第 3 章　第一个JSP应用

在第 1 章中，已经介绍了 Web 的请求响应机制，以及 JSP 实现动态页面所存在的特点和优势。第 2 章重点介绍了如何搭建 JSP 应用的运行环境，其中包括 JDK 软件包的安装和配置，以及 Tomcat 服务器的安装和配置。另外还介绍两种用于 Web 开发的编辑器。

在了解了第 1 章和第 2 章的基本知识后，就可以动手创建一个简单的 JSP 应用了。这一章节即将带领读者逐步学会如何创建和开发第一个 JSP 的 Web 应用。

本章要点包括以下内容：

- 创建 Web 应用模块
- Web 应用的功能介绍
- Web 应用的实现
- Web 应用模块的发布和卸载
- 如何修改 Web 应用模块的调用路径

3.1　创建Web应用模块

在开发一个 Web 应用程序时，需要在服务器端创建一个 Web 应用模块（也称为 Web 站点），将所用 Web 页面程序放置在这个模块中进行统一调用和管理。一个 Web 模块中一般包括所有创建的页面程序以及 XML 部署文件和 JAR 包文件（将零散文件打包成 JAR 类型的压缩文件）等。

这里将创建的第一个 JSP 应用取名为 welcome，即该 Web 应用模块名为 welcome。当需要调用 welcome 模块下的 index.jsp 页面时，可以在 IE 浏览器中输入请求地址：http:localhost:8080/welcome/index.jsp。这里的 localhost 指定为本机（端口号为 8080），当然也可以使用域名或者 IP 来指定网络中某台服务器；该请求地址表示调用本地服务器中 welcome 模块下的 index.jsp 页面文件。

3.1.1　Tomcat服务器默认Web应用模块调用路径

当客户端通过地址 http:DNS/welcome/index.jsp 调用域名为 DNS 的服务器下的 welcome 模块中的 index.jsp 页面时，服务器端又是如何查找到名为 welcome 的 Web 模块呢？显然搜索整个硬盘查找是不现实的，那么服务器是如何解决这个问题的呢？其中有两种方法可以想到：

- 指定一个特定的目录路径，即将所有创建的 Web 模块放置在这个目录下，服务器专门查找这个目录下的 Web 模块。
- 通过服务器中的配置文件指定某个模块的真实存放路径，每次调用时，服务器首先在配置文件中查找到该模块的实际路径，然后到那个目录下调用和执行相应页面程序。

Tomcat 服务器正是使用这两种方法来查找 Web 模块的。Tomcat 服务器默认的 Web 应用模块调用路径为 Tomcat 5.0\webapps 目录下。Tomcat 首先会查找该目录下的所有 Web 应用模块。至于使用配置文件指定 Web 模块路径的方法将在本章后面小节重点讲解。

3.1.2　Tomcat服务器调用Web模块的类型

Tomcat 服务器进行调用的 Web 模块可以存在如下两种类型：

- ❑ 通过 Ant 工具发布的 Web 模块，一般会被打包成 WAR 文件（扩展名为.war），然后再部署到 Web 服务器上。Tomcat 服务器会自动调用该 WAR 压缩包中的所有页面程序。
- ❑ Tomcat 也可以调用零散文件，即没有打包成 WAR 压缩包的 Web 模块。

在 Web 应用程序开发过程中，会通过 Tomcat 服务器调用零散文件进行调试。而当 Web 应用项目完成之后，需要作为一个版本打包发布到 Web 服务器上进行调用执行。这时 Web 模块就应该以 WAR 压缩包形式存在。这样易于在服务器上部署（因为除了 Web 模块之后，还有可能存在 EJB 等其他模块），而零散文件存在服务器就显得零乱了。

3.1.3　创建该JSP应用的Web模块

本章节所介绍的第一个 JSP 应用将创建在 Tomcat 服务器默认的 Tomcat 5.0\webapps 目录下，并将以零散文件的形式进行调用。创建的该 Web 模块名为 welcome，例如图 3.1 展示了创建的 welcome 模块所在的目录。

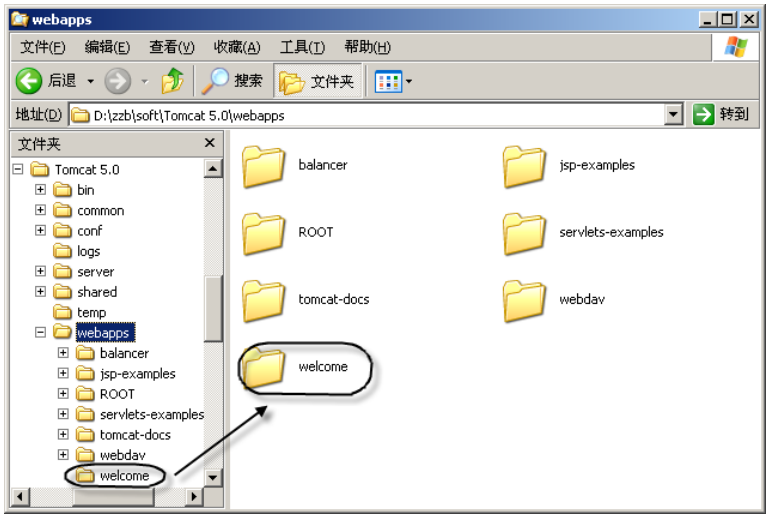

图 3.1　welcome 模块所在目录

一般 Web 模块中包括一个 WEB-INF 子目录，该子目录下放置 Web 应用的配置文件、CLASS 类文件（Java 文件编译之后的二进制文件）以及 JAR 包文件。在第七章节将向读者介绍如何使用 Eclipse＋Lomboz 工具方便地构建 Web 项目和 Web 模块。

3.2　Web应用的功能介绍

本章创建的第一个 JSP 应用将实现用户欢迎提示功能。整个 Web 应用的功能非常简单，仅仅包括两个 JSP 页面，即用户通过 reguser.jsp 页面中的表单提交用户姓名，而 weluser.jsp 页面用于输出欢迎该用户的提示语。图 3.2 显示了页面之间的调用关系，用户通过 reguser.jsp 页面填写个人姓名，然后将表单提交给 weluser.jsp 进行处理，weluser.jsp 页面负责在显示器上打印出“Welcome ×××！”字符串，

其中×××表示某用户姓名。

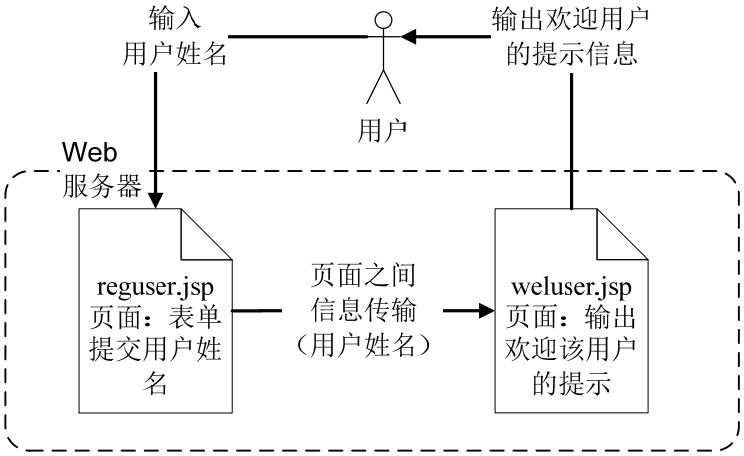

图 3.2　页面之间的调用关系

当用户调用 reguser.jsp 页面时，运行效果如图 3.3 所示，需要用户在输入框中填写个人姓名，然户单击“提交”按钮将页面提交给 weluser.jsp 页面进行相应处理。weluser.jsp 页面的运行效果如图 3.4 所示，它负责输出欢迎用户的提示信息。

图 3.3　reguser.jsp 运行效果

图 3.4　weluser.jsp 运行效果

展示了该 JSP 应用的功能后，接下来的小节将逐步编写代码实现该功能。

3.3　Web应用的实现

Web 容器一般默认调用模块下的 index.htm 或者 index.jsp 页面，例如 welcome 模块下存在一个 index.jsp 页面，则输入 http://localhost:8080/welcome 地址时，Web 容器会默认调用 index.jsp 页面。这是通过 web.xml 配置文件（在 Tomcat 5.0\conf 目录下）指定的，在配置文件中设置默认访问页面的代码段如下所示：

```
<web-app>
……
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
        <welcome-file>index.htm</welcome-file>
        <welcome-file>index.jsp</welcome-file>
```

```
    </welcome-file-list>
……
</web-app>
```

代码说明：该配置文件的设置指定 Web 容器默认调用各模块下的 index.html、index.htm 或者 index.jsp 页面文件。

注意：以上提及的 web.xml 配置文件在 Tomcat 5.0\conf 目录下，该文件为 Tomcat 容器的全局配置文件。另外，各 Web 模块也可以有自己的 web.xml 配置文件，如果各单独 Web 模块没有明确指定，则按照 Tomcat 的全局配置文件设置属性，否则按照各 Web 模块的配置信息进行设置。

3.3.1　创建index.jsp页面

该页面程序非常简单，它只是该 JSP 应用的入口页面，因为通过 web.xml 配置文件已经指定默认调用 index.jsp 页面文件。该页面程序实现页面跳转，代码如下：

```
<%
    //注释：调用 JSP 内置对象 response 中的 sendRedirect()方法跳转到 reguser.jsp 页面
    response.sendRedirect("reguser.jsp");
%>
```

代码说明：

（1）JSP 页面允许使用<%　%>来嵌入 Java 语句，以实现页面的动态输出。

（2）在<%　%>之间的注释语句是和 Java 一样，有关 JSP 中的多种注释方法将在第五章节中详细介绍。

（3）在 JSP 中保留了多个内置对象，以方便调用这些对象中的方法（不需要初始化，即可以直接调用对象的方法，就类似于调用类的静态方法一样）。response 就是其中的一个内置对象，该内置对象专门负责处理用户响应，有关内置对象的介绍见第六章内容。

3.3.2　创建reguser.jsp页面

reguser.jsp 页面负责登记用户姓名，单击“提交”按钮将表单信息提交给 weluser.jsp 页面进行相应处理。该页面的实现代码如下所示：

```
<%@ page contentType="text/html; charset=GBK" language="java" import="java.sql.*" errorPage="" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=GBK">
<title>输入用户名</title>
</head>
<body background="images/bg.gif">
<br>
<form name="form" action="weluser.jsp" method="get">
<table width="200" border="1" align="center">
  <tr>
    <td> 用户名：</td>
    <td> <input type="text" name="username" size="10"></td>
  </tr>
  <tr>
```

```
    <td colspan="2" align="center">
      <input type="submit" value="提交"> | <input type="reset" value="重置">
    </td>
  </tr>
</table>
</form>
</body>
</html>
```

程序说明：

（1）程序的第一行是有关 page 指令属性的设置，contentType 属性指定页面输出格式为 html 以及使用 GBK 编码规则。其中页面输出格式除了 html 之外还可以是 xml、image 等类型，编码规则还可以是 gb2312、uft-8 等。有关指令的详细知识见第五章内容。

（2）页面程序的第二行是有关文档类型定义（DTD），读者不需要对此有太多了解，而且在本实例中，有没有这行代码并不重要。读者如果有兴趣，请自己查看有关 XML 中的文档类型定义的知识。

（3）程序的其他部分完全符合 HTML 代码编写规则，有关 HTML 知识可以查看本书第四章的内容介绍。

（4）需要重点介绍的是<form>标签中的 method 属性值的设置，method 属性值除了“get”之外还可以是“post”、“put”和“delete”。但是在实际开发过程中，经常使用的就是“get”和“post”。当表单提交少量数据时，可以将 method 属性值设置为“get”，而当需要传输大量数据信息（例如图片等文件信息）时，method 值需要设置成“post”。

其实该页面程序还存在不足之处，即没有检查用户名是否为空的情况。如果用户没有填写姓名，则应该不予提交，并给出用户名为空的提示。下面使用 JavaScript 页面脚本语言来实现该功能，其中有关 JavaScript 的详细介绍见第四章内容。

3.3.3　修改reguser.jsp页面程序

在 reguser.jsp 页面中的<body>标签前面插入如下一段 JavaScript 代码：

```
<script language="javascript">
<!--
  function checkuser(){
    if(document.form.username.value == ""){        //如果 username 输入框内容为空
      alert("请输入用户名");                        //弹出提示对话框
      document.form.username.focus();              //光标停留在 username 输入框中
    }
    Else                                           //否则进行表单提交
      document.form.submit();
}
-->
</script>
```

代码说明：有关 JavaScript 代码的详细编写规则和语法见第四章内容，该段 JavaScript 代码实现对 username 输入框内容是否为空进行判断，并做出相应处理。

另外需要将<input type="submit" value="提交">代码替换成<input type="button" value="提交" onClick="checkuser()">。其中的 onClick="checkuser()"语句定义一个单击事件，当用户单击“提交”按钮时将触发 JavaScript 中定义的 checkuser()方法。

当用户名为空时，单击“提交”按钮的运行效果如图 3.5 所示。

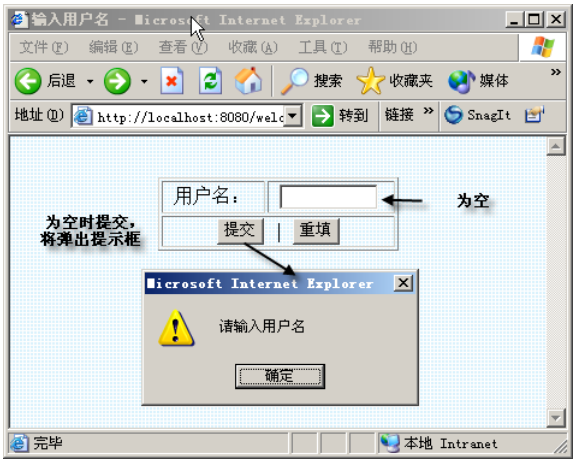

图 3.5　修改后的 reguser.jsp 页面运行效果

3.3.4　创建weluser.jsp页面

weluser.jsp 页面获取 reguser.jsp 页面所提交的用户名信息，并输出欢迎该用户的提示信息。详细代码如下：

```
<%@ page contentType="text/html; charset=GBK" language="java" import="java.sql.*" errorPage="" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=GBK">
<title>欢迎您</title>
</head>
<body background="images/bg.gif">
<center>
<%
      String username = request.getParameter("username");          //获取用户名信息
      out.println("Welcome "+username+"!");                        //输出欢迎该用户的提示信息
%>
</center>
</body>
</html>
```

程序说明：

（1）使用 request 内置对象中的 getParameter()方法来获取到上页面传递的数据。request 内置对象专门负责处理用户请求的操作，有关该内置对象的详细介绍见第六章节内容。

（2）out 也是 JSP 中的一个内置对象，负责操作输出流，详细介绍见第六章内容。

至此，所有的页面程序已经编写完成，下面就可以在前台通过浏览器来访问。但是当调用页面程序时，Web 容器却提示找不到该 Web 模块，这是因为还没有将该 Web 模块发布到服务器上。接下来向读者介绍如何将编写好的 Web 模块发布到服务器上。

3.4　Web模块的发布和卸载

完成的 Web 模块需要发布到 web 服务器上才能在前台通过浏览器进行访问。可以通过 Tomcat

5.0\conf 目录下的 server.xml 配置文件以及在 Tomcat 5.0\conf\Catalina\localhost 目录下创建某模块的特定配置文件来发布。后面介绍使用 Eclipse＋Lomboz 构建 Web 应用时，会介绍使用 Lomboz 来发布 Web 模块。

3.4.1　Web模块的发布

所有发布的 Web 模块可以通过如图 3.6 所示的界面进行统一管理。进入该 Web 管理界面的方法为：单击第 2 章图 2.16 左边的“Tomcat Manager”链接进入，这时需要输入用户名 admin 和密码 123456。

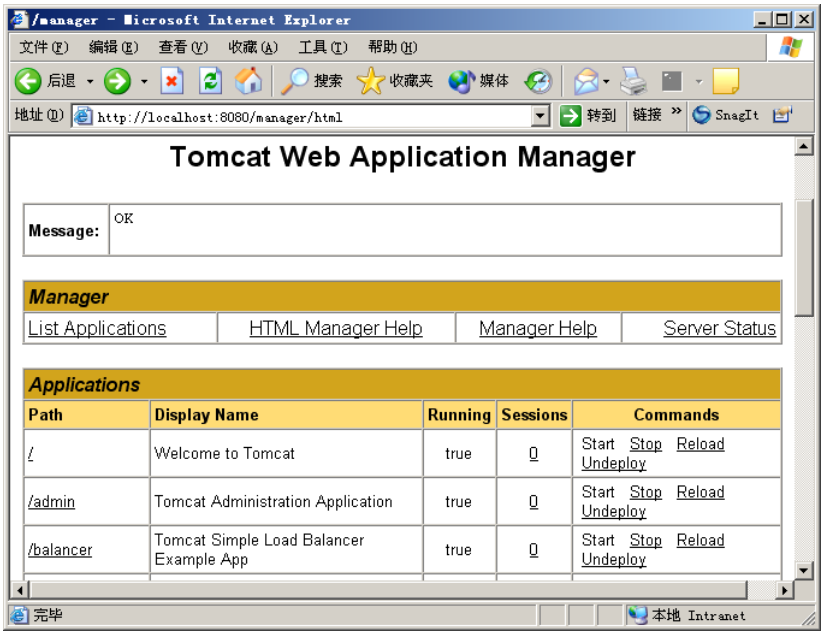

图 3.6　Web 应用的管理界面

在 Web 应用的管理页面中可以单击“Start”、“Stop”命令来启动和停止某个 Web 应用，以及单击“Reload”、“Undeploy”命令来重新装载和卸载某个 Web 应用。下面介绍通过配置文件来发布 Web 模块。

由于以上创建的 Web 模块名为 welcome，所以需要在 Tomcat 5.0\conf\Catalina\localhost 目录下创建一个名为 welcome.xml 的配置文件，文件内容如下：

```
<Context path="/welcome" docBase="welcome" debug="0" privileged="true"></Context>
```

代码说明：

- 这里 path 是设置要发布的 Web 模块名称为 welcome。如果端口是 8080，访问这个 Web 应用的地址为：http://localhost:8080/welcome 或者 http://127.0.0.1:8080/welcome。此处 IP 地址 127.0.0.1 和 localhost 是对等的。
- docBase 定义了开发的 Web 应用程序所在位置，即告诉系统到什么地方查找这个 Web 应用。由于 Tomcat 默认的 Web 调用路径为 Tomcat 5.0\webapps，所以这里设置成相对路径。

如果读者记不得这些代码，可以将同目录下的 balancer.xml 文件复制过来再进行相应修改。

除了创建 welcome.xml 配置文件之外，还可以在 server.xml 配置文件的末尾（</Host>标签前面）插入以上一段代码。只是这样设置，Tomcat 还会在 Tomcat 5.0\conf\Catalina\localhost 目录下自动创建一个 welcome.xml 配置文件。

就这么简单就可以将开发的 Web 应用发布到 Tomcat 服务器上了。

3.4.2　Web应用的运行

Web 应用发布到 Tomcat 服务器上后，就可以通过浏览器来访问了，运行效果如图 3.3 和 3.4 所示。如果这里抛出找不到 JDK 的异常，请查看 JDK 的环境变量是否配置有误。另外打开 Tomcat 的“Configure”配置窗口，确保如图 3.7 所示的配置信息。

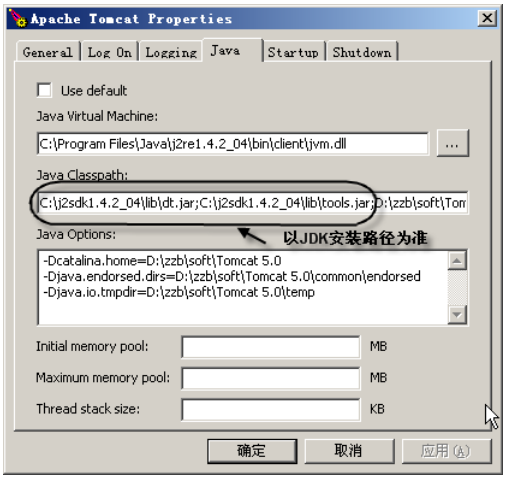

图 3.7　Tomcat 参数配置界面

3.4.3　Web模块的卸载

在前一小节已经对 Web 应用的卸载有所提及，即单击图 3.7 管理界面中的“Undeploy”命令来卸载相应的 Web 应用。卸载完之后，Tomcat 5.0\webapps 目录下的 Web 模块以及 Tomcat 5.0\conf\Catalina\localhost 目录下的配置文件将全部被删除。管理页面中的“Stop”命令是和“Undeploy”不一样的，它仅仅停止该模块的访问，可以通过“Start”命令重新启动。

3.4.4　修改Web应用的调用路径

读者阅读到这里，是否已经发现了一个问题，如果所有的 Web 应用只有创建或者发布到 Tomcat 5.0\webapps 目录下才能通过 IE 访问，那么将给开发人员带来很多麻烦。每当一个页面程序修改了，都要重新发布才能查看修改效果，这样将大大降低开发效率。有没有一种方法可以通过配置文件指定其他目录下的 Web 模块，然后通过浏览器就可以直接访问该 Web 应用。

同样需要使用到 Tomcat 5.0\conf\Catalina\localhost 目录下配置文件（例如 welcome 模块的 welcome.xml 配置文件）。

为了说明问题，接下来将 Tomcat 5.0\webapps 目录下创建的 welcome 模块剪切到 D:\zzb\source 目录下，并通过 Web 应用的管理页面将已存在的 welcome 模块卸载掉。然后修改 Tomcat 5.0\conf\Catalina\localhost 目录下的 welcome.xml 文件，内容如下：

```
<Context path="/welcome" docBase="D:\zzb\source\welcome" debug="0" privileged="true" reloadable="true">

</Context>
```

代码说明：和之前 welcome.xml 配置文件惟一不同的就是 docBase 路径设置使用了绝对路径，以上由于 Web 模块存在 Tomcat 默认的 Tomcat 5.0\webapps 目录下。

另外一种配置方法同样是修改 server.xml 文件，方法和前面一样。

为了证明浏览器访问的是 D:\zzb\source 目录下的 welcome 模块程序，在 weluser.jsp 程序中添加 out.println(" 修改过后 ")代码。然后重新通过浏览器访问该 Web 应用，weluser.jsp 出现如图 3.8 所示的页面效果。

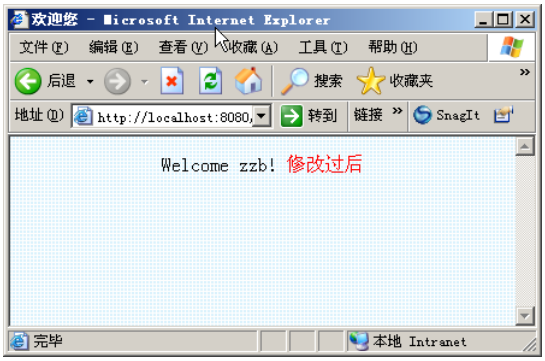

图 3.8　weluser.jsp 页面效果

这样修改之后，开发人员就不需要每次修改一下程序都重新发布一次 Web 模块来查看效果。等整个 Web 应用开发完成后再发布到 Tomcat 服务器上。

3.5　本章小结

本章介绍的实例非常简单，但主要以该实例讲解如何创建和发布一个 Web 应用。考虑到开发的方便性可以通过配置文件指定其他目录下的 Web 模块，从而不要发布就可以查看每次的修改效果。了解了本章内容之后，读者应该对一个 Web 应用的实现过程有一个清晰的思路。