第5章 基本JSP程序设计

Java Server Pages(JSP)是基于Java语言的脚本技术,是Java EE Web层用于生成网页的另外一种主要技术。JSP使用类似于HTML的标记和Java代码段,能够将HTML代码从Web页面的业务逻辑中分离出来。JSP技术规范的目标是通过提供一种比Servlet更简洁的程序设计结构,以简化动态Web页面的生成和管理。每个JSP页面在第一次被调用时都会被翻译成一个Servlet,而该Servlet是JSP页面中的标记和脚本标记指定的嵌入动态内容的结合体。本章首先介绍JSP的基本运行原理、执行机制和生命周期,然后在此基础上介绍JSP页面的基本组成、语法、JSP隐含对象的用途和适用范围。最后,本章通过实例阐述了在Oracle JDeveloper和OC4J环境下,开发、部署和运行JSP的基本原理和方法。

5.1 JSP概述

JSP是一种用于取代CGI的技术,而且性能比CGI脚本优越。Servlet在服务器上运行并且截获来自客户端浏览器的请求,适用于确定如何处理客户请求以及调用其他的服务器对象。但是,Servlet并不适用于生成页面内容。另外,从Java代码中生成标记是很难维护的,而且Servlet必须由熟悉Java语言的开发者编写。Servlet与JSP在功能上虽然有所重叠,但是可以把Servlet看作控制对象,而把JSP看作视图对象。在创建Java Web应用时,不需要在使用Servlet还是使用JSP之间做出艰难的选择。Servlet与JSP是互补的技术,对复杂的Web应用,它们两者都会被使用到。

另一方面,JSP利用JavaBeans和Java标记对静态HTML代码和动态数据进行了分离。静态HTML代码由HTML程序员负责编写,而动态数据和JavaBeans由Java程序员负责编写。这样的分工原则,可以使不同的程序员专注于各自的领域。

5.1.1 JSP 运行原理

当客户使用浏览器上网时，服务器能够解释来自客户机的信息，这是因为客户机和服务器都遵守了 TCP/IP 族的标准。TCP/IP 通过 Internet 决定了信息的分发和路由。HTTP 给出了 Web 页面请求和响应的格式，这些协议共同工作在网络的传输层上。浏览器根据 HTTP 定制的规则构造请求，然后浏览器通过另一个称为 TCP/IP 堆栈的软件处理请求。TCP/IP 堆栈初始化请求的分发和路由。当请求最终到达一台服务器时，它的 TCP/IP 堆栈将所有到达的请求组合在一起，并将请求传输给服务器软件（即 OC4J 提供的 JSP 容器），根据 HTTP 规则解释这个请求。

服务器不仅运行 JSP 容器，也运行一个 TCP/IP 堆栈，还可以运行其他的软件。通常，JSP 容器仅负责请求/响应周期的 HTTP 部分。如果用户请求的是一个 JSP 页面，则服务器将这个请求转发给 JSP 容器，JSP 容器解释 JSP 代码并构造一个 HTML 文档传输给浏览器。

服务器将请求转发给一个 JSP 容器有许多方法，其中常用的一种方法是通过 TCP/IP 堆栈进行通信。通常服务器软件和 JSP 容器并不驻留在同一台计算机上，它们通过一个 TCP/IP 堆栈共享信息。图 5-1 阐述了 JSP 的运行原理。

提示：读者出于学习目的，可以在本地计算机上同时运行一个客户机浏览器、一个服务器和一个 JSP 容器（OC4J），在运行时再对 Web 应用进行实际的部署。

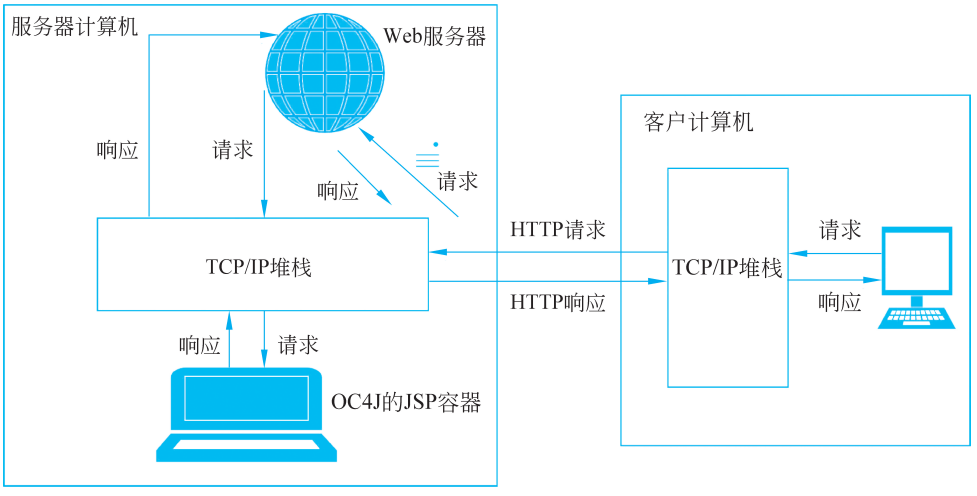

图 5-1 服务器与 JSP 容器通过 TCP/IP 堆栈进行通信

5.1.2 JSP 生命周期

JSP 生命周期包括 jspInit()、jspService()，以及 jspDestroy()3 个方法，这些方法是根据 JSP 的状态从 JSP 容器中被调用的。

javax.servlet.jsp 包中定义了一个 JspPage 接口（继承自 Servlet 接口），该接口定义了 jspInit()与 jspDestroy()两个方法。无论客户端使用哪种通信协议，实现 JspPage 接口的

类都可以经由 jspInit()与 jspDestroy()方法完成初始化和资源释放动作。针对 HTTP 通信协议，javax.servlet.jsp 包定义了一个 HttpJspPage 接口。该接口只定义了一个 jspService()方法。

一般地，把 jspInit()、jspService()，以及 jspDestroy() 3 个方法称为 JSP 生命周期方法。

- 当一个 JSP 页面第一次请求时，JSP 容器将把该 JSP 页面转换为一个 Servlet。JSP 容器首先把该 JSP 页面转换成一个 Java 源文件，在转换时如果发现语法错误，则将中断转换，并向服务器和客户机输出错误信息；如果转换成功，JSP 容器将用 javac 把 Java 源文件编译成.class 文件。上述过程执行完成之后，JSP 容器将创建一个 Servlet 实例，该 Servlet 的 jspInit()方法将被执行。
- jspInit()方法在 Servlet 生命周期中只被执行一次，然后将调用 jspService()方法处理来自客户机的请求。对每一个请求，JSP 容器将会创建一个新线程来处理这个请求。如果有多个客户机同时请求这个 JSP 页面，则 JSP 容器将会创建多个线程。每个客户机请求对应一个线程。以多线程方式执行 JSP 页面可大大降低对系统的资源需求，提高系统的并发量及响应时间，但是应当注意多线程的编程限制。
- 由于这个 Servlet 始终驻留在内存之中，所以响应速度非常快。如果 JSP 页面被修改过，服务器将根据设置决定是否对其重新编译。如果需要重新编译，则将编译结果取代内存中的 Servlet，并继续进行上述过程。虽然 JSP 执行效率很高，但在第一次调用时由于需要进行转换和编译，所以会稍有延迟。此外，如果出现系统资源不足的情形，jspDestroy()方法首先将被调用，然后 Servlet 实例将被标记加入“垃圾回收器”处理。

5.1.3　JSP 执行过程

JSP 页面的执行过程如图 5-2 所示。

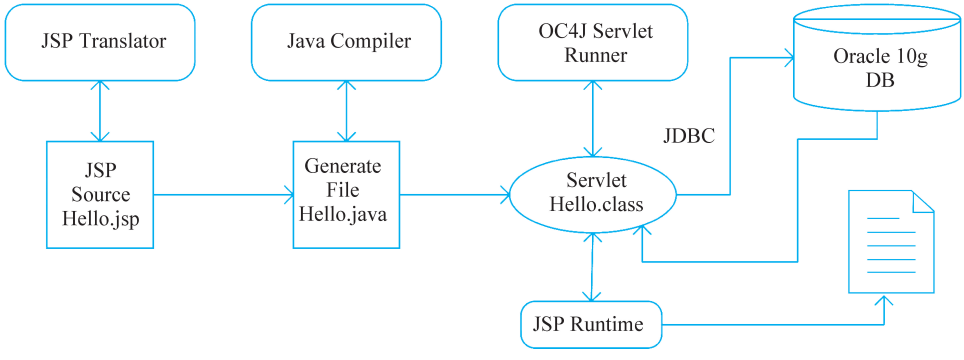

图 5-2　JSP 页面的执行过程

- JSP 页面及其相关文件总称为翻译单元(Translation Unit)。JSP 容器第一次为一个 JSP 页面截获一个请求时，翻译单元将把它翻译成一个 Servlet。这个编译过程

包括两个阶段：第一个阶段是把 JSP 源代码转换成一个 Servlet；第二个阶段包括编译这个 Servlet。

- JSP 页面第一次由 JSP 容器装入时，实现 JSP 标记的 Servlet 代码会自动生成、编译并加载到 OC4J 提供的 Servlet 容器中。这种情况发生在翻译时间，而且只在第一次请求 JSP 页面时才发生，所以第一次访问 JSP 页面时响应速度会稍慢些。但以后的请求直接由已经编译过的 Servlet 处理，这些过程发生在运行时间里。因此，理解 JSP 页面执行过程的关键是正确区分翻译时间与运行时间。

5.2 JSP 脚本元素

JSP 页面包含 HTML 与 JSP 两类标记。HTML 标记用一对尖括号括住，而 JSP 标记则括在一对尖括号和一对百分号中。HTML 是页面设计的基本语言，使用 JSP 标记的文档会被 JSP 容器转换成 HTML 标记。JSP 不仅可以使用本身固有的语法，也可以和 HTML 标记一起使用。JSP 页面可以是模板元素、注释、脚本元素、指令元素以及操作元素这 5 种元素中的一种或组合体。

如果 JSP 页面中使用的编程语言位于 JSP 标记的外部，则 JSP 容器会将它识别为模板数据(Template Data)，并直接将它显示在页面中。模板元素是指 JSP 的静态 HTML 或 XML 内容，它对 JSP 的显示是非常必要的。模板元素是网页的框架，它影响着页面的结构和美观程度。在编译 JSP 页面时，JSP 容器将把模板元素编译到 Servlet 中。当客户请求该 JSP 页面时，JSP 容器会把模板元素发送到客户端。一个带有 Java 代码的 JSP 标记称为脚本元素，JSP 页面中有声明、表达式和脚本 3 种类型的脚本元素。

1. 声明

声明(Declaration)用于在 JSP 页面中定义方法和实例变量。声明并不生成任何将会送给客户机的输出。其一般语法格式如下：

```
< % !方法或实例变量的定义 %>
```

2. 表达式

表达式(Expression)用于把动态内容(变量值或方法的运算结果)传递给客户端浏览器界面。其一般语法格式如下。

```
< %=变量或方法的值 %>
```

“变量或方法的值”可以是前面声明中定义的方法或实例变量中的任何一个返回值。例如，下面的程序片段：

```
<%! public double pi=3.14159; %>
<%=pi %>      →输出 3.14159
pi            →输出 pi
```

从输出结果可以看出，pi 只有在 JSP 表达式中才有效，而没有使用表达式的 pi 将被原样显示在浏览器的界面上。在表达式中可以使用任何形式的变量，JSP 容器会将标记

中的值全部转换成字符串并输出到浏览器界面上。

3. 脚本

脚本(Scripting)是用于解释 Java 语言的标记。如果在 JSP 页面中嵌入 Java 语句,则 JSP 容器会解释脚本标记内的 Java 语句,并根据其语法在浏览器页面上运行。其一般语法格式如下。

```
<%Java 代码 %>
```

脚本标记中的 Java 代码大都用于完成运算功能,所以这些 Java 代码的运算结果无法直接显示在浏览器页面中。与声明不同,脚本标记可以将一个完整的程序分开使用。例如下面的程序片段:

```
<%for(int i=0; i<3; i++) { %>
    这是脚本
<%} %>
```

4. 注释

所谓注释(Comments),是在程序代码中用于说明程序流程的语句。JSP 页面中的注释语句如表 5-1 所示。

表 5-1　JSP 页面的注释语句

种　　类	表 示 形 式
HTML 注释语句	<!--注释-->
JSP 注释语句	<%--注释语句--%>
Script 语言注释语句	<%//注释语句%><%/*注释语句*/%>

5.3　基于 IDE 开发 JSP 页面

基于 JDeveloper 开发 JSP 页面,可以充分利用 IDE 的可视化编辑器提供的 HTML 组件面板和 JSP 组件面板提供的标记,有效地提升开发效率。下面通过一个实例,介绍开发 JSP 页面的可视化编辑器的使用方法。这个实例要求编写一个 JSP 页面,用于显示 1～6 的平方根表。

(1) 在 JDeveloper 中创建 ch05.jws,在该 Application 中创建一个工程文件 sqrtTable.jpr。

(2) 在工程文件 sqrtTable 中创建一个 JSP 程序。从主菜单中选择 File→New 命令,在显示的对话框的 Categories 区域中选择 Web Tier→JSP,在 Item 区域中选择 JSP,单击“确定”按钮,则显示 Welcome to the Create JSP Wizard 窗口。单击“下一步”按钮,显示如图 5-3 所示的选择 Web 应用的版本对话框。

(3) 单击“下一步”按钮,显示如图 5-4 所示的对话框。在 File Name 域输入 sqrtTable.jsp,在 Directory Name 域使用它的默认值,Type 选择 JSP Page(*.jsp)。

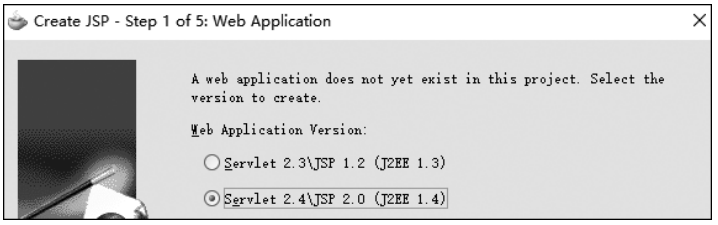

图 5-3　选择 Web 应用的版本

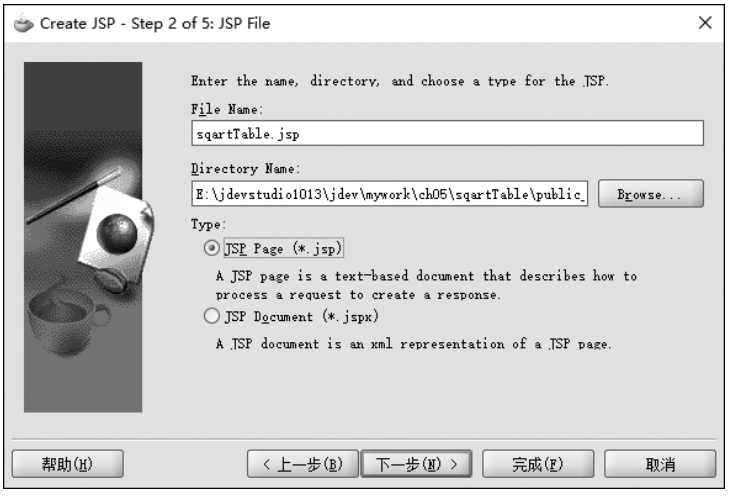

图 5-4　输入 Web 应用的名称、路径以及选择程序类型

（4）单击“下一步”按钮，显示选择“出错页面参数”对话框，如图 5-5 所示。

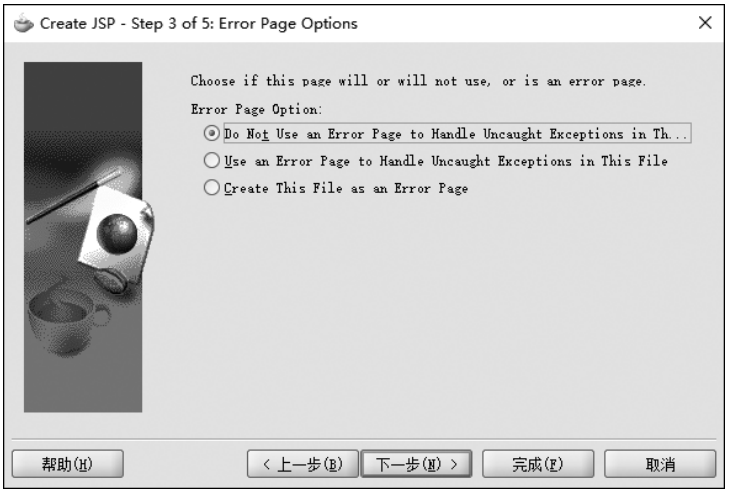

图 5-5　选择“出错页面参数”对话框

（5）单击“下一步”按钮，显示“使用其他组件库”对话框，本例暂不使用。单击“下一

步”按钮，显示如图 5-6 所示的对话框。可以选择 HTML 版本，输入 JSP 页面的标题，设置 JSP 页面的背景颜色和样式表。

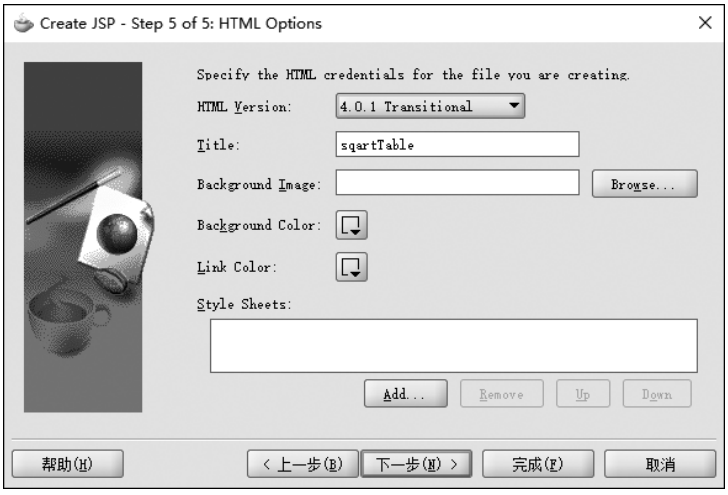

图 5-6　选择 HTML 版本，输入 JSP 页面的标题等

(6) 单击“完成”按钮，在 Code Editor 生成 JSP 页面源代码。

(7) 根据题目要求修改生成的 JSP 源代码，具体如下所示。

```
1.  <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
2.  "http://www.w3.org/TR/html4/loose.dtd">
3.  <%@ page contentType="text/html;charset=GB2312"%>
4.  <HTML><HEAD>
5.  <META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=GB2312">
6.  <TITLE>Table Of Square Roots</TITLE></HEAD>
7.  <BODY>
8.  <!--表标题-->
9.  <CENTER>平方根表</CENTER>
10. <!--表头-->
11. <TABLE CELLSPACING="2" CELLPADDING="1" BORDER="2" WIDTH="100%" ALIGN=
    "LEFT">
12. <TR>
13. <TD>数字</TD>
14. <TD>平方值</TD>
15. </TR>
16. <%--表体部分，输出数的平方根--%>
17. <%for(int n=1;n<=6;n++) { %>
18. <TR>
19. <%/*输出数*/ %><TD><%=n%></TD>
20. <%/*输出数的平方根*/ %><TD><%=Math.sqrt(n) %></TD>
21. </TR>
22. <%} %>
```

```
23.  </TABLE></BODY>
24.  </HTML>
```

(8) 完成 JSP 页面的源代码编写工作之后，就可以将 Web 应用部署到 OC4J 容器。图 5-7 所示为 JSP 页面的执行结果。

图 5-7　JSP 页面的执行结果

【分析讨论】

- 第 3 句表示指定输出文本为 HTML，字符编码为 GBK。
- 该 JSP 页面创建了一个 HTML 表，表的每一行都封装在＜TR＞和＜/TR＞标记中。由于使用了 for 循环，所以执行一次循环体，都要用一对新的＜TR＞和＜/TR＞标记创建表的一个新行。
- 每一对标记中都有两个 JSP 表达式：一个用于输出数 n；一个用于输出该数的平方根。

5.4　JSP 隐含对象

JSP 可以使用 Servlet API 提供的特定隐含对象。这些对象在 JSP 页面中可以使用标准变量实现访问，并且不再需要开发人员重新声明，就可以在任何 JSP 表达式和脚本中使用，这些对象称为隐含对象(Implicit Objects)。利用这些隐含对象，可以在 JSP 页面中直接存取 Web 应用的运行环境信息，如表 5-2 所示。例如，通过 request 对象可以取得 HTTP 请求内容，通过 application 对象可以取得 web.xml 文件的配置信息等。

表 5-2　JSP 隐含对象及用途

JSP 隐含对象	用　　途
request	客户机发送的 HTTP 请求
response	服务器要发送给客户端的 HTTP 响应
out	输出数据流的 JspWrite 对象
config	JSP 页面编译后产生的 Servlet 相关信息的 ServletConfig 对象

续表

JSP 隐含对象	用　　途
pageContext	JSP 页面信息的 pageContext 对象
page	相当于 Java 语言的 this 对象
exception	其他 JSP 抛出的异常
session	用于存取 HTTP 会话内容
application	Web 应用配置信息的 ServletContext 对象

5.4.1　对象使用范围

JSP 对象、JavaBeans 对象以及隐含对象的使用范围是一个非常重要的概念，因为它定义了相关对象来自哪个 JSP 页面以及生存时间。对象的使用范围在内部依赖于上下文环境(Context)，一个 Context 为资源提供了一个不可见的容器与接口，供它们与程序环境通信。例如，一个 Servlet 在一个上下文环境中运行，这个 Servlet 就需要知道关于服务器的所有资源都可以从这个上下文环境中提取，并且服务器需要与这个 Servlet 通信的所有信息通过这个上下文环境传递。JSP 技术规范为开发人员使用的对象定义了 4 个范围，如表 5-3 所示。

表 5-3　对象的使用范围

维　护	使 用 说 明
page	只能在引用特定对象的 JSP 页面中访问这些对象
request	可以在所有服务于当前请求的页面中访问这些对象，其中包括转发到或包含在原始 JSP 页面中的页面。相应的请求被导入这些 JSP 页面
session	只能通过定义相关对象时访问的 JSP 页面访问这些对象
application	应用程序范围对象可以由一个给定上下文环境中的所有 JSP 页面访问

5.4.2　request 对象

HTTP 描述了来自客户机的请求与服务器的响应两方面内容。在 JSP 页面中，可以用 request 和 response 两个隐含对象表示这两方面的内容。

下面的 Web 应用实例，用于得到来自客户机请求的相关信息。该 Web 应用的实现原理是：request 对象是 ServletRequest 接口的一个针对协议和具体实现的子类，具有 request 使用范围。而且 request 对象包含了客户机向服务器发出请求的内容，可以通过该对象了解客户机向服务器发出请求的内容和客户端所要求的信息。

在 ch05.jws 中创建一个工程文件 jspRequest.jpr，在该工程中创建一个 JSP 页面文件 jspRequest.jsp 和一个部署描述文件 jspRequest.deploy。jspRequest.jsp 的源代码如下所示。

```
1.  <%@ page contentType="text/html;charset=GB2312"%>
2.  <HTML><HEAD><META HTTP-EQUIV="Content-Type" CONTENT="text/html;
    charset=GB2312">
3.  <TITLE>request/response 实例</TITLE></HEAD>
4.  <BODY>
5.  <B>Browser:</B><%=request.getHeader("User-Agent") %><BR>
6.  <B>Cookies:</B><%=request.getHeader("Cookie") %><BR>
7.  <B>Accepted MIME types:</B><%=request.getHeader("Accept") %><BR>
8.  <B>HTTP method:</B><%=request.getMethod() %><BR>
9.  <B>IP Address:</B><%=request.getRemoteAdd() %><BR>
10. <B>DNS Name(or IP Address again):</B><%=request.getRemoteHost() %><BR>
11. <B>Country:</B><%=request.getLocale().getDisplayCountry() %><BR>
12. <B>Language:</B><%=request.getLocale().getDisplayLanguage() %><BR>
13. </BODY></HTML>
```

JSP 页面的执行结果如图 5-8 所示。

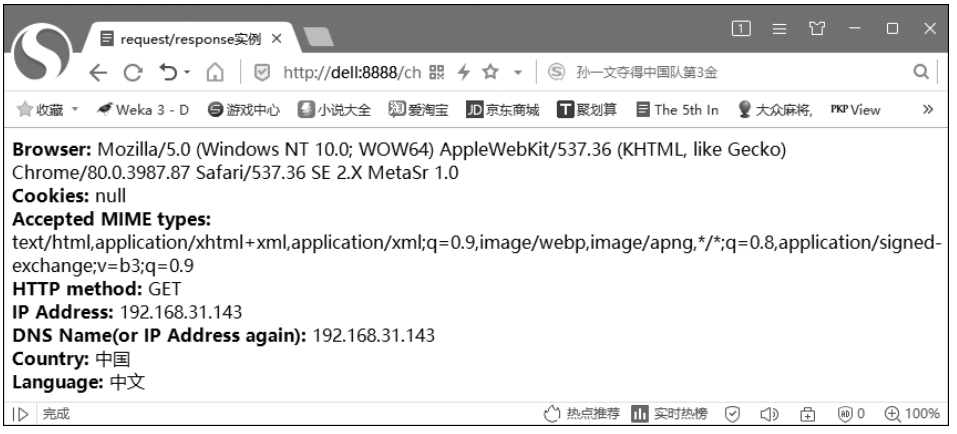

图 5-8 JSP 页面的执行结果

【分析讨论】

- 一个 HTTP 请求可以有一个或多个标题，每个标题都有一个名字和一个值。为了返回客户机请求的标题值，JSP 页面调用了 request 对象的 getHeader()方法。通过 JSP 页面中的第 5、6、7 句可以得到以下 3 个值：User-Agent——包含一个描述客户机浏览器的信息；Cookie 标题——包含客户机前一次访问服务器时发送的信息，标题值是一个标题号。Accept 标题——包含浏览器响应请求时接收到的 MIME 类型的列表。
- JSP 页面中的第 8 句，通过调用 request 对象的 getMethod()方法得到 HTTP 的请求方法是 get()。
- 第 9 句通过调用 getRemoteAdd()方法返回产生请求的客户机 IP 地址。第 10 句通过调用 getRemoteHost()方法得到服务器的 DNS。如果没有可用的 DNS，则服务器再次返回 IP 地址。