JSP程序设计教程

第3章 JSP语法

第 3 章 JSP语法

- 3.1 了解JSP的基本构成 √
- 3.2 JSP的指令标识 √
- 3.3 JSP的脚本标识 √
- 3.4 JSP的注释 √
- 3.5 动 作 标 识 √

3.1 了解JSP的基本构成

在学习JSP语法之前，首先来初步了解一下JSP页面的基本结构。请看下面的代码：

```
<!-- JSP中的指令标识 -->
<%@ page language="java" contentType="text/html; charset=gb2312" %>
<%@ page import="java.util.Date" %>
<!-- HTML标记语言 -->
<html>
  <head><title>JSP页面的基本构成</title></head>
  <body>
   <center>
<!-- 嵌入的Java代码 -->
    <% String today=new Date().toLocaleString(); %>
<!-- JSP表达式 -->
    今天是：<%=today%>
<!-- HTML标记语言 -->
   </center>
  </body>
</html>
```

在上面的代码中，并没有包括JSP中的所有元素，但它仍然构成了一个动态的JSP程序。访问包含了该代码的JSP页面后，将显示用户访问该页面的当前时间。暂且不对其功能实现进行讲解，先来介绍该页面的组成元素。

3.1 Java语言基础

3.1.1 JSP中的指令标识 √

3.1.2 HTML标记语言 √

3.1.3 嵌入的Java代码片段 √

3.1.4 JSP表达式 √

3.1.1 JSP中的指令标识

利用JSP指令可以使服务器按照指令的设置来执行动作和设置在整个JSP页面范围内有效的属性。例如，上述代码中的第一个page指令指定了在该页面中编写JSP脚本使用的语言为Java，并且还指定了页面响应的MIME类型和JSP字符的编码；第二个page指令所实现的功能类似于Java中的import语句，用来向当前的JSP文件中导入需要用到的包文件。

3.1.2 HTML标记语言

HTML标记在JSP页面中作为静态的内容，浏览器将会识别这些HTML标记并执行。在JSP程序开发中，这些HTML标记语言主要负责页面的布局、设计和美观，可以说是网页的框架。

3.1.3 嵌入的Java代码片段

嵌入到JSP页面中的Java代码，在客户端浏览器中是不可见的。它们需要被服务器执行，然后由服务器将执行结果与HTML标记语言一同发送给客户端进行显示。通过向JSP页面中嵌入Java代码，可以使该页面生成动态的内容。

3.1.4 JSP表达式

JSP表达式主要用于数据的输出。它可以向页面输出内容以显示给用户，还可以用来动态地指定HTML标记中属性的值。

3.2 JSP的指令标识

指令标识在客户端是不可见的，它是被服务器解释并被执行的。通过指令标识可以使服务器按照指令的设置来执行动作和设置在整个JSP页面范围内有效的属性。在一个指令中可以设置多个属性，这些属性的设置可以影响到整个页面。

在JSP中主要包含3种指令，分别是page指令（页面指令）、include指令和taglib指令。

指令通常以“<%@”标记开始，以“%>”标记结束，以上3种指令的通用格式如下：

3.2 JSP的指令标识

<%@ 指令名称 属性1="属性值" 属性2="属性值" …%>

下面将分别介绍JSP的3种指令格式。

- 3.2.1 使用page指令 √
- 3.2.2 使用include指令 √
- 3.2.3 使用taglib指令 √

3.2.1 使用page指令

page指令即页面指令，可以定义在整个JSP页面范围内有效的属性，其使用格式如下：

```
<%@ page attribute1="value1" attribute2="value2" …%>
```

page指令可以放在JSP页面中的任意行，但为了利于程序代码的阅读，习惯上放在文件的开始部分。Page指令具有多种属性，通过这些属性的设置可以影响到当前的JSP页面。

例如，在页面中正确设置当前页面响应的MIME类型为text/html，如果MIME类型设置不正确，则当服务器将数据传输给客户端进行显示时，客户端将无法识别传送来的数据，从而不能正确地显示内容。

3.2.1 使用page指令

Page指令中除import属性外，其他属性只能在指令中出现一次。Page指令具有的属性如下：

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

虽然Page指令具有如此多的属性，但在实际编程过程中，并不是每个属性都必须一一列出，其中很多属性可以忽略，此时Page指令将使用这些属性的默认值来设置JSP页面

3.2.1 使用page指令

language属性：设置当前页面中编写JSP脚本使用的语言，默认值为java，例如：

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

上述代码设置了当前页面中使用Java语言来编写JSP脚本，目前只能设置为Java。

contentType属性：设置页面响应的MIME类型，通常被设置为text/html，例如：

```
<%@ page contentType="text/html" %>
```

如果该属性设置不正确，如设置为text/css，那么客户端浏览器在显示HTML样式时，不能对HTML标识进行解释，而直接显示HTML代码。

3.2.1 使用page指令

在该属性中还可以设置JSP字符的编码，例如：

```
<%@ page contentType="text/html;charset=gb2312" %>
```

默认的编码为ISO-8859-1。

import 属性：import属性类似于Java中的import语句，用来向JSP文件中导入需要用到的包。在Page指令中可多次使用该属性来导入多个包。例如：

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

或者通过逗号间隔，来导入多个包。

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

3.2.1 使用page指令

在JSP中已经默认导入了以下包：

```
java.lang.*
javax.servlet.*
javax.servlet.jsp.*
javax.servlet.http.*
```

所以，即使没有通过import属性进行导入，在JSP页面中也可以调用上述包中的类。

若要在页面中使用编写的JavaBean，也可通过import属性来导入。还可以通过<jsp:useBean>动作标识来创建一个JavaBean实例进行调用。

3.2.1 使用page指令

extends属性：extends属性用于指定将一个JSP页面转换为Servlet后继承的类。在JSP中通常不会设置该属性，JSP容器会提供继承的父类。并且如果设置了该属性，一些改动会影响JSP的编译能力。

session属性：该属性默认值为true，表示当前页面支持session，设为false表示不支持session。

buffer属性：该属性用来设置out对象（JspWriter类对象）使用的缓冲区的大小。若设置为none，表示不使用缓存，而直接通过PrintWriter对象进行输出；如果将该属性指定为数值，则输出缓冲区的大小不应小于该值，默认值为8KB（因不同的服务器而不同，但大多数情况下都为8KB）。

3.2.1 使用page指令

autoFlush属性：该属性默认值为true，表示当缓冲区已满时，自动将其中的内容输出到客户端。如果设为false，则当缓冲区中的内容超出其设置的大小时，会产生“JSP Buffer overflow”溢出异常。

注意：若buffer属性设为none，则autoFlush不能设为false。

isThreadSafe属性：该属性默认值为true，表示当前JSP页面被转换为Servlet后，会以多线程的方式来处理来自多个用户的请求；如果设为false，则转换后的Servlet会实现SigleThreadModel接口，该Servlet将以单线程的方式来处理用户请求，即其他请求必须等待直到前一个请求处理结束。

3.2.1 使用page指令

info属性：该属性可设置为任意字符串，如当前页面的作者或其他有关的页面信息。可通过Servlet.getServletInfo()方法来获取设置的字符串。例如：

```
<%@ page info="This is index.jsp!" %>
<%=this.getServletInfo()%>
```

访问页面后，将显示：This is index.jsp!

errorPage属性：该属性用来指定一个当前页面出现异常时所要调用的页面。如果属性值是以“/”开头的路径，则将在当前应用程序的根目录下查找文件；否则，将在当前页面的目录下查找文件。

3.2.1 使用page指令

isErrorPage属性：将该属性值设为true，此时在当前页面中可以使用exception异常对象。若在其他页面中通过errorPage属性指定了该页面，则当前者出现异常时，会跳转到该页面，并可在该页面中通过exception对象输出错误信息。相反，如果将该属性设置为false，则在当前页面中不能使用exception对象。该属性默认值为false。

【例3-1】 errorPage属性及isErrorPage属性的应用

isELIgnored属性：通过该属性的设置，可以使JSP容器忽略表达式语言“${}”。其值只能为true或false。设为true，则忽略表达式语言。

3.2.1 使用page指令

pageEncoding 属性：该属性用来设置JSP页面字符的编码。默认值为ISO-8859-1。

3.2.2 使用include指令

该指令用于在当前的JSP页面中，在当前使用该指令的位置嵌入其他的文件，如果被包含的文件中有可执行的代码，则显示代码执行后的结果。该指令的使用格式如下：

```
<%@ include file="文件的绝对路径或相对路径" %>
```

file属性：该属性指定被包含的文件，该属性不支持任何表达式，也不允许通过如下的方式来传递参数。

```
<%@ include file="welcome.jsp?name=yxq" %>
```

如果该属性值以“/”开头，那么指定的是一个绝对路径，将在当前应用的根目录下查找文件；如果是以文件名称或文件夹名开头，那么指定的是一个相对路径，将在当前页面的目录下查找文件。

3.2.2 使用include指令

使用include指令引用外部文件，可以减少代码的冗余。例如，有两个JSP页面都需要应用下图所示的网页模板进行布局。

<table>
<tr><td colspan="2">LOGO图片区：top.jsp</td></tr>
<tr><td>侧栏
left.jsp</td><td>内容显示区：main.jsp</td></tr>
<tr><td colspan="2">页尾:end.jsp</td></tr>
</table>

其中，这两个页面中的LOGO图片区、侧栏和页尾的内容都不会发生变化。如果通过基本JSP语句来编写这两个页面，会导致编写的JSP文件出现大量的冗余代码，不仅降低了开发进程而且会给程序的维护带来很大的困难。为了解决该问题，可以将这个复杂的页面分成若干个独立的部分，将相同的部分在单独的JSP文件中进行编写。

3.2.2 使用include指令

这样在多个页面中应用上述的页面模板时，就可通过include指令在相应的位置上引入这些文件，从而只需对内容显示区进行编码即可。类似的页面代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<table>
        <tr><td colspan="2"> <%@ include file="top.jsp"%> </td></tr>
        <tr>
                <td><%@ include file="side.jsp"%></td>
                <td>在这里对内容显示区进行编码</td>
        </tr>
        <tr><td colspan="2"><%@ include file="end.jsp"%></td></tr>
</table>
```

3.2.3 使用taglib指令

在JSP页面中，可以直接使用JSP提供的一些动作元素标识来完成特定功能，如使用<jsp:include>包含一个文件。通过使用taglib指令，开发者就可以在页面中使用这些基本标识或自定义的标识来完成特殊的功能。
taglib指令的使用格式如下：

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

uri属性：该属性指定了标签描述符，该描述符是一个对标签描述文件（*.tld）的映射。在tld标签描述文件中定义了该标签库中的各个标签名称，并为每个标签指定一个标签处理类。

3.2.3 使用taglib指令

prefix属性：该属性指定一个在页面中使用由uri属性指定的标签库的前缀。前缀不能命名为jsp、jspx、java、javax、sun、servlet和sunw。

开发者可通过前缀来引用标签库中的标签。以下为一个简单的使用JSTL的代码：

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

该段代码通过<c:set>标签将hello值赋给了变量name。

3.3 JSP的脚本标识

在JSP页面中，脚本标识使用得最为频繁。因为它们能够很方便、灵活地生成页面中的动态内容，特别是Scriptlet脚本程序。JSP中的脚本标识包括以下三种元素：声明标识（Declaration）、JSP表达式（Expression）和脚本程序（Scriptlet）。通过这些元素，就可以在JSP页面中像编写Java程序一样来声明变量、定义函数或进行各种表达式的运算。在JSP页面中需要通过特殊的约定来表示这些元素，并且对于客户端这些元素是不可见的，它们由服务器执行。

3.3 JSP的脚本标识

- 3.3.1 JSP表达式（Expression） √
- 3.3.2 声明标识（Declaration） √
- 3.3.3 脚本程序（Scriptlet） √

3.3.1 JSP表达式

表达式用于向页面中输出信息，其使用格式为：

```
<%= 变量或可以返回值的方法或Java表达式 %>
```

特别要注意，“<%”与“=”之间不要有空格。

JSP表达式在页面被转换为Servlet后，转换为了out.print()方法。所以JSP表达式与JSP页面中嵌入到小脚本程序中的out.print()方法实现的功能相同。如果通过JSP表达式输出一个对象，则该对象的toString()方法会被自动调用，表达式将输出toString()方法返回的内容。

JSP表达式可以应用到以下几种情况。

（1）向页面输出内容，例如下面的代码：

3.3.1 JSP表达式

```
<% String name="www.xxx.com"; %>
用户名：<%=name%>
```

运行该段代码将显示：
用户名：www.xxx.com

（2）生成动态的链接地址，例如下面的代码：

```
<% String path="welcome.jsp"; %>
<a href="<%=path%>">链接到welcom.jsp</a>
```

运行该段代码将生成下面的HTML代码：
链接到welcome.jsp

（3）动态指定Form表单处理页面，例如下面的代码：

```
<% String name="logon.jsp"; %>
<form action="<%=name%>"></form>
```

运行该段代码将生成下面的HTML代码：
<form action="logon.jsp"></form>

3.3.1 JSP表达式

（4）为通过循环语句生成的元素命名，例如下面的代码：

```
<% for(int i=1;i<3;i++){ %>
        file<%=i%>：<input type="text"
name="<%="file"+i%>"><br>
<% } %>
```

运行该段代码将生成下面的HTML代码：

file1：<input type="text" name="file1">

file2：<input type="text" name="file2">

3.3.2 声明标识（Declaration）

在JSP页面中可以声明变量或方法，其声明格式为：

```
<%! 声明变量或方法的代码 %>
```

特别要注意，在“<%”与“!”之间不要有空格。声明的语法与在Java语言中声明变量和方法时是一样的。
在页面中通过声明标识声明的变量和方法，在整个页面内都有效，它们将成为JSP页面被转换为Java类后类中的属性和方法。并且它们会被多个线程即多个用户共享。也就是说，其中的任何一个线程对声明的变量或方法的修改都会改变它们原来的状态。它们的生命周期从创建到服务器关闭后结束。
下面将通过一个具体实例来介绍声明标识的应用。

【例3-2】 一个简单的网站计数器

3.3.3 脚本程序（Scriptlet）

脚本程序是在JSP页面中使用“<%”与“%>”标记起来的一段Java代码。在脚本程序中可以定义变量、调用方法和进行各种表达式运算，且每行语句后面要加入分号。在脚本程序中定义的变量在当前的整个页面内都有效，但不会被其他的线程共享，当前用户对该变量的操作不会影响到其他的用户。当变量所在的页面关闭后就会被销毁。

脚本程序使用格式如下：

```
<% Java程序片段 %>
```

脚本程序的使用比较灵活，它所实现的功能是JSP表达式无法实现的。

【例3-3】 脚本程序的应用

3.4 JSP的注释

在JSP页面中可以应用多种注释，如HTML中的注释、Java中的注释和在严格意义上说属于JSP页面自己的注释：带有JSP表达式和隐藏的注释。在JSP规范中，它们都属于JSP中的注释，并且它们的语法规则和运行的效果有所不同。下面将介绍JSP中的各种注释。

- 3.4.1 HTML中的注释 √
- 3.4.2 带有JSP表达式的注释 √
- 3.4.3 隐藏注释 √
- 3.4.4 脚本程序（Scriptlet）中的注释 √

3.4.1 HTML中的注释

JSP文件是由HTML标记和嵌入的Java程序片段组成的，所以在HTML中的注释同样可以在JSP文件中使用。注释格式如下：

```
<!-- 注释内容 -->
```

【例3-4】 HTML注释的应用

3.4.2 带有JSP表达式的注释

在HTML注释中可以嵌入JSP表达式，注释格式如下：

```
<!-- comment<%=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，从而来执行该表达式，而对注释中的其他内容不做任何操作。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。

【例3-5】 带有JSP表达式注释的应用

3.4.3 隐藏注释

在前面已经介绍了如何应用HTML中的注释，这种注释虽然在客户端浏览页面时不会看见，但它却存在于源代码中，可通过在客户端查看源代码看到被注释的内容。所以严格来说，这种注释并不安全。下面将介绍一种隐藏注释，注释格式如下：

```
<%-- 注释内容 --%>
```

使用该方法注释的内容，不仅在客户端浏览时看不到，而且即使是通过在客户端查看HTML源代码，也不会看到，所以安全性较高。

【例3-6】 隐藏注释的应用

3.4.4 脚本程序（Scriptlet）中的注释

在脚本程序中所包含的是一段Java代码，所以在脚本程序中的注释和在Java中的注释是相同的。脚本程序中包括下面3种注释方法。

1．单行注释

单行注释的格式如下：

```
// 注释内容
```

该方法进行单行注释，符号“//”后面的所有内容为注释的内容，服务器对该内容不进行任何操作。因为脚本程序在客户端通过查看源代码是不可见的，所以在脚本程序中通过该方法被注释的内容也是不可见的，并且在后面将要提到的通过多行注释和提示文档进行注释的内容都是不可见的。

3.4.4 脚本程序（Scriptlet）中的注释

【例3-7】 单行注释的应用

2．多行注释

多行注释的是通过“/*”与“*/”符号进行标记，它们必须成对出现，在它们之间输入的注释内容可以换行。注释格式如下：

```
/*
  注释内容1
  注释内容2
  …
*/
```

为了程序代码的美观，开发员习惯上在每行的注释内容前面加入一个“*”号，构成以下的注释格式：

3.4.4 脚本程序（Scriptlet）中的注释

```
/*
 * 注释内容1
 * 注释内容2
 * …
 */
```

同单行注释一样，在“/*”与“*/”之间被注释的所有内容，即使是JSP表达式或其他的脚本程序，服务器都不会做任何处理，并且多行注释的开始标记和结束标记可以不在同一个脚本程序中同时出现。

【例3-8】 多行注释的应用

3．提示文档注释

3.4.4 脚本程序（Scriptlet）中的注释

该种注释会被Javadoc文档工具生成文档时所读取，文档是对代码结构和功能的描述。

注释格式如下：

```
/**
   提示信息1
   提示信息2
   …
*/
```

该注释方法与前面介绍的多行注释很相似，但细心的读者会发现它是以“/**”符号作为注释的开始标记，而不是“/*”。与多行注释一样，被“/**”和“/*”符号注释的所有内容，服务器都不会做任何处理。

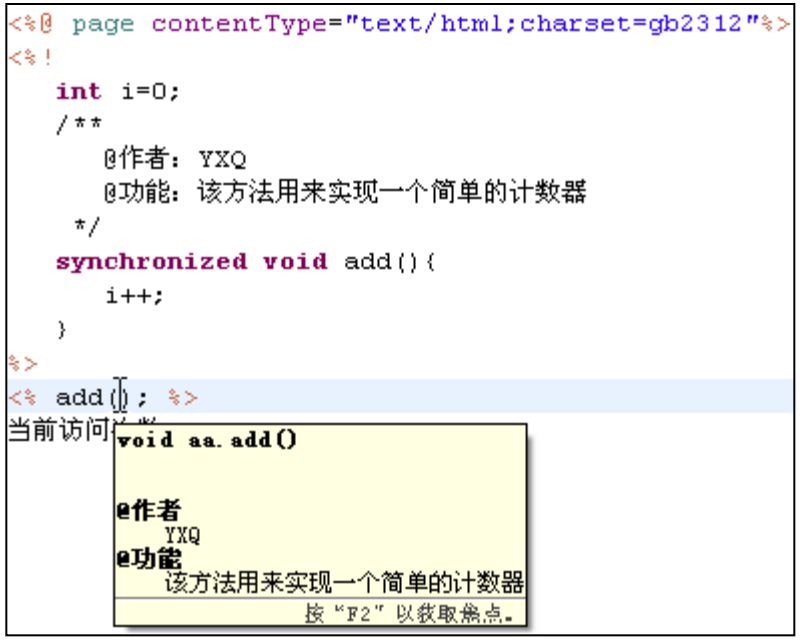
3.4.4 脚本程序（Scriptlet）中的注释

在Eclipse开发工具中向创建的JSP文件输入下面的代码，然后将鼠标指针移动到指定的代码上，将会出现提示信息。

```
<%!
  int i=0;
  /**
    @作者：YXQ
    @功能：该方法用来实现一个简单的计数器
   */
  synchronized void add(){
            i++;
  }
 %>
<% add(); %>
当前访问次数：<%=i%>
```

3.4.4 脚本程序（Scriptlet）中的注释

将鼠标指针移动到<% add(); %>代码上，将出现下图如图所示的提示信息。



3.5 动 作 标 识

在JSP中提供了一系列的使用XML语法写成的动作标识，这些标识可用来实现特殊的功能，例如请求的转发、在当前页中包含其他文件、在页面中创建一个JavaBean实例等。

动作标识是在请求处理阶段按照在页面中出现的顺序被执行的，只有它们被执行的时候才会去实现自己所具有的功能。这与指令标识是不同的，因为在JSP页面被执行时首先进入翻译阶段，程序会先查找页面中的指令标识并将它们转换成Servlet，所以这些指令标识会首先被执行，从而设置了整个的JSP页面。

动作标识通用的使用格式如下：

3.5 动 作 标 识

```
<动作标识名称 属性1="值1" 属性2="值2"…/>

或

<动作标识名称 属性1="值1" 属性2="值2" …>
        <子动作 属性1="值1" 属性2="值2" …/>
</动作标识名称>
```

在JSP中提供的常用的标准动作标识有：

3.5 动 作 标 识

- <jsp:include> √
- <jsp:forward> √
- <jsp:useBean> √
- <jsp:setProperty> √
- <jsp:getProperty> √
- <jsp:fallback> √
- <jsp:plugin> √

<jsp:include>

<jsp:include>动作标识用于向当前的页面中包含其他的文件，这个文件可以是动态文件也可以是静态文件。该标识的使用格式如下：

```
<jsp:include page="被包含文件的路径" flush="true|false"/>
```

或者向被包含的动态页面中传递参数：

```
<jsp:include page="被包含文件的路径" flush="true|false">
        <jsp:param name="参数名称" valude="参数值"/>
</jsp:include>
```

page属性：该属性指定了被包含文件的路径，其值可以是一个代表了相对路径的表达式。当路径是以“/”开头时，则按照当前应用的路径查找这个文件；如果路径是以文件名或目录名称开头，那么将按照当前的路径来查找被包含的文件。

<jsp:include>

flush属性：表示当输出缓冲区满时，是否清空缓冲区。该属性值为boolean型，默认值为false，通常情况下设为true。

<jsp:param>子标识可以向被包含的动态页面中传递参数。

<jsp:include>标识对包含的动态文件和静态文件的处理方式是不同的。如果被包含的是静态的文件，则页面执行后，在使用了该标识的位置处将会输出这个文件的内容。如果<jsp:include>标识包含的是一个动态的文件，那么JSP编译器将编译并执行这个文件。不能通过文件的名称来判断该文件是静态的还是动态的，<jsp:include>标识会识别出文件的类型。

<jsp:include>

<jsp:include>动作标识与include指令都可用来包含文件，下面来介绍它们之间存在的差异。

1．属性

include指令通过file属性来指定被包含的页面，include指令将file属性值看作一个实际存在的文件的路径，所以该属性不支持任何表达式。若在file属性值中应用JSP表达式，则会抛出异常，如下面的代码：

```
<%  String path="logon.jsp";  %>
<%@ include file="<%=path%>" %>
```

该用法将抛出下面的异常：

File ″/<%=path%>″ not found

<jsp:include>

<jsp:include>动作标识通过page属性来指定被包含的页面，该属性支持JSP表达式。

2．处理方式

使用include指令被包含的文件，它的内容会原封不动地插入到包含页中使用该指令的位置，然后JSP编译器再对这个合成的文件进行翻译。所以在一个JSP页面中使用include指令来包含另外一个JSP页面，最终编译后的文件只有一个。

使用<jsp:include>动作标识包含文件时，当该标识被执行时，程序会将请求转发到（注意是转发，而不是请求重定向）被包含的页面，并将执行结果输出到浏览器中，然后返回包含页继续执行后面的代码。因为服务器执行的是两个文件，所以JSP编译器会分别对这两个文件进行编译。

<jsp:include>

3．包含方式

使用include指令包含文件，最终服务器执行的是将两个文件合成后由JSP编译器编译成的一个Class文件，所以被包含文件的内容应是固定不变的，若改变了被包含的文件，则主文件的代码就发生了改变，因此服务器会重新编译主文件。include指令的这种包含过程称为静态包含。

使用<jsp:include>动作标识通常是来包含那些经常需要改动的文件。此时服务器执行的是两个文件，被包含文件的改动不会影响到主文件，因此服务器不会对主文件重新编译，而只需重新编译被包含的文件即可。而对被包含文件的编译是在执行时才进行的，也就是说，只有当<jsp:include>动作标识被执行时，使用该识包含的目标文件才会被编译，否则被包含的文件不会被编译，所以这种包含过程称为动态包含。

<jsp:include>

4．对被包含文件的约定

使用include指令包含文件时，对被包含文件有约定。

【例3-9】 通过include指令包含文件

【例3-10】 通过include动作标识包含文件

技巧：如果要在JSP页面中要显示大量的文本文字，可以将文字写入静态文件中（如记事本），然后通过include指令或动作标识包含进来。

<jsp:forward>

<jsp:forward>动作标识用来将请求转发到另外一个JSP、HTML或相关的资源文件中。当该标识被执行后，当前的页面将不再被执行，而是去执行该标识指定的目标页面。
该标识使用的格式如下：

```
<jsp:forward page="文件路径 | 表示路径的表达式"/>
```

如果转发的目标是一个动态文件，还可以向该文件中传递参数，使用格式如下：

```
<jsp:include page="被包含文件的路径" flush="true|false">
        <jsp:param name="参数名称" valude="参数值"/>
</jsp:include>
```

<jsp:forward>

page属性：该属性指定了目标文件的路径。如果该值是以“/”开头，表示在当前应用的根目录下查找文件，否则就在当前路径下查找目标文件。请求被转向到的目标文件必须是内部的资源，即当前应用中的资源。

如果想通过forward动作转发到应用外部的文件中，例如，当前应用为A，在根目录下的index.jsp页面中存在下面的代码用来将请求转发到应用B中的logon.jsp页面。

```
<jsp:forward page="http://localhost:8080/B/logon.jsp"/>
```

那么将出现下面的错误提示：

The requested resource (/http://localhost:8080/B/logon.jsp) is not available

<jsp:forward>

仔细观察可以看到，错误提示中的路径前自动加入了一个“/”，这是因为index.jsp页面在应用A的根目录下，当forward标识被执行时，会在该目录下来查找page属性指定的目标文件，所以会提示资源不存在的信息。

<jsp:param>子标识用来向动态的目标文件中传递参数。

这里重点提示一下，<jsp:forward>标识实现的是请求的转发操作，而不是请求重定向。它们之间的一个区别就是：进行请求转发时，存储在request对象中的信息会被保留并被带到目标页面中；而请求重定向是重新生成一个request请求，然后将该请求重定向到指定的URL，所以事先存储在request对象中的信息都不存在了。

<jsp:useBean>

通过应用<jsp:useBean>动作标识可以在JSP页面中创建一个Bean实例，并且通过属性的设置可以将该实例存储到JSP中的指定范围内。如果在指定的范围内已经存在了指定的Bean实例，那么将使用这个实例，而不会重新创建。通过<jsp:useBean>标识创建的Bean实例可以在Scriptlet中应用。该标识的使用格式如下：

<jsp:useBean>

```
<jsp:useBean
          id="变量名"
    scope="page|request|session|application"
    {
          class="package.className"|
        type="数据类型"|
        class="package.className" type="数据类型"|
        beanName="package.className" type="数据类型"
    }
/>
<jsp:setProperty name="变量名" property="*"/>
```

也可以在标识体内嵌入子标识或其他内容：

<jsp:useBean>

```
<jsp:useBean id="变量名"
scope="page|request|session|application" …>
        <jsp:setProperty name="变量名" property="*"/>
</jsp:useBean>
```

这两种使用方法是有区别的。在页面中应用<jsp:useBean>标识创建一个Bean时，如果该Bean是第一次被实例化，那么对于<jsp:useBean>标识的第二种使用格式，标识体内的内容会被执行，若已经存在了指定的Bean实例，则标识体内的内容就不再被执行了。而对于第一种使用格式，无论在指定的范围内是否已经存在一个指定的Bean实例，<jsp:useBean>标识后面的内容都会被执行。

<jsp:useBean>

下面将对<jsp:useBean>标识中各属性的用法进行详细介绍。

1. id属性

该属性指定一个变量，在所定义的范围内或Scriptlet中将使用该变量来对所创建的Bean实例进行引用。该变量必须符合Java中变量的命名规则。

<jsp:useBean>

2. type=″数据类型″

type属性用于设置由id属性指定的变量的类型。type属性可以指定要创建实例的类的本身、类的父类或者是一个接口。

使用type属性来设置变量类型的使用格式如下：

```
<jsp:useBean id="us" type="com.Bean.UserInfo" scope="session"/>
```

如果在session范围内，已经存在了名为“us”的实例，则将该实例转换为type属性指定的UserInfo类型（必须是合法的类型转换）并赋值给id属性指定的变量；若指定的实例不存在将抛出“bean us not found within scope”异常。

<jsp:useBean>

3. scope属性

该属性指定了所创建Bean实例的存取范围，省略该属性时的值为page。<jsp:useBean>标识被执行时，首先会在scope属性指定的范围来查找指定的Bean实例，如果该实例已经存在，则引用这个Bean，否则重新创建，并将其存储在scope属性指定的范围内。scope属性具有的可选值如下。

page：指定了所创建的Bean实例只能够在当前的JSP文件中使用，包括在通过include指令静态包含的页面中有效。

request：指定了所创建的Bean实例可以在请求范围内进行存取。在请求被转发至的目标页面中可通过request对象的getAttribute("id属性值")方法获取创建的Bean实例。

<jsp:useBean>

一个请求的生命周期是从客户端向服务器发出一个请求到服务器响应这个请求给用户后结束，所以请求结束后，存储在其中的Bean的实例也就失效了。

session：指定了所创建的Bean实例的有效范围为session。session是当用户访问Web应用时，服务器为用户创建的一个对象，服务器通过session的ID值来区分其他的用户。针对某一个用户而言，在该范围中的对象可被多个页面共享。

注意：可以使用session对象的getAttribute("id属性值")方法获取存储在session中的Bean实例，也可以使用session对象的getValue("id属性值")来获取，但该方法不建议使用。

<jsp:useBean>

application：该值指定了所创建的Bean实例的有效范围从服务器启动开始到服务器关闭结束。application对象是在服务器启动时创建的，它被多个用户共享。所以访问该application对象的所有用户共享存储于该对象中的Bean实例。

注意：可以使用application对象的getAttribute("id属性值")方法获取存在于application中的Bean实例。

4. class="package.className"

class属性指定了一个完整的类名，其中package表示类包的名字，className表示类的Class文件名称。通过class属性指定的类不能是抽象的，它必须具有公共的、没有参数的构造方法。在没有设置type属性时，必须设置class属性。

<jsp:useBean>

使用class属性定位一个类的使用格式如下：

```
<jsp:useBean id="us" class="com.Bean.UserInfo"
scope="session"/>
```

程序首先会在session范围中来查找是否存在名为“us”的UserInfo类的实例，如果不存在，那么会通过new操作符实例化UserInfo类来获取一个实例，并以“us”为实例名称存储到session范围内。

5. class=″package.className″ type=″数据类型″

class属性与type属性可以指定同一个类，在<jsp:useBean>标识中class属性与type属性一起使用时的格式如下：

<jsp:useBean>

```
<jsp:useBean id="us" class="com.Bean.UserInfo"
type="com.Bean.UserBase" scope="session"/>
```

这里假设UserBase类为UserInfo类的父类。该标识被执行时，程序首先创建了一个以type属性的值为类型，以id属性值为名称的变量us，并赋值为null；然后在session范围内来查找这个名为“us”的Bean实例，如果存在，则将其转换为type属性指定的UserBase类型（类型转换必须是合法的）并赋值给变量us；如果实例不存在，那么将通过new操作符来实例化一个UserInfo类的实例并赋值给变量us，最后将us变量储在session范围内。

6. beanName=″package.className″ type=″数据类型″

<jsp:useBean>

beanName属性与type属性可以指定同一个类，在<jsp:useBean>标识中beanName属性与type属性一起使用时的格式如下：

```
<jsp:useBean id="us" beanName="com.Bean.UserInfo"
type="com.Bean.UserBase"/>
```

这里假设UserBase类为UserInfo类的父类。该标识被执行时，程序首先创建了一个以type属性的值为类型，以id属性值为名称的变量us，并赋值为null；然后在session范围内来查找这个名为“us”的Bean实例，如果存在，则将其转换为type属性指定的UserBase类型（类型转换必须是合法的）并赋值给变量us；如果实例不存在，那么将通过instantiate()方法从UserInfo类中实例化一个类并将其转换成UserBase类型后赋值给变量us，最后将变量us存储在session范围内。

<jsp:useBean>

通常情况下应用<jsp:useBean>标识的格式如下：

```
<jsp:useBean id="变量名" class="package.className"/>
```

如果想在多个页面中共享这个Bean实例，可将scope属性设置为session。

在页面中使用<jsp:useBean>标识来实例化一个Bean实例后，可以通过<jsp:setProperty>属性来设置或修改该Bean中的属性，或者通过<jsp:getProperty>标识来读取该Bean中指定的属性。

<jsp:setProperty>

<jsp:setProperty>标识通常情况下与<jsp:useBean>标识一起使用，它将调用Bean中的setXxx()方法将请求中的参数赋值给由<jsp:useBean>标识创建的JavaBean中对应的简单属性或索引属性。该标识的使用格式如下：

```
<jsp:setProperty
        name="Bean实例名"
    {
     property="*" |
        property="propertyName" |
        property="propertyName" param="parameterName" |
        property="propertyName" value="值"
    }/>
```

<jsp:setProperty>

1. name属性

name属性用来指定一个存在JSP中某个范围中的Bean实例。<jsp:setProperty>标识将会按照page、request、session和application的顺序来查找这个Bean实例，直到第一个实例被找到。若任何范围内不存在这个Bean实例，则会抛出异常。

2. property="*"

property属性取值为“*”时，则request请求中所有参数的值将被一一赋给Bean中与参数具有相同名字的属性。如果请求中存在值为空的参数，那么Bean中对应的属性将不会被赋值为Null；如果Bean中存在一个属性，但请求中没有与之对应的参数，那么该属性同样不会被赋值为Null，

<jsp:setProperty>

在这两种情况下的Bean属性都会保留原来或默认的值。

该种使用方法要求请求中参数的名称和类型必须与Bean中属性的名称和类型一致。但由于通过表单传递的参数都是String类型的，所以JSP会自动将这些参数转换为Bean中对应属性的类型。下表给出了JSP自动将String类型转换为其他类型时所调用的方法。

<jsp:setProperty>

其 他 类 型	转 换 方法
boolean	java.lang.Boolean.valueOf(String).booleanValue()
Boolean	java.lang.Boolean.valueOf(String)
byte	java.lang.Byte.valueOf(String).byteValue()
Byte	java.lang.Byte.valueOf(String)
double	java.lang.Double.valueOf(String).doubleValue()
Double	java.lang.Double.valueOf(String)
int	java.lang.Integer.valueOf(String).intValue()
Integer	java.lang.Integer.valueOf(String)
float	java.lang.Float.valueOf(String).floatValue();
Float	java.lang.Float.valueOf(String)
long	java.lang.Long.valueOf(String).longValue()
Long	java.lang.Long.valueOf(String)

<jsp:setProperty>

3. property="propertyName"

property属性取值为Bean中的属性时，则只会将request请求中与该Bean属性同名的一个参数的值赋给这个Bean属性。更进一步讲，如果property属性指定的Bean属性为userName，那么指定Bean中必须存在setUserName()方法，否则会抛出类似于下面的异常：

Cannot find any information on property 'userName' in a bean of type 'com.Bean.UserInfo'

在此基础上，如果请求中没有与userName同名的参数，则该Bean属性会保留原来或默认的值，而不会被赋值为Null。

与将property属性赋值为“*”一样，当请求中参数的类型与Bean中属性类型不一致时，JSP会自动进行转换。

<jsp:setProperty>

4. property="propertyName" param="parameterName"

param属性指定一个request请求中的参数，property属性指定Bean中的某个属性。该种使用方法允许将请求中的参数赋值给Bean中与该参数不同名的属性。如果param属性指定参数的值为空，那么由property属性指定的Bean属性会保留原来或默认的值而不会被赋为Null。

5. property="propertyName" value="值"

其中，value属性指定的值可以是一个字符串数值或表示一个具体值的JSP表达式或EL表达式。该值将被赋给property属性指定的Bean属性。

<jsp:setProperty>

当value属性指定的是一个字符串时，如果指定的Bean属性与其类型不一致时，则会根据表3-3中的方法将该字符串值自动转换成对应的类型。

当value属性指定的是一个表达式时，那么该表达式所表示的值的类型必须与property属性指定的Bean属性一致，否则会抛出“argument type mismatch”异常。

通常<jsp:setProperty>标识与<jsp:useBean>标识一起使用，但这并不是绝对的，应用如下的方法同样可以将请求中的参数值赋给JavaBean中的属性。

【例3-11】 <jsp:setProperty>标识的使用

<jsp:getProperty>

<jsp:getProperty>属性用来从指定的Bean中读取指定的属性值，并输出到页面中。该Bean必须具有getXxx()方法。

<jsp:getProperty>标识的使用格式如下：

```
<jsp:getProperty name="Bean实例名" property="propertyName"/>
```

name属性：name属性用来指定一个存在某JSP范围中的Bean实例。<jsp:getProperty>标识将会按照page、request、session和application的顺序来查找这个Bean实例，直到第一个实例被找到。若任何范围内不存在这个Bean实例则会抛出“Attempted a bean operation on a null object”异常。

<jsp:getProperty>

property属性：该属性指定了要获取由name属性指定的Bean中的哪个属性的值。若它指定的值为“userName”，那么Bean中必须存在getUserName()方法，否则会抛出下面的异常：Cannot find any information on property 'userName' in a bean of type '此处为类名'

如果指定Bean中的属性是一个对象，那么该对象的toString()方法被调用，并输出执行结果。

<jsp:fallback>

<jsp:fallback>是<jsp:plugin>的子标识，当使用<jsp:plugin>标识加载Java小应用程序或JavaBean失败时，可通过<jsp:fallback>标识向用户输出提示信息。该标识的使用格式如下：

```
<jsp:plugin type="applet" code="com.source.MyApplet.class"
codebase=".">
        …
        <jsp:fallback>加载Java Applet小程序失败!</jsp:fallback>
        …
</jsp:plugin>
```

<jsp:plugin>

使用<jsp:plugin>标识可以在页面中插入Java Applet小程序或JavaBean，它们能够在客户端运行。该标识会根据客户端浏览器的版本转换成<object>或<embed>HTML元素。该标识的使用格式如下：

```
<jsp:plugin
    type="applet | bean"    code=""    codebase=""    [name=""]
   [archive=""]    [align=""]    [height=""]    [width=""]
   [hspace=""]    [vspace=""]    [jreversion=""]    [nspluginurl=""]
    [iepluginurl=""]
    [<jsp:params>
      <jsp:param name="parameterName" value="{parameterValue | <%=expression
%>}"/>
     </jsp:params>]
    [<jsp:fallback>加载失败提示信息</jsp:fallback>]>
</jsp:plugin>
```

<jsp:plugin>

<jsp:plugin>标识中的各属性的简要介绍如下表所示。

属　性	说　明
type	该属性指定了所要加载的插件对象的类型，可选值为“bean”和“applet”
code	指定了要加载的Java类文件的名称。该名称可包含扩展名和类包名，如“com.applet.MyApplet.class”
codebase	默认值为当前访问的JSP页面的路径，该属性用来指定code属性指定的Java类文件所在的路径
name	指定了加载的Applet或Bean的名称
archive	指定预先加载的存档文件的路径，多个路径可用逗号进行分隔
align	加载的插件对象在页面中显示时的对齐方式。可选值为“bottom”、“top”、“middle”、“left”和“right”
height和width	加载的插件对象在页面中显示时的高度和宽度，单位为像素。这两个属性值支持JSP表达式或EL表达式
hspace和vspace	加载的Applet或Bean在屏幕或单元格中所留出的空间大小，hspace表示左右，vspace表示上下，它们不支持任何表达式
jreversion	在浏览器中执行Applet或Bean时所需的Java Runtime Environment(JRE)的版本，默认值为1.1
nspluginurl和iepluginurl	分别指定了Netscape Navigator用户和Internet Explorer用户能够使用的JRE的下载地址
<jsp:params>	在该标识中可包含多个<jsp:param>标识，用来向Applet或Bean中传递参数
<jsp:fallback>	当加载Java类文件失败时，用来显示给用户提示信息

<jsp:plugin>

下面对<jsp:plugin>标识中重要属性的用法进行详细的介绍。

1．type属性

type属性指定了所要加载的插件对象的类型，一般为Java Applet小程序或JavaBean类。可选值为“applet”和“bean”。该属性没有缺省值，必须设置可选值中的一个，否则会抛出异常。

2．code属性

code属性指定了加载的Java类的文件名称。该名称可包含扩展名和类包名，如“com.applet.MyApplet.class”。

<jsp:plugin>

3．codebase属性

默认值为当前访问的JSP页面的路径，该属性用来指定code属性指定的Java类文件所在的目录。注意，当程序执行到<jsp:plugin>标识加载插件时，容器是从当前引用该标识来加载插件的JSP页面所在的目录开始，并根据codebase属性和code属性指定的值来查找指定的插件。

（1）如果codebase属性值为“/”或“”，那么容器将按照“协议+主机+code属性值”的路径来查找插件对象。

【例3-12】 codebase属性的使用1

<jsp:plugin>

（2）如果codebase属性值为“.”，那么容器将按照当前访问的JSP文件的目录为基础路径开始查找插件对象。查找的路径为“协议+主机+当前访问的JSP文件目录+code属性值指定的路径”。

【例3-13】 codebase属性的使用2

（3）如果codebase属性值以“. /”开头，那么容器将按照当前访问的JSP页面所在的目录加上codebase属性指定的目录为基础路径开始查找插件对象。

【例3-14】 codebase属性的使用3

<jsp:plugin>

（4）如果codebase属性是以“.. /”开头，那么容器将按照当前访问的JSP页面所在目录的上一级目录加上codebase属性指定的目录为基础路径开始查找插件对象。

4. nspluginurl和iepluginurl属性

这两个属性分别指定了Netscape Navigator用户和Internet Explorer用户能够使用的JRE的下载地址。使用方法如下：

```
<jsp:plugin type="applet" code="com.applet.MyApplet.class"
codebase="./applet"   iepluginurl="http://localhost:8080">
        <jsp:fallback>加载Java Applet小程序失败!</jsp:fallback>
</jsp:plugin>
```

<jsp:plugin>

若当前的Internet Explorer用户没有安装JRE，则访问包含下面代码的JSP页面后浏览器自动弹出如下图所示的提示。

弹出该提示的前提是需要在浏览器中进行相应的安全设置。打开浏览器中的“工具”/“Internet选项”子菜单，然后选择“安全”选项卡并单击“自定义级别”按钮，在弹出的“安全设置”对话框中进行如下图所示的设置。

<jsp:plugin>

5. <jsp:param>子标识

在该标识内可包含多个<jsp:param>子标识，每个<jsp:param>标识指定一个向要加载的Java Applet或Bean中传递的参数。它们在<jsp:plugin>标识中的使用格式如下：

<jsp:plugin>

```
<jsp:plugin type="applet" code="com.applet.MyApplet.class"
codebase="./applet">
        <jsp:params>
                <jsp:param name="username" value="YXQ"/>
                <jsp:param name="userpwd" value="123"/>
        </jsp:params>
        <jsp:fallback>加载Applet失败！</jsp:fallback>
</jsp:plugin>
```

【例3-15】 <jsp:param,子标签的使用>