第2章 JSP 基础语法知识

本章要点

1. JSP 语法声明、表达式的编写。
2. JSP+JavaBean 编程。
3. JSP 的动作指令。

学习目标

1. 掌握 JSP 语法注释声明。
2. 掌握 JSP 程序开发模式。
3. 掌握 JSP 的指令。
4. 掌握 JSP 的动作。

2.1 JSP 语法注释声明

JSP 语法缺少不了注释声明，注释是为了能让他人看懂代码。JSP 的声明是表示“这是 JSP 语言”的关键。代码段与表达式共同组成了 JSP 的程序。

2.1.1 语法注释

在 JSP 页面中可以使用多种注释，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。本小节将介绍 JSP 中的各种注释。

1. HTML 中的注释

JSP 文件是由 HTML 标记和嵌入的 Java 程序段组成的，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 2-1】HTML 中的注释：

```
<!--欢迎提示信息!-->
<table><tr><td>欢迎访问！</td></tr></table>
```

使用该方法注释的内容在客户端浏览器中是看不到的，但可以通过查看 HTML 源代码看到这些注释内容。

访问该页面后，将会在客户端浏览器中输出以下内容：

```
欢迎访问！
```

通过查看 HTML 源代码，将会看到如下内容：

```
<!--欢迎提示信息！-->
<table><tr><td>欢迎访问！</td><tr></table>
```

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 2-2】使用带有 JSP 表达式的注释：

```
<% String name="XYQ"; %>
<!--当前用户：<%=name%>-->
<table><tr><td>欢迎登录：<%=name%></td></tr></table>
```

访问该页面后，将会在客户端浏览器中输出以下内容：

```
欢迎登录：XYQ
```

通过查看 HTML 源代码，将会看到以下内容：

```
<!--当前用户：<%=name%>-->
<table><tr><td>欢迎登录：XYQ</td></tr></table>
```

3. 隐藏注释

前面已经介绍了如何使用 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。下面介绍一种隐藏注释，注释格式如下：

```
<%--注释内容--%>
```

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

【例 2-3】使用隐藏注释：

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

访问该页面后，将会在客户端浏览器中输出以下内容：

```
当前时间为：2017-3-19  15:27:20
```

通过查看 HTML 源代码，将会看到以下内容：

```
<table>
   <tr><td>当前时间为：2017-3-19 15:27:20</td></tr>
</table>
```

4. 脚本程序(Scriptlet)中的注释

脚本程序中包含的是一段 Java 代码，所以在脚本程序中的注释与在 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
```

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注释的内容也是不可见的，并且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。

【例 2-4】JSP 文件中包含以下代码：

```
<%
int count = 6;  //定义一个计数变量
%>
计数变量 count 的当前值为：<%=count%>
```

访问该页面后，将会在客户端浏览器中输出以下内容：

```
计数变量 count 的当前值为：6
```

通过查看 HTML 源代码，将会看到以下内容：

```
计数变量 count 的当前值为：6
```

因为服务器不会对注释的内容进行处理，所以可以通过该注释暂时删除某一行代码。例如下面的代码。

【例 2-5】使用单行注释暂时删除一行代码：

```
<%
String name = "XYQ";
//name = "XYQ2017";
%>
用户名：<%=name%>
```

包含上述代码的 JSP 文件被执行后，将输出如下结果：

```
用户名：XYQ
```

(2) 多行注释。

多行注释是通过“/*”与“*/”符号进行标记的，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
*/
```

为了程序界面的美观，开发人员习惯在每行注释内容的前面添加一个“*”号，构成如下所示的注释格式：

```
/*
* 注释内容 1
* 注释内容 2
*/
```

与单行注释一样，在“/*”与“*/”之间注释的所有内容，即使是 JSP 表达式或其他脚本程序，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 2-6】在 JSP 文件中包含以下代码：

```
<%@ page contentType="text/html;charset=UTF-8"%>
<%
   String state = "0";
   /* if(state.equals("0")) {   //equals()方法用来判断两个对象是否相等
          state = "主版";
%>
          将变量 state 赋值为“主版”。<br>
<%
      }
   */
%>
变量 state 的值为：<%=state%>
```

包含上述代码的 JSP 文件被执行后，将输出如图 2-1 所示的结果。
若去掉代码中的“/*”和“*/”符号，则将输出如图 2-2 所示的结果。

图 2-1　多行注释(一)

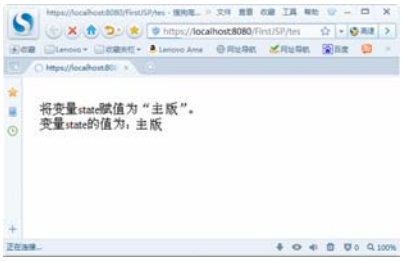

图 2-2　多行注释(二)

(3) 文档注释。
该种注释会被 Javadoc 文档工具在生成文档时读取，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
*/
```

该注释方法与上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，对于被注释的所有内容，服务器都不会做任何处理。

【例 2-7】在 Eclipse 开发工具中，在创建的 JSP 文件中输入以下代码：

```
<%!
    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%>
```

将鼠标指针移动到<% add(); %>代码上，将出现如图 2-3 所示的提示信息。

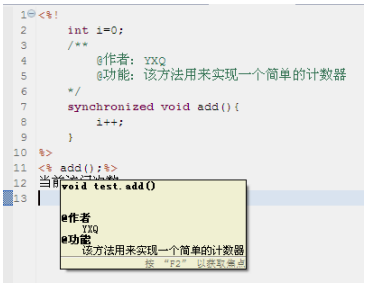

图 2-3　提示文档注释

2.1.2 声明

在 JSP 页面中可以声明变量、方法和类，其声明格式如下：

```
<%!声明变量、方法和类的代码 %>
```

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的语法是一样的。

1. 声明变量

在“<%!”和“%>”标记之间声明变量，即在“<%!”和“%>”之间放置 Java 的变量声明语句。变量的类型可以是 Java 语言允许的任何数据类型。我们将这些变量称为 JSP 页面的成员变量。

【例 2-8】声明变量：

```
<%!
int x, y=100, z;
String tom=null, jery="Love JSP";
Date date;
%>
```

这里，“<%!”和“%>”之间声明的变量在整个 JSP 页面内都有效，因为 JSP 引擎将 JSP 页面转译成 Java 文件时，将这些变量作为类的成员变量，这些变量的内存空间直到服务器关闭才被释放。当多个客户请求一个 JSP 页面时，JSP 引擎为每个客户启动一个线程，这些线程由 JSP 引擎服务器来管理。这些线程共享 JSP 页面的成员变量，因此任何一个用户对 JSP 页面成员变量操作的结果，都会影响到其他用户。

2. 方法声明

在“<%!”和“%>”标记之间声明的方法，在整个 JSP 页面有效，但是，方法内定义的变量只在方法内有效。

【例 2-9】声明方法：

```
<%@ page contentType="text/html; charset=utf-8" %>
<%!
int num = 0;                    //声明一个计数变量
synchronized void add() {       //该方法实现访问次数的累加操作
   num++;
}
%>
<% add(); %>
<html>
    <body><center>您是第<%=num%>位访问该页面的游客！</center></body>
</html>
```

运行结果如图 2-4 所示。

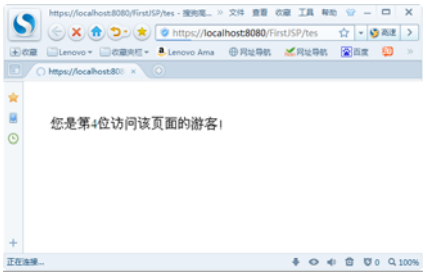

图 2-4　使用方法的声明

示例中声明了一个 num 变量和 add()方法。add()方法对 num 变量进行累加操作，synchronized 修饰符可以使多个同时访问 add()方法的线程排队调用。

当第一个用户访问该页面后，变量 num 被初始化，服务器执行<% add(); %>小脚本程序，从而 add()方法被调用，num 变为 1。当第二个用户访问时，变量 num 不再被重新初始化，而使用前一个用户访问后的 num 值，之后调用 add()方法，num 值变为 2。

3. 声明类

可以在“<%!”和“%>”之间声明一个类。该类在 JSP 页面内有效，即在 JSP 页面的 Java 程序段部分可以使用该类创建对象。下例中，定义了一个 Circle 类，该类的对象负责求圆的面积。当客户向服务器提交圆的半径后，该对象计算圆的面积。

【例 2-10】使用类的声明：

```
<%@ page contentType="text/html; charset=utf-8"%>
<HTML>
<BODY>
<FONT size="4">
<p>请输入圆的半径：<BR>
<FORM action="" method=get name=form>
   <INPUT type="text" name="cat" value="1">
   <INPUT TYPE="submit" value="送出" name=submit>
</FORM>
<%!
public class Circle
{
   double r;
   Circle(double r)
   {
      this.r = r;
   }
   double 求面积()
   {
      return Math.PI*r*r;
   }
}
%>
<%
String str = request.getParameter("cat");
double r;
```

```
if(str != null)
{
    r = Double.parseDouble(str);
}
else
{
    r = 1;
}
Circle circle = new Circle(r);
%>
<p>圆的面积是：<%=circle.求面积()%>
</FONT>
</BODY>
</HTML>
```

运行结果如图 2-5 所示。

图 2-5　使用类声明

2.1.3　代码段

JSP 允许在“<%”和“%>”之间插入 Java 程序段。一个 JSP 页面可以有许多程序段，这些程序段将被 JSP 引擎按顺序执行。

在一个程序段中声明的变量叫作 JSP 页面的局部变量，它们在 JSP 页面内的相关程序段以及表达式内都有效。这是因为 JSP 引擎将 JSP 页面转译成 Java 文件时，将各个程序段的这些变量作为类中某个方法的变量，即局部变量。

利用程序段的这个性质，有时可以将一个程序段分割成几个更小的程序段，然后在这些小的程序段之间再插入 JSP 页面的一些其他标记元素。

当程序段被调用执行时，会为这些变量分配内存空间，当所有的程序段调用完毕后，这些变量即可释放所占的内存。

当多个客户请求一个 JSP 页面时，JSP 引擎为每个客户启动一个线程，一个客户的局部变量和另一个客户的局部变量会分配不同的内存空间。因此，一个客户对 JSP 页面局部变量操作的结果，不会影响到其他客户的这个局部变量。

【例 2-11】下面的程序段可以计算 1 到 100 的和：

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

```
<BODY>
<FONT size="10">
<%!
long continueSum(int n)
{
    int sum = 0;
    for(int i=1; i<=n; i++)
    {
        sum = sum + i;
    }
    return sum;
}
%>

<p> 1 到 100 的连续和：
<br>
<%
long sum;
sum = continueSum(100);
out.print(" " + sum);
%>

</FONT>
</BODY>
</HTML>
```

运行结果如图 2-6 所示。

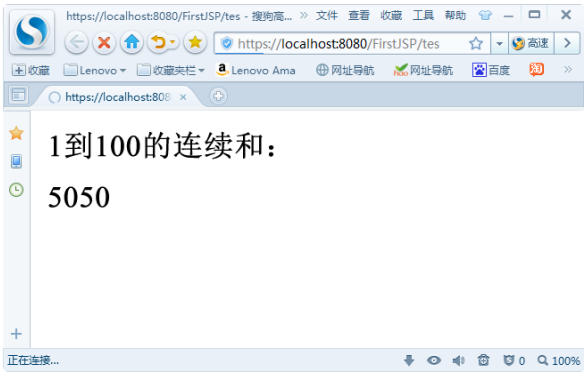

图 2-6　在 JSP 中使用 Java 代码段

2.1.4　表达式

表达式用于在页面中输出信息，其使用格式如下：

```
<%=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，变成了 out.print()方法。所以，JSP 表达式与 JSP 页面中嵌入小脚本程序中的 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 表达式可以应用于以下几种情况。

(1) 向页面输出内容。

【例 2-12】向页面输出内容：

```
<% String name = "www.123.com"; %>
用户名：<%=name%>
```

上述代码将生成如下运行结果：

```
用户名：www.123.com
```

(2) 生成动态的链接地址。

【例 2-13】生成动态的链接地址：

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

上述代码将生成如下的 HTML 代码：

```
<a href="welcome.jsp">链接到 welcome.jsp</a>
```

(3) 动态指定 Form 表单处理页面。

【例 2-14】动态指定 Form 表单处理页面：

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

上述代码将生成如下 HTML 代码：

```
<form action="logon.jsp"></form>
```

(4) 为通过循环语句生成的元素命名。

【例 2-15】为通过循环语句生成的元素命名：

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

上述代码将生成如下 HTML 代码：

```
file1:<input type="text" name="file1"><br>
file2:<input type="text" name="file2"><br>
```

2.2 JSP 程序开发模式

JSP 程序开发模式包括 JSP 编程、JSP+JavaBean 编程、JSP+JavaBean+Servlet 编程、MVC 模式。本节除讲解以上几种开发模式外，还将讲解在运行 JSP 时常见的出错处理方法。

2.2.1 单纯的 JSP 编程

在 JSP 编程模式下，通过应用 JSP 中的脚本标志，可以直接在 JSP 页面中实现各种功能。虽然这种模式很容易实现，但是，其缺点也非常明显。因为将大部分的 Java 代码与 HTML 代码混淆在一起，会给程序的维护和调试带来很多困难，而且难以理清完整的程序结构。

这就好比规划管理一个大型企业，如果将负责不同任务的所有员工都安排在一起工作，势必会造成公司秩序混乱、不易管理等许多隐患。所以说，单纯的 JSP 页面编程模式是无法应用到大型、中型甚至小型的 JSP Web 应用程序开发中的。

2.2.2 JSP+JavaBean 编程

JSP+JavaBean 编程模式是 JSP 程序开发经典设计模式之一，适合小型或中型网站的开发。利用 JavaBean 技术，可以很容易地完成一些业务逻辑上的操作，例如数据库的连接、用户登录与注销等。JavaBean 是一个遵循了一定规则的 Java 类，在程序的开发中，将要进行的业务逻辑封装到这个类中，在 JSP 页面中，通过动作标签来调用这个类，从而执行这个业务逻辑。此时的 JSP 除了负责部分流程的控制外，主要用来进行页面的显示，而 JavaBean 则负责业务逻辑的处理。可以看出，JSP+JavaBean 设计模式具有一个比较清晰的程序结构，在 JSP 技术的起步阶段，该模式曾被广泛应用。

图 2-7 表示该模式对客户端的请求进行处理的过程，相关的说明如下。

(1) 用户通过客户端浏览器请求服务器。

(2) 服务器接收用户请求后调用 JSP 页面。

(3) 在 JSP 页面中调用 JavaBean。

(4) 在 JavaBean 中连接及操作数据库，或实现其他业务逻辑。

(5) JavaBean 将执行的结果返回 JSP 页面。

(6) 服务器读取 JSP 页面中的内容(将页面中的静态内容与动态内容相结合)。

(7) 服务器将最终的结果返回给客户端浏览器进行显示。

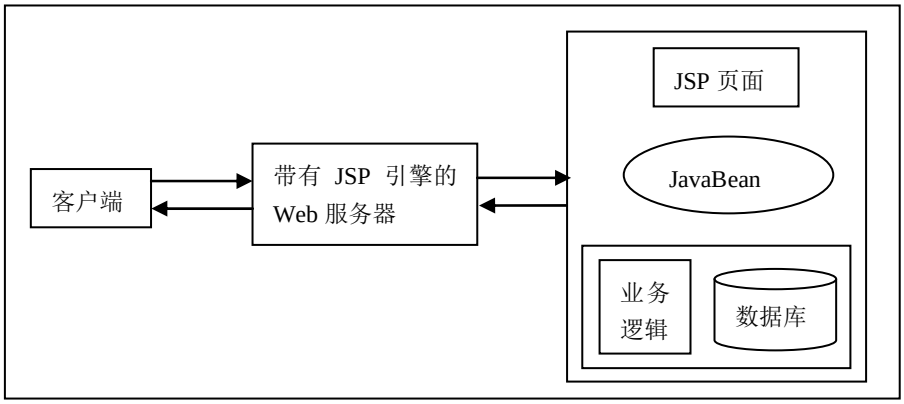

图 2-7 JSP+JavaBean 设计模式

2.2.3 JSP+JavaBean+Servlet 编程

JSP+JavaBean 设计模式虽然已经对网站的业务逻辑和显示页面进行了分离，但这种模式下的 JSP 不但要控制程序中的大部分流程，而且还要负责页面的显示，所以仍然不是一种理想的设计模式。

在 JSP+JavaBean 设计模式的基础上加入 Servlet 来实现程序中的控制层，是一个很好的选择。在这种模式中，由 Servlet 来执行业务逻辑并负责程序的流程控制，JavaBean 组件实现业务逻辑，充当模型的角色，JSP 用于页面的显示。可以看出，这种模式使得程序中的层次关系更明显，各组件的分工也非常明确。图 2-8 表示该模式对客户端的请求进行处理的过程。

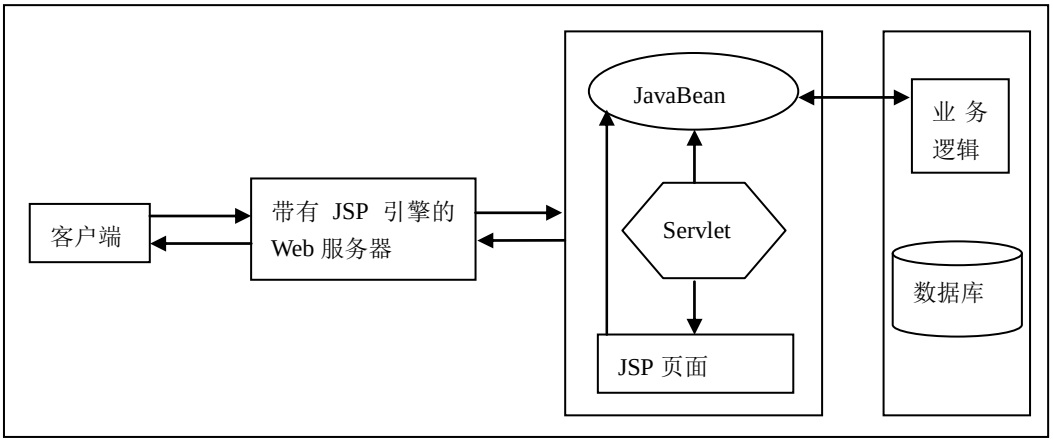

图 2-8 JSP+JavaBean+Servlet 设计模式

图 2-8 所示的模式中，各步骤的说明如下。

(1) 用户通过客户端浏览器请求服务器。

(2) 服务器接收用户请求后调用 Servlet。

(3) Servlet 根据用户请求调用 JavaBean 处理业务。

(4) 在 JavaBean 中连接及操作数据库，或实现其他业务逻辑。

(5) JavaBean 将结果返回 Servlet，在 Servlet 中将结果保存到请求对象中。

(6) 由 Servlet 转发请求到 JSP 页面。

(7) 服务器读取 JSP 页面中的内容(将页面中的静态内容与动态内容结合)。

(8) 服务器将最终的结果返回给客户端浏览器进行显示。

但 JSP+JavaBean+Servlet 模式同样也存在缺点。该模式遵循了 MVC 设计模式，MVC 只是一个抽象的设计概念，它将待开发的应用程序分解为三个独立的部分：模型(Model)、视图(View)和控制器(Controller)。虽然用来实现 MVC 设计模式的技术可能都是相同的，但各公司都有自己的 MVC 架构。也就是说，这些公司用来实现自己的 MVC 架构所应用的技术可能都是 JSP、Servlet 与 JavaBean，但它们的流程及设计却是不同的，所以工程师需要花更多的时间去了解。从项目开发的观点上来说，因为需要设计 MVC 各对象之间的数据交换格式与方法，所以在系统的设计上需要花费更多的时间。

使用 JSP+JavaBean+Servlet 模式进行项目开发时，可以选择一个实现了 MVC 模式的现

成的框架，在此框架的基础上进行开发，能够大大节省开发时间，会取得事半功倍的效果。目前，已有很多可以使用的现成的 MVC 框架，例如 Struts 框架。

2.2.4 MVC 模式

MVC(Model-View-Controller，模型-视图-控制器)是一种程序设计概念，它同时适用于简单的和复杂的程序。使用该模式，可将待开发的应用程序分解为三个独立的部分：模型、视图和控制器。

提出这种设计模式主要是因为应用程序中用来完成任务的代码(模型，也称为“业务逻辑”)通常是程序中相对稳定的部分，并且会被重复使用，而程序与用户进行交互的页面(视图)，却是经常改变的。如果因需要更新页面而不得不对业务逻辑代码进行改动，或者要在不同的模块中应用相同的功能时重复地编写业务逻辑代码，不仅会降低整体程序开发的进程，而且会使程序变得难以维护。因此，将业务逻辑代码与外观呈现分离，将会更容易地根据需求的改变来改进程序。MVC 模式的模型如图 2-9 所示。

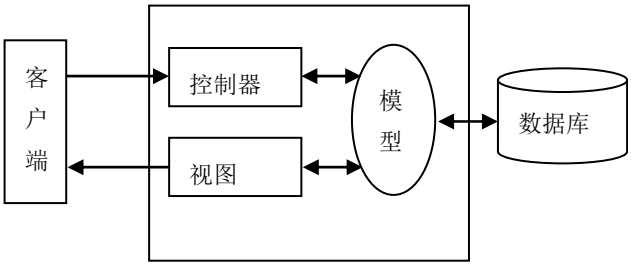

图 2-9　MVC 模式的模型

Model(模型)：MVC 模式中的 Model(模型)指的是业务逻辑的代码，是应用程序中真正用来完成任务的部分。

View(视图)：视图实际上就是程序与用户进行交互的界面，用户可以看到它的存在。视图可以具备一定的功能，并应遵守对其所做的约束。在视图中，不应包含对数据处理的代码，即业务逻辑代码。

Controller(控制器)：控制器主要用于控制用户请求并做出响应。它根据用户的请求，选择模型或修改模型，并决定返回什么样的视图。

2.2.5 运行 JSP 时常见的出错信息及处理

(1) 页面显示 500 错误，错误信息如下：

```
An error occurred at line: 6 in the generated java file
Syntax error on token ";", import expected after this token
```

错误原因见如下代码：

```
<%@ page langue="java" import="java.utli.*; java.text,*"
  pageEncoding="GBK">
```

import 中的分隔符应该是逗号，不能用分号。

(2) 页面显示 500 错误，错误信息如下：

```
org.apache.jasper.JasperException: Unable to compile class for JSP:
An error occurred at line: 6 in the generated Java file
Syntax error on tokens, delete these tokens
```

此类信息都表示页面的编写出现了语法错误。

例如，指令中出现了错误字符，或者使用了错误的属性名，或者有错误的属性值。

(3) 页面显示 500 错误，错误信息如下：

```
org.apache.jasper.JasperException: /index.jsp(1,1) Unterminated &lt;
%@ page tag
```

该信息告诉用户：指令标签有错误。

(4) 页面显示中文为乱码。例如：

```
???????JSP??---?????
```

原因见如下代码：

```
<%@ page language="java" contentType="text/html, charset=GBK" import="java.
util.*, java.text.* " pageEncoding="GBK"%>
```

这里 contentType=“text/html, charset=GBK”分隔符用的是逗号，而此处只能用分号。

(5) 错误：ClassNotFoundException。代表类没有被找到的异常。

原因：通常出现在 JDBC 连接代码中，对应的驱动 JAR 包没有导入，或 sqljdbc.jar 对应的 Class.forName(类名)中的类名写错了。

(6) 错误信息：主机 TCP/IP 连接失败。

原因：SQL Server 配置管理器中，未启用对应的 SQL Server 服务的 TCP/IP 协议；或 SQL Server 服务器没有开启服务；或连接字符串中的 localhost 写错了；或启用的服务是开发版的 SQL Server，即启用了 SQL Express 服务；或端口号写成了 localhost:8080。

(7) 出错信息：数据库连接失败。

① 检查 JAR 包导入。

② 检查连接字符串和驱动类字符串(要避免使用 SQL Server 2000 的连接字符串)，例如“databasename=数据库名”写成了“datebasename=数据库名”或“localhost:1433”写成了“localhost:8080”。

2.3 JSP 的指令

JSP 中的指令在客户端是不可见的，它是被服务器解释并执行的。指令通常以“<%@”标记开始，以“%>”标记结束。JSP 中常用的是 page 和 include 指令。JSP 指令的用法如下：

```
<%@ 指令名称  属性 1"属性值 1"  属性 2"属性值 2"  …  属性 n"属性值 n" %>
```

2.3.1 page 指令

page 指令是页面指令，可以定义在整个 JSP 页面范围有效的属性和相关的功能。利用 page 指令，可以指定脚本语言，导入需要的类，指明输出内容的类型，指定处理异常的错

误页面，以及指定页面输出缓存的大小，还可以一次设置多个属性。

page 指令的属性如下：

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

提示

①语法格式说明中的“ [” 和“]” 符号括起来的内容表示可选项。②可以在一个页面上使用多个 page 指令，其中的属性只能使用一次(import 属性除外)。

page 指令将使用这些属性的默认值来设置 JSP 页面，下面介绍 page 指令的 13 个属性。

(1) language 属性：设置当前页面中编写 JSP 脚本所使用的语言，默认值为 java。

例如：

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

目前只可以使用 Java 语言。

(2) contenType 属性：设置发送到客户端文档响应报头的 MIME(Multipurpose Internet Mail Extention)类型和字符编码，多个值之间用“;”分开。contenType 的用法如下：

```
<%@ page contenType="MIME 类型; charset=字符编码"  %>
```

MIME 类型被设置为 text/html，如果该属性设置不正确，如设置为 text/css，则客户端浏览器显示 HTML 样式时，不能对 HTML 标识进行解释，而直接显示 HTML 代码。

在 JSP 页面中，默认情况下设置的字符编码为 ISO-8859-1，即 contentType="text/html; charset= ISO-8859-1"。但一般情况下，应该将该属性设置为

```
contentType="text/html; charset=gb2312"
```

此处设置 MIME 类型为 text/html，网页所用字符集为简体中文(国标码 gb2312)。

(3) import 属性：用来导入程序中要用到的包或类，可以有多个值，无论是 Java 核心包中自带的类还是用户自行编写的类，都要在 import 中引入。import 属性的用法如下：

```
<%@ page import="包名. 类名"  %>
```

如果想要导入包里的全部类，可以这样使用：

```
<%@ page import="包名.*"  %>
```

在 page 指令中，可多次使用该属性来导入多个类。例如：

```
<%@ page import="包名.类1"  %>
<%@ page import="包名.类2"  %>
```

或者通过逗号间隔来导入多个类：

```
<%@ page import="包名.类1,包名.类2" %>
```

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

【例 2-16】显示欢迎信息和用户登录的日期时间。

本例通过导入 java.util.Date 类来显示当前的日期时间。具体步骤如下。

① 使用 page 指令的 import 属性将 java.util.Date 类导入，然后向用户显示欢迎信息，并把当前日期时间显示出来。具体代码如下：

```
<%@ page import="java.util.Date" language="java" contentType="text/html;
charset=gb2312"%>
<html >
<body>
您好，欢迎光临本站！<br/>
您登录的时间是<%=new Date()  %>
</body>
</html>
```

② 运行该页面，结果如图 2-10 所示。

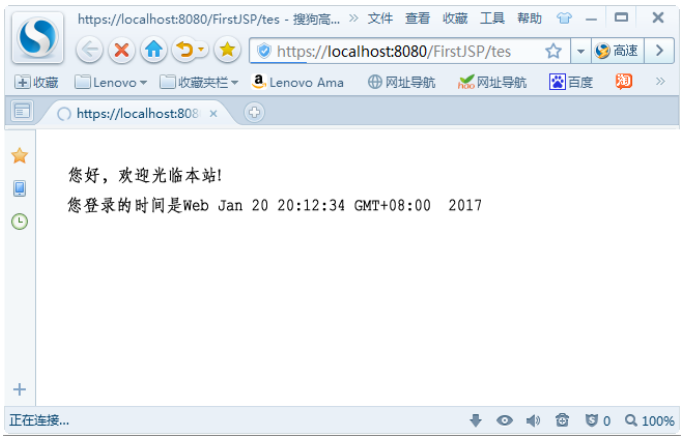

图 2-10　显示欢迎信息和用户登录的日期时间

(4) info 属性：设置 JSP 页面的相关信息，如当前页面的作者、编写时间等。此值可设置为任意字符串，由 Servlet.getServletInfo()方法来获取所设置的值。

【例 2-17】设置并显示 JSP 页面的作者等相关信息。

本例通过 page 指令的 info 属性来设置页面的相关信息，通过 Servlet.getServletInfo()方法来获取所设置的值，具体步骤如下。

① 使用 page 指令的 info 属性设置页面的作者、版本以及编写时间等。具体代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<%@ paqe info="作者: FreshAir < br/>版本: v1<0< br />编写时间:2017 年 3 月 20 日
  星期一<br/>敬请关注, 谢谢!
<html>
<body>
<%
String str = this.getServletlnfo();
out.print (str) ;
%>
</body>
<html>
```

② 运行该页面，结果如图 2-11 所示。

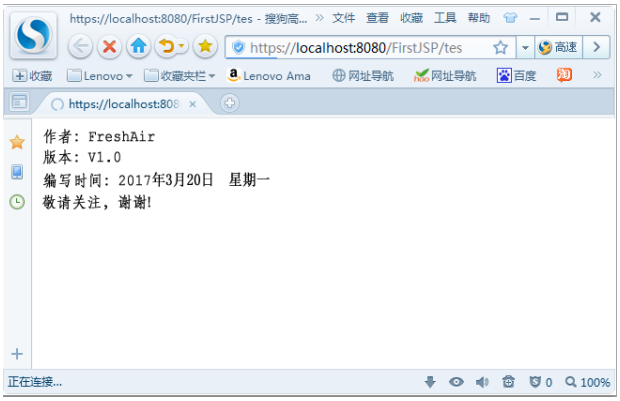

图 2-11 设置并显示 JSP 页面的作者相关信息

(5) extends 属性：指定将 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。

(6) session 属性：表示当前页面是否支持 session，如果为 false，则在 JSP 页面中不能使用 session 对象以及 scope=session 的 JavaBean 或 EJB。该属性的默认值为 true。

(7) errorPage 属性：用于指定 JSP 文件的相对路径，在页面出错时，将转到这个 JSP 文件来进行处理。与此相适应，需要将这个 JSP 文件的 isErrorPage 属性设为 true。

设置 errorPage 属性后，JSP 网页中的异常仍然会产生，只不过此时捕捉到的异常将不由当前网页进行处理，而是由 errorPage 属性所指定的网页进行处理。如果该属性值设置为以“ / ”开头的路径，则错误处理页面在当前应用程序的根目录下；否则在当前页面所在的目录下。

(8) isErrorPage 属性：指示一个页面是否为错误处理页面。设置为 true 时，在这个 JSP 页面中的内置对象 exception 将被定义，其值将被设定为调用此页面的 JSP 页面的错误对象，以处理该页面所产生的错误。

isErrorPage 属性的默认值为 false，此时不能使用内置对象 exception 来处理异常，否则将产生编译错误。

例如，在发生异常的页面上有如下用法：

```
<%@ page errorPage="error.jsp" %>
```

用上面的代码，就可以指明当该 JSP 页面出现异常时，跳转到 error.jsp 去处理异常。而在 error.jsp 中，需要使用下面的语句来说明可以进行错误处理：

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

【例 2-18】页面出现异常的处理。

本例通过 page 指令的 errorPage 和 isErrorPage 两个属性来演示当页面出现异常时应如何处理。具体步骤如下。

① 创建 2-18.jsp 页面，使用 page 指令的 errorPage 属性指定页面出现异常时所转向的页面。具体代码如下：

```
<%@ page contentType="text/html; charset=gb2312"
errorPage="2-18error.jsp" %>
<html>
<body>
<%
//此页面如果发生异常，将向 2-18error.jsp 抛出异常，并令其进行处理
int x1=5;
int x2=0;
int x3=x1/x2;
out.print(x3);
%>
</body>
</html>
```

该程序执行的是除法运算，如果除数为 0，将会抛出一个数学运算异常，从 errorPage="2-18error.jsp"可以看出，程序指定 2-18error.jsp 为其处理异常。

② 创建 2-18error.jsp 页面，使用 page 指令的 isErrorPage 属性指定为出错页面，此页面可以使用 exception 异常对象处理错误信息。具体代码如下：

```
<%@ page contentType="text/html; charset=gb2312" isErrorPage="true" %>
<html>
<body>
出现错误，错误如下：<br/>
<hr>
<%=exception.getMessage() %>
</body>
</html>
```

③ 运行 2-18.jsp 页面，结果如图 2-12 所示。

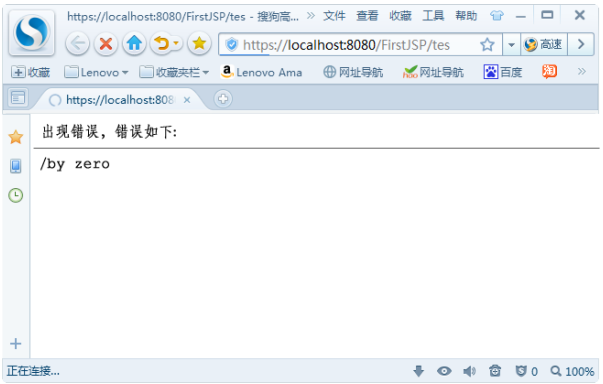

图 2-12　页面出现异常处理

提示

为了确保当页面出错时跳转到 errorPage 所指的页面，需要打开 IE 浏览器，选择“工具”→“Internet 选项”菜单命令，在弹出的对话框中选择“高级”选项卡，取消选中“显示友好 HTTP 错误信息”复选框。

(9) buffer 属性：内置输出流对象 out 负责将服务器的某些信息或运行结果发送到客户端显示，buffer 属性用来指定 out 缓冲区的大小。其值可以是 none、8KB 或是给定的 KB 值。值为 none 表示没有缓存，直接输出至客户端的浏览器中；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，默认为 8KB(因不同的服务器而不同，但大多数情况下都为 8KB)。

(10) autoFlush 属性：当缓冲区满时，设置是否自动刷新缓冲区。默认值为 true，表示当缓冲区满时，自动将其中的内容输出到客户端；如果设为 false，则当缓冲区满时会出现 JSP Buffer overflow 溢出异常。

提示

当 buffer 属性的值设置为 none 时，autoFush 属性的值不能设置为 false。

(11) isThreadSafe 属性：设置 JSP 页面是否可以多线程访问。默认值为 true，表示当前 JSP 页面被转换为 Servlet 后，会以多线程的方式处理来自多个用户的请求；如果设置为 false，则转换后的 Servlet 会实现 SingleThreadMode 接口，并且将以单线程的方式来处理用户请求。

(12) pageEncoding 属性：设置 JSP 页面字符的编码，常见的编码类型有 ISO-8859-1、gb2312 和 GBK 等。默认值为 ISO-8859-1。其用法如下：

```
<%@ page pageEncoding="字符编码"  %>
```

例如：

```
<%@ page pageEncoding="gb2312"  %>
```

这表示网页使用了 gb2312 编码，与 contentType 属性中的字符编码设置作用相同。

(13) isELIgnored 属性：其值可设置为 true 或 false，表示是否在此 JSP 网页中执行或忽略表达式语言${}。设置为 true 时，JSP 容器将忽略表达式语言。

2.3.2 include 指令

include 指令用于通知 JSP 引擎在翻译当前 JSP 页面时，将其他文件中的内容合并进当前 JSP 页面转换成的 Servlet 源文件中，这种在源文件级别进行引入的方式，称为静态引入，当前 JSP 页面与静态引入的文件紧密结合为一个 Servlet。这些文件可以是 JSP 页面、HTML 页面、文本文件或是一段 Java 代码。其语法格式如下：

```
<%@ include file="relativeURL | absoluteURL"  %>
```

说明如下。

(1) file 属性指定被包含的文件，不支持任何表达式，例如下面是错误的用法：

```
<% String f="top.html";  %>
<%@ include file ="<%=f  %>"  %>
```

(2) 不可以在 file 所指定的文件后接任何参数，如下用法也是错误的：

```
<%@ include file="top.jsp?name=zyf"  %>
```

(3) 如果 file 属性值以“ / ”开头，将在当前应用程序的根目录下查找文件；如果是以文件名或文件夹名开头，将在当前页面所在的目录下查找文件。

提示

使用 include 指令是以静态方式包含文件，也就是说，被包含文件将原封不动地插入 JSP 文件中，因此，在所包含的文件中不能使用<html></html>、<body></body >标记，否则会因为与原有的 JSP 文件有相同标记而产生错误。另外，因为原文件和被包含文件可以相互访问彼此定义的变量和方法，所以要避免变量和方法在命名上产生冲突。

【例 2-19】使用 include 指令标记静态插入一个文本文件 Hello.txt，并在当前页面同一个 Web 服务目录中显示“很高兴认识你！Nice to meet you.”，具体操作步骤如下。

① Hello.txt 文本文件的代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
很高兴认识你!
Nice to meet you.
```

② 创建 2-19.jsp 页面，具体代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html> <body bgcolor=cyan>
<H3> <% include file="Hello.txt" %>
</H3>
</body>
</html>
```

2.4 JSP 的动作

JSP 动作利用 XML 语法格式的标记来控制服务器的行为，完成各种通用的 JSP 页面功

能，也可以实现一些处理复杂业务逻辑的专用功能。如利用 JSP 动作可以动态地插入文件、重用 JavaBean 组件、把用户重定向到另外的页面、为 Java 插件生成 HTML 代码。

JSP 动作与 JSP 指令的不同之处是，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 JSP 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 JSP 页面，所以，JSP 指令是在页面转换时期被编译执行的，且编译一次；而 JSP 动作是在客户端请求时按照在页面中出现的顺序被执行的，它们只有被执行的时候才会去实现自己所具有的功能，且基本上是客户每请求一次，动作标识就会执行一次。

JSP 动作的通用格式如下：

```
<jsp:动作名 属生 1="属性值 1" … 属性 n="属性值 n" /> 
```

或者

```
<jsp:动作名；属性 l="属性值 1" … 属性 n="属性值 n">相关内容</jsp:动作名>
```

JSP 中常用的动作包括<jsp:include>、<jsp:param>、<jsp:forward>、<jsp:plugin>、<jsp:useBean>、<Jsp:setProperty>、<jsp:getProperty>。

2.4.1 <jsp:include>动作标记

<jsp:include>动作标记用于把另外一个文件的输出内容插入当前 JSP 页面的输出内容中，这种在 JSP 页面执行时引入的方式称为动态引入，这样，主页面程序与被包含文件是彼此独立的，互不影响。被包含的文件可以是一个动态文件(JSP 文件)，也可以是一个静态文件(如文本文件)。

其语法格式如下：

```
<jsp:include page="relativeURL | <%= expressicry%>" />
```

说明：page 属性指定了被包含文件的路径，其值可以是一个代表相对路径的表达式。当路径以“/”开头时，将在当前应用程序的根目录下查找文件；如果是以文件名或文件夹名开头，将在当前页面的目录下查找文件。书写此动作标记时，“jsp”和“:”以及“include”三者之间不要有空格，否则会出错。

<jsp:include>动作标记对包含的动态文件和静态文件的处理方式是不同的。如果包含的是一个静态文件，被包含文件的内容将直接嵌入 JSP 文件中存放<jsp:include>动作的位置，而且当静态文件改变时，必须将 JSP 文件重新保存(重新转译)，然后才能访问变化了的文件；如果包含的是一个动态文件，则由 Web 服务器负责执行，把执行后的结果传回包含它的 JSP 页面中，若动态文件被修改，则重新运行 JSP 文件时就会同步发生变化。

【例 2-20】在 JSP 文件中使用<jsp:include>动作标记包含静态文件。

① 创建静态文件 staFile.txt，输入以下代码：

```
<font color="blue" size="3">
<br>这是静态文件 staFile.txt 的内容!
</font>
```

② 创建主页面文件 2-20.jsp，具体代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<body>
使用&lt;jsp:include&gt;动作标记将静态文件包含到 JSP 文件中!
</hr>
<jsp:include page="staFile.txt"  />
</body>
</html>
```

③ 运行 2-20.jsp，运行结果如图 2-13 所示。

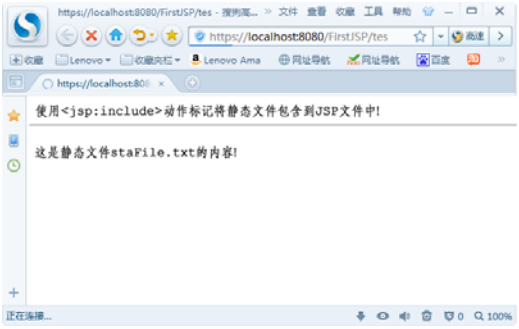

图 2-13　使用<jsp:include>动作标记包含静态文件

要注意，<jsp:include>动作与前面讲解的 include 指令作用类似，现将它们之间的差异总结如下。

(1) 属性不同。

include 指令通过 file 属性来指定被包含的页面，该属性不支持任何表达式。如果在 file 属性值中应用了 JSP 表达式，会抛出异常。例如下面的代码：

```
<% String fpath="top.jsp"; % >
<%@ include file="<%=fpath%>"  %>
```

该用法将会抛出如下异常：

```
File  "/<%=fpath%>" not fount
```

<jsp:include>动作是通过 page 属性来指定被包含页面的，该属性支持 JSP 表达式。

(2) 处理方式不同。

使用 include 指令包含文件时，被包含文件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，所以最终编译后的文件只有一个。

而使用<jsp:include>动作包含文件时，只有当该标记被执行时，程序才会将请求转发到(注意是转发，而不是请求重定向)被包含的页面，再将其执行结果输出到浏览器中，然后重新返回到包含页来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 编器将对这两个文件分别进行编译。

(3) 包含方式不同。

include 指令的包含过程为静态包含，因为在使用 include 指令包含文件时，服务器最终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。

<jsp:include>动作的包含过程为动态包含，通常用来包含那些经常需要改动的文件。因为服务器执行的是两个文件，被包含文件的改动不会影响主文件，因此服务器不会对主文件重新编译，而只须重新编译被包含的文件即可。并且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>动作被执行时，使用该标记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。

(4) 对被包含文件的约定不同。

使用 include 指令包含文件时，因为 JSP 编译器是对主文件和被包含文件进行合成后再翻译，所以对被包含文件有约定。例如，被包含的文件中不能使用<html></html>、<body></body>标记；被包含文件要避免变量和方法在命名上与主文件冲突的问题。

提示

如果在 JSP 页面中需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 指令或动作标记包含进来，以提高代码的可读性。

2.4.2 <jsp:param>动作标记

当使用<jsp:include>动作标记引入的是一个能动态执行的程序时，如 Servlet 或 JSP 页面，可以通过使用<jsp:param>动作标记向这个程序传递参数信息。

其语法格式如下：

```
<jsp:include page="relativeURL | <%=expression%>">
<jsp:param name="pName1" value="pValuel | <%=expression1%>" />
<jsp:param name="pName2" value="pValue2 l |<%=expression2%>" />
…
</jsp : include>
```

说明：<jsp:param>动作的 name 属性用于指定参数名，value 属性用于指定参数值。在<jsp:include>动作标记中，可以使用多个<jsp:param>传递参数。另外，<jsp:forward>和<jsp:plugin>动作标记中都可以利用<jsp:param>传递参数。

【例 2-21】使用<jsp:param>动作标记向被包含文件传递参数。

① 创建主页面 2-21.jsp，用<jsp:include>包含用于对三个数进行排序的页面 order.jsp，并且使用<jsp:param>向其传递 3 个参数。具体代码如下：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
<title> param 动作标记应用示例 </title>
</head>
<body>
使用&lt;jsp:include&gt;包含用于对三个数进行排序的页面 order.jsp,<br>
并利用&lt;jsp:param&gt;把待排序的三个数 8,3,5 传给 order.jsp 后，<br>
所得结果如下：
<hr/>
<jsp:include page="order.jsp">
```

```
  <jsp:param name="num1" value="8">
<jsp:param name="num2" value="3">
<jsp:param name="num3" value="5">
</jsp:include>
</body>
</html>
```

② 创建用于对三个数进行排序的页面 order.jsp，具体代码如下：

```
<% @ page contentType="text/html;charset=gb2312" %>
<html>
<head>
<title> param 动作标记应用示例 </title>
</head>
<body>
<%
String str1=request.getParameter("num1");    //取得参数 num1 的值
int m1=Inter.parseInt(str1);              //将字符串转换成整型
String str2=request.getParameter("num2");    //取得参数 num2 的值
int m2=Inter.parseInt(str2);              //将字符串转换成整型
String str3=request.getParameter("num3");    //取得参数 num3 的值
int m3=Inter.parseInt(str3);              //将字符串转换成整型
int t;
if(m1>m2)
{
t=m1;
m1=m2;
m2=t;
}
if(m2>m3)
{
t=m2;
m2=m3;
m3=t;
}
if(m1>m2)
{
t=m1;
m1=m2;
m2=t;
}
%>
<font color="blue" size="4">
这三个数从小到大的顺序为：<%=m1>、<%=m2>、<%=m3>
</body>
</html>
```

③ 运行 2-21.jsp，运行结果如图 2-14 所示。

图 2-14　使用<jsp:param>动作标记向被包含文件传递参数

2.4.3　<jsp:forward>动作标记

在大多数的网络应用程序中，都有这样的情况：在用户成功登录后转向欢迎页面，此处的“转向”，就是跳转。<jsp:forward>动作标记就可以实现页面的跳转，用来将请求转到另外一个 JSP、HTML 或相关的资源文件中。当该标记被执行后，当前的页面将不再被执行，而是去执行该标记指定的目标页面，但是，用户此时在地址栏中看到的仍然是当前网页的地址，而内容却已经是转向的目标页面了。

其语法格式如下：

```
<jsp:forward page="relativeURL" | "<%=expression  %>" />
```

如果转向的目标是一个动态文件，还可以向该文件传递参数，使用格式如下：

```
<jsp: forward page="relativeURL" | "<%=expression%>" />
<jsp:param name="pName1" value="pValuel | <%=expression1%>" />
<jsp:param name="pName2" value="pValue2 | <%=expression2%>" />
```

说明如下：

(1) page 属性用于指定要跳转到的目标文件的相对路径，也可以通过执行一个表达式来获得。如果该值以“ / ”开头，表示在当前应用的根目录下查找目标文件，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 动作转发到外部的文件中，将出现资源不存在的错误信息。

(2) forward 动作执行后，当前页面将不再被执行，而是去执行指定的目标页面。

(3) 转向到的文件可以是 HTML 文件、JSP 文件、程序段，或者其他能够处理 request 对象的文件。

(4) forward 动作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，事先储存在 request 对象中的信息都不存在了。

【例 2-22】使用<jsp: forward>动作标记实现网页跳转。

① 创建主页面 2-22.jsp，通过表单输入用户名和密码，单击“登录”按钮，利用<jsp: forward>动作标记跳转到页面 target.jsp。具体代码如下：

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

```
<form action=" " method="bost" name="Form"> <!--提交给本页处理-->
用户名:<input name="UserName" type="text"> <br/>
密 & nbsp;码; <input name="UserPwd" type="text"> <br/>
<input type="submit" value"登录">
</form>
<%
   //单击“登录”按钮时，调用 Form1.submit()方法提交表单至本文件，
   //用户名和密码均不为空时，跳转到 targe.jsp，并且把用户名和密码以参数形式传递
   String s1=null, s2=null;
   s1=request.getParameter("UserName");
   s2=request.getParameter("UserPwd ");
   if(s1!=null && s2!=null)
   {
%>
      <jsp:forward page="target.jsp">
        <jsp:param name="Name" value="<%=s1%>"/>
        <jsp:param name="Pwd" value="<%=s2%>"/>
      </jsp:forward >
<%
    }
%>
</body>
</html>
```

② 创建所转向的目标文件 target.jsp，具体代码如下：

```
<%@ page contentType="text/html;  charset=gb2312" %>
<html>
<body>
<%
String strName=request.getParameter("UserName");
String strPwd=request.getParameter("UserPwd ");
out.println(strNmae+"您好，您的密码是:"+strPwd);
%>
```

③ 运行 2-22.jsp，结果如图 2-15 所示。

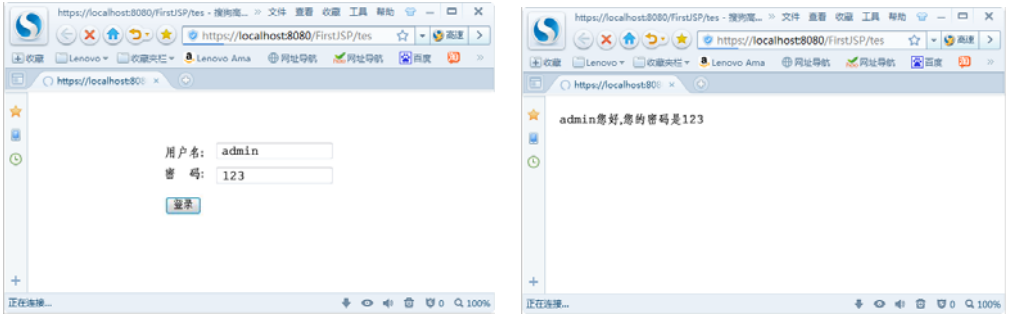

图 2-15　使用<jsp:forward>动作标记实现网页跳转

2.4.4　<jsp:plugin>动作标记

<jsp:plugin>动作可以在页面中插入 Java Applet 小程序或 JavaBean，它们能够在客户端

运行，但此时，需要在 IE 浏览器中安装 Java 插件。当 JSP 文件被编译并送往浏览器时，<jsp:plugin>动作将会根据浏览器的版本，替换成<object>或者<embed>页面 HTML 元素。注意，<object>用于 HTML 4.0，<embed>用于 HTML 3.2。

通常，<jsp:plugin>元素会指定对象是 Applet 还是 Bean，同样也会指定 class 的名字以及位置。另外，还会指定将从哪里下载 Java 插件。该动作的语法格式如下：

```
<jsp:plugin
type="bean | app1et"  code="C1assFi1eName"
codebase="c1assFi1eDirectoryName"
[name="instanceName"]
[archive="URIToArchive,…"]
[align="bottom | top | middle | left | right"]
[height="disp1ayPixels"]
[width="displayPixels"]
[hspace="1eftRightPixels"]
[vspace="topBottomPixels"]
[jreversion="JREVersionNumber | 1.1"]
[nspluginurl="URLToPlugin"]
[iepluginurl="URLToPlugin"]>
[<jsp:params>
<jsp:param name="parameterName"
value="{parameterValue |<%=expression %>"/>
</jsp:params>]
[<jsp:fallback>text message for user</jsp:fallback>]
</jsp:plugin>
```

参数说明如下：

(1) type 属性的作用是定义插入对象的类型，对象类型有两个值，分别是 bean 或者 applet。(必须定义的属性)

(2) code 属性定义插入对象的类名，该类必须保存在 codebase 属性指定的目录内。(必须定义的属性)

(3) codebase 属性定义对象的保存目录。(必须定义的属性)

(4) name 属性定义 bean 或 Applet 的名字。

(5) archive 属性定义 Applet 运行时需要的类包文件。

(6) align 属性定义 Applet 的显示方式。

(7) height 属性定义 Applet 的高度。

(8) width 属性定义 Applet 的长度。

(9) hspace 属性定义 Applet 的水平空间。

(10) vspace 属性定义 Applet 的垂直空间。

(11) jreversion 属性定义 Applet 运行时所需要的 JRE 版本，缺省值是 1.1。

(12) nspluginurl 属性定义 Netscape Navigator 用户在没有定义 JRE 运行环境时下载 JRE 的地址。

(13) iepluginurl 属性定义 IE 用户在没有定义 JRE 运行环境时下载 JRE 的地址。

(14) jsp:params 标识的作用是定义 Applet 的传入参数。

(15) jsp:fallback 标识的作用是当对象不能正确显示时传给用户的信息。

【例 2-23】使用<jsp:plugin>动作标记在 JSP 中加载 Java Applet 小程序。

① 创建 2-23.jsp 页面，使用<jsp: plugin >动作标记加载：

```
<%@ page contentType="text/html;  charset=gb2312" %>
<html>
<body>
加载 MyApplet.class 文件的结果如下: <hr/>
<jsp:plugin type="applet" code="MyApplet.class" codebase="." >
  jreversion="1.2" width="400" heigh="80"
<jsp:fallback>
   加载 Java Applet 小程序失败!
</jsp:fallback>
</jsp:plugin>
</body> </html>
```

② 其中插件所执行的类 MyApplet.class 的源文件为 MyApplet.java，代码如下：

```
import java.applet.*;
import java.awt.*;

public class MyApplet extends Applet
{
   public void paint(Graphics g)
   {
      g.setColor(Color.red);
      g.drawString("您好!我就是 Applet 小程序!",5,10);
      g.setColor(Color.green);
      g.drawString("我是通过应用<jsp:plugin>动作标记",5,30);
      g.setColor(Color.blue);
      g.drawString("将 Applet 小程序嵌入到 JSP 文件中",5,50);
      }
}
```

将 2-23.jsp 及 MyApplet.java 文件经过 Java 编译器编译成功后，生成的 MyApplet.class 字节文件都存放在 ch02 目录下。

重新启动 Tomcat 后，在 IE 浏览器的地址栏中输入 http://localhost:8080/ch02/2-23.jsp，按 Enter 键后，若客户机上没有安装 JVM(Java 虚拟机)，将会访问 Sun 公司的网站，并且弹出下载 Java plugin 的界面。下载完毕后，将会出现 Java plugin 插件的安装界面，可以按照向导提示，逐步完成安装过程。然后，就可以使用 JVM 而不是 IE 浏览器自带的 JVM 来加载执行 MyApplet.class 字节码文件了，最终得到的运行结果如图 2-16 所示。

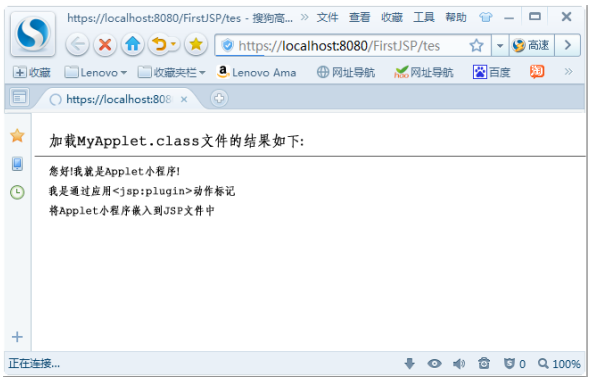

图 2-16　使用<jsp:plugin>标记在 JSP 中加载 Java Applet 小程序

2.4.5 <jsp:useBean>动作标记

<jsp:useBean>动作标记用于在 JSP 页面中创建 bean 实例，并且通过设置相关属性，可以将该实例存储到指定的范围。如果在指定的范围已经存在该 bean 实例，那么将使用这个实例，而不会重新创建。

实际工程中，常用 JavaBean 做组件开发，而在 JSP 页面中，只需要声明并使用这个组件，较大程度地实现了静态内容和动态内容的分离。

声明 JavaBean 的语法格式如下：

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

也可以在标记体内嵌入子标记，例如：

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

以上两种使用方法是有严格区别的，当在页面中使用<jsp:useBean>标记创建一个 bean 时，对于第二种使用格式，如果该 bean 是第一次被实例化，那么标记体内的内容会被执行；如果已经存在指定的 bean 实例，则标记体内的内容就不再被执行。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指定的 bean 实例，<jsp:useBean>标记后面的内容都会被执行。

下面对<jsp:useBean>动作中各属性的用法进行详细介绍。

(1) id 属性：在 JSP 中给这个 bean 实例取的名字，即指定一个变量，只要在它的有效范围内，均可使用这个名称来调用它。该变量必须符合 Java 中变量的命名规则。

(2) scope 属性：设置所创建的 bean 实例的有效范围，取值有 4 种：page、request、session、application。默认情况下取值为 page。

① 值为 page：在当前 JSP 页面及当前页面以 include 指令静态包含的页面中有效。

② 值为 request：在当前的客户请求范围内有效。在请求被转发至的目标页面中，如果要使用原页面中创建的 bean 实例，通过 request 对象的 getAttribute(“id 属性值”)方法来获取。请求的生命周期是从客户端向服务器发出请求开始，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 实例也就失效了。

③ 值为 session：对当前 HttpSession 内的所有页面都有效。当用户访问 Web 应用程序时，服务器为用户创建一个 session 对象，并通过 session 的 ID 值来区分不同的用户。针对某一个用户而言，对象可被多个页面共享。通过 session 对象的 getAttribute(“id 属性值”)方法获取存储在 session 中的 bean 实例。

④ 值为 application：所有用户共享这个 bean 实例。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，可以被多个用户共享。所以，访问 application 对象的所有用户共享存储于该对象中的 bean 实例。使用 application 对象的 getAttribute(“id 属性值”)方法获取存在于 application 对象中的 bean 实例。

Scope 属性之所以很重要，是因为只有在不存在具有相同 id 和 scope 的对象时，<sp:useBean>才会实例化新的对象；如果已有 id 和 scope 都相同的对象，则直接使用已有的对象，此时，jsp:useBean 开始标记和结束标记之间的任何内容都将被忽略。

(3) type=“数据类型”：设置由 id 属性指定的 bean 实例的类型。该属性可指定要创建实例的类的本身、类的父类或者一个接口。

通过 type 属性设置 bean 实例类型的格式如下：

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

如果在 session 范围内，名为 stu 的实例已经存在，则将该实例转换为 type 属性指定的 StudentInfo 类型(此时的类型转换必须是合法的)并赋值给 id 属性指定的变量;若指定的实例不存在，将会抛出“bean stu not found within scope”异常。

(4) class=“package.className”：该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。例如，通过 class 属性定位一个类的格式如下：

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

程序首先会在 session 范围中查找是否存在名为 stu 的 StudentInfo 类的实例，如果存在，就会通过 new 操作符实例化 StudentInfo 类来获取一个实例，并以 stu 为实例名称存储在 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，这两个属性一起使用时的格式举例说明如下：

```
<jsp:useBean id="stu" class="com.Bean.StudentInfo"
type="com.Bean. StudentBase"
scope="session"  />
```

(6) beanName=“package.className” type=“数据类型”：beanName 属性与 type 属性可以指定同一个类，这两个属性一起使用时的格式举例说明如下：

```
<jsp:useBean id="stu"  beanName="com.Bean.StudentInfo"
type="com.Bean.StudentBase"/>
```

假设 StudentBase 类为 StudentInfo 类的父类。执行到该标记时，首先，程序会创建一个以 id 属性值为名称的变量 stu，类型为 type 属性的值，并初始化为 null；然后在 session 范围内查找名为 stu 的 bean 实例。如果实例存在，则将其转换为 type 属性指定的 StudentBase 类型(此时的类型转换必须是合法的)并赋值给变量 stu：如果实例不存在，则将通过

instantiate()方法，从 StudentInfo 类中实例化一个类，并将其转换成 StudentBase 类型后赋值给变量 stu，最后将变量 stu 存储在 session 范围内。

一般情况下，使用如下格式来应用<jsp:useBean>标记：

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

如果多个页面中共享这个 bean 实例，可将 scope 属性设置为 session。

使用<jsp:useBean>标记在页面中实例化 bean 实例后，设置或修改该 bean 中的属性就可以用<jsp:setProperty>来完成，读取该 bean 中指定的属性要用<jsp:getProperty>来完成，这两个标记在下面小节中将陆续介绍。当然，读取和设置 bean 中的属性还有另一种方式，就是在脚本程序中利用 id 属性所命名的对象变量，通过调用该对象的方法显式地读取或者修改其属性。

2.4.6 <jsp:setProperty>动作标记

<jsp:setProperty>标记通常与<jsp:useBean>标记一起使用，它以请求中的参数给创建的 JavaBean 中对应的属性赋值，通过调用 bean 中的 setXxx()方法来完成。其语法格式如下：

```
<jsp:useBean id="变量名" ... />
{
  property="*"
    | property="propertyName"
    | property="propertyName"  parme="parmeterName"
    | property="propertyName"  value="值"
 }
/>
```

下面对<jsp:setProperty>动作中各属性的用法进行详细介绍。

(1) name 属性：用来指定一个存在于 JSP 中某个范围内的 bean 实例。

<jsp:setProperty>动作标记将按照 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。如果任何范围内都不存在这个 bean 实例会抛出异常。

(2) property=“*”：当 property 的取值为“*”时，要求 bean 属性的名称与类型要与 request 请求中参数的名称及类型一致，以便用 bean 中的属性来接收客户输入的数据，系统会根据名称来自动匹配。如果 request 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。这两种情况下的 bean 属性都会保留原来的值或者默认的值。

此种使用方法的限定条件是：请求中参数的名称和类型必须与 bean 中属性的名称和类型完全一致。但通过表单传递的参数都是 String 类型，所以，JSP 会自动地将这些参数转换为 bean 中对应属性的类型。

表 2-1 给出了 JSP 自动将 String 类型转换为其他类型时所调用的方法。

表 2-1 JSP 自动将 String 类型转换为其他类型时所调用的方法

其他类型	转换方法
Integer	java.lang.Integer.value()Of(String)
int	java.lang.Integer.value()Of(String).intValue()
Double	java.lang.Double.value()Of(String)
double	java.lang.Double.value()Of(String).doubleValue()
Float	java.lang.Float.value()Of(String)
float	java.lang.Float.value()Of(String).floatValue()
Long	java.lang.Long.value()Of(String)
long	java.lang.Long.value()Of(String).longValue()
Boolean	java.lang.Boolean.value()Of(String)
boolean	java.lang.Boolean.value()Of(String).booleanValue()
Byte	java.lang.Byte.value()Of(String)
byte	java.lang.Byte.value()Of(String).byteValue()

(3) property=“upropertyName”：当 property 属性取值为 bean 中的属性时，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。如果请求中没有与 property 所指定的同名参数，则该 bean 属性会保留原来的值或默认的值，而不会被赋值为 null。与 property 属性值为“*”时一样，当请求中参数的类型与 bean 中的属性类型不一致时，JSP 会自动进行转换。

(4) property=“propertyName” param=“parameterName”：property 属性指定 bean 中的某个属性，param 属性指定 request 请求中的参数。该种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 属性会保留原来的值或默认的值，而不会被赋为 null。

(5) property=“propertyName”value=“值”：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。当 value 属性是一个字符串时，如果指定的 bean 属性与其类型不一致，JSP 会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，则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，将会抛出 argument type mismatch 异常。

2.4.7 <jsp:getProperty>动作标记

<jsp:getProperty>标记用来获得 bean 中的属性，并将其转换为字符串，再在 JSP 页面中输出，该 bean 中必须具有 getXxx()方法。使用的语法格式如下：

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

下面对 name 属性和 property 属性的用法进行详细介绍。

(1) name 属性：用来指定一个存在于 JSP 中某个范围内的 bean 实例。

<jsp:getProperty>标记会按照 page、request、session 和 application 的顺序查找 bean 实例，直到第一个实例被找到。如果任何范围内都不存在这个 bean 实例，则会抛出异常。

(2) property 属性：该属性指定要获取由 name 属性指定的 bean 中的哪个属性的值。若

它指定的值为 stuName，那么 bean 中必须存在 getStuName()方法，否则会抛出异常。如果指定 bean 中的属性是一个对象，那么该对象的 toString()方法将被调用，并输出执行结果。

【例 2-24】创建一个 JavaBean，设置并且读取它的 info 属性值。

① 在 Eclipse 环境下，创建 JavaBean 文件 SimpleBean.java，步骤如下。

鼠标右击，从快捷菜单中选择“新建”→“包”命令，输入包名“com.bean”，然后右击包名 com.bean，从快捷菜单中选择“新建”→“类”命令，输入类名“SimpleBean”，之后输入如下代码：

```
package com.bean;
public class SimpleBean {
   private String message=" ";
   public String getMessage ( )  {
      return(message);
   }
   Public void setMessage(String str)  {
      this.message=str;
   }
}
```

② 新建文件 2-24.jsp，创建名为 Beanl 的 JavaBean，设置 message 属性的值为“您好，欢迎使用 JSP!”，再获取其值输出到页面。具体代码如下：

```
<%@ page contentType="text/html;  charset=gb2312" %>
<html>
<body>
使用动作标记&lt;jsp:useBean&th;创建一个 Bean 实例，名称为 Bean1,<br/>
&lt;setProperty&gt;用于设置 Bean1 中属性 message 的值为"您好,欢迎使用 JSP!",<br/>
&lt:setProperty&gt;用于获取 Bean1 中属性 message 的值并输出<br/>
<jsp:useBean id="Bean1" class="com.bean.SimpleBean"  />
<jsp:setProperty name="Bean1" property="message">
  value="您好, 欢迎使用 JSP! "  />
<hr>
<font size=4 color="blue"> message 的值为:
  <jsp:getProperty name="Bean1" property="message"  />
</font>
</body>
</html>
```

③ 运行 2-24.jsp，运行结果如图 2-17 所示。

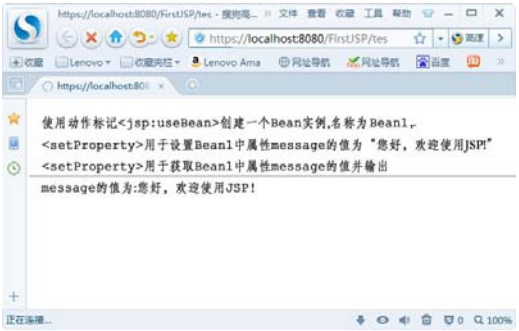

图 2-17　创建和使用 JavaBean

2.5 案例：JSP 指令标记

实训内容和要求

编写三个 JSP 页面：first.jsp、second.jsp 和 third.jsp。另外，要求用“记事本”编写一个 hello.txt 文件。hello.txt 中的每行都有若干个英文单词，这些单词之间用空格分隔，每行之间用
分隔，如下所示：

```
package apple void back public
<BR>
private throw class hello welcome
```

实训步骤

(1) first.jsp 的具体要求。

first.jsp 使用 page 指令设置 contentType 属性的值为 text/plain，使用 include 指令静态插入 hello.txt 文件。first.jsp 的代码如下：

```
<%@ page contentType="text/plain" %>
<html>
<body>
<font Size=4 color=blue>
<%@ include file="hello.txt" %>
</font>
</body>
</html>
```

(2) second.jsp 的具体要求。

second.jsp 使用 page 指令设置 contentType 属性的值为 application/vn.ms-powerpoint，使用 include 指令静态插入 hello.txt 文件。second.jsp 的代码如下：

```
<%@ page contentType="application/vnd.ms-powerpoint" %>
<html>
<body>
<font Size=2 color=blue>
<%@ include file="hello.txt" %>
</font>
</body>
</html>
```

(3) third.jsp 的具体要求。

third.jsp 使用 page 指令设置 contentType 属性的值为 application/msword，使用 include 指令静态插入 hello.txt 文件。third.jsp 的代码如下：

```
<%@ page contentType="application/msword" %>
<html>
<body>
<font Size=4 color=blue>
<%@ include file="hello.txt" %>
```

```
</font>
</body>
</html>
```

本 章 小 结

本章主要介绍了 JSP 的基本语法(注释、声明、代码段、表达式)、JSP 程序开发模式、调试处理、JSP 指令标记和动作标记的使用。指令标记在编译阶段就被执行，通过指令标记，可以向服务器发出指令，要求服务器根据指令进行操作，这些操作相当于数据的初始化。动作标记是在请求处理阶段被执行的。通过对本章的学习，读者可以掌握 JSP 的基本语法编写格式，以及指令和动作的使用语法。

习　　题

一、填空题

1. _____是一段在客户端请求时需要先被服务器执行的 Java 代码，它可以产生输出，同时也是一段流控制语句。

2. 在 JSP 的三种指令中，用来定义与页面相关的指令是_____指令；用于在 JSP 页面中包含另一个文件的指令是_____(静态包含)；用来定义一个标签库以及其自定义标签前缀的指令是_____。

3. <jsp:include>动作元素允许在页面被请求时包含一些其他资源，如一个静态的_____文件和动态的 JSP_____文件。

4. JSP 的隐藏注释格式为_____或者_____，JSP 的输出注释的格式是_____。

5. Page 指令的 MIME 类型的默认值为_____，默认字符集是_____ 。

二、选择题

1. 在 JSP 中调用 JavaBean 时不会用到的标记是(　　)。

   A. <javabean>　B. <jsp:useBean>　C. <jsp:setProperty>　D. <jsp:getProp>

2. 关于 JavaBean 正确的说法是(　　)。

   A. Java 文件与 bean 所定义的类名可以不同，但一定要注意区分字母的大小写

   B. 在 JSP 文件中引用 bean，其实就是用<jsp:useBean>语句

   C. 被引用的 bean 文件的文件名后缀为.java

   D. bean 文件放在任何目录下都可以被引用

3. 对于预定义<%!预定义%>的说法错误的是(　　)。

   A. 一次可声明多个变量和方法，只要以“;”结尾就行

   B. 一个声明仅在一个页面中有效

   C. 声明的变量将作为局部变量

   D. 在预定义中声明的变量将在 JSP 页面初始化时初始化

4. 在 JSP 中使用<jsp:getProperty>标记时，不会出现的属性是(　　)。
   A. name　　B. property　　C. value　　D. 以上皆不会出现
5. 对于<%!和%>之间声明的变量，以下说法正确的是(　　)。
   A. 不是 JSP 页面的成员变量
   B. 多个用户同时访问该页面时，任何一个用户对这些变量的操作，都会影响到其他用户
   C. 多个用户同时访问该页面时，每个用户对这些变量的操作都是互相独立的，不会互相影响
   D. 以上皆正确

三、问答题

1. 阐述 include 静态包含和动态包含指令<jsp:include>有哪些区别。
2. 阐述 include 指令标记与 include 动作标记有哪些不同。
3. 编写一个 JSP 页面，显示大写英文字母表。