的多个内置对象，这些对象的使用加大了 JSP 使用的方便。