4.1 简单的 JSP 范例：显示一句话

4.1.1 实例说明

这里用一个简单的例子来说明 JSP 的基本结构和语法。这个例子跟很多其他编程语言入门的例子一样，使用 JSP 简单地输出一句“hello world”。通过这个例子，可以了解到 JSP 的基本语法。

用文本编辑器（如前边介绍的 EditPlus）新建一个文本文件，名称为 helloworld.jsp 并在该文件中添加下面 4 行代码：

```
<%@ page contentType="text/HTML;charset=GB2312"%>
<%  //简单显示一句话的示例
    out.print("Hello world!");
%>
```

我们以这个例子来说明 JSP 文件的基本语法。JSP 文件以“.jsp”为扩展名，并将它放置到$CATALINA_HOME\webapps\test 目录下。

注意：$CATALINA_HOME 表示你的 Tomcat 的安装目录。如果你是按照第 3 章的方法安

装的，则$CATALINA_HOME 就等于 C:\java\tomcat-4.0.6，也就是说你应该把 helloworld.jsp 文件放到 C:\java\tomcat-4.0.6\webapps\test 目录下。以后我们还会用这种方法来表示路径，请注意按照自己的实际情况去寻找相应的路径。

4.1.2 代码分析

这里，我们以这个简单的 JSP 文件为例说明 JSP 的语法基本元素。

1. JSP 页面伪指令（JSP Page Directive）

JSP 页面伪指令为页面提供全局信息，如导入语句、错误处理页面或该页面是否为会话的一部分等，其基本格式为：

```
<%@ page att="val" %> 或<jsp:directive.page att="val"\>
```

在本例中 page 有两个属性，contentType="text/HTML"定义内容类型为 HTML 文本类型，而 charset=GB2312 定义本页使用 GB2312 为该页内码。虽然本例中的字符编码并没有实际作用，但如果页面中有中文时，这个标记是不可省略的。

除了 contentType 和 charset 属性以外，page 指令还可以包含以下属性：

- language="java"：声明该脚本语言的种类，暂时只能用"java" 。
- extends="package.class"：标明 JSP 编译时需要加入的 Java 类的全名，但是要慎重地使用它，因为它会限制 JSP 的编译能力。
- import="{package.class | package.* }, ..."：需要导入的 Java 包的列表，这些包就作用于程序段、表达式以及声明。
- session="{true|false}"：表明会话数据是否可供页面使用，默认为 true。
- buffer="{none|8kb|sizekb}"：确定输出流是否可以缓冲，默认设置为 8kb，并且与下边的 autoFlush 一起使用。
- autoFlush="{true|false}"：如果设置为 true，当缓冲区满时，刷新输出缓冲，而不是引发一个异常事件。默认设置为 true。
- isThreadSafe="{true|false}"：默认设置为 true，它会通知 JSP 引擎能够立刻处理多个用户的请求。同步状态是开发者的工作，以确保该页面是线程安全的。如果把 isThreadSafe 设为 false，那么只能使用控制客户访问网页的单线程模式。
- info="text"：可以通过页面上的 Servlet.getServletInfo()方法进行访问的网页上的信息。
- isErrorPage="{true|false}"：标记该页面是否为错误处理页面，和下边的 errorPage 一起使用。
- errorPage="path to error page"：给出处理异常事件的 JSP 页面的路径。如果设置此项，应保证该路径的有效性，并把该 JSP 页面的 isErrorPage 设为 true。

2. JSP 表达式（JSP Expression）

JSP 表达式的基本格式为：

```
<% 表达式 %>或者<jsp:expression>表达式</jsp:expression>
```

JSP 表达式用于计算并输出，它等价于<%out.print("表达式"); %>。其中的表达式元素表示的是一个在脚本语言中被定义的表达式，它在运行后自动转化为字符串，然后插入到这个表达式在 JSP 文件的位置显示。因为这个表达式的值已经被转化为字符串，所以可以在一行文本中插入这个表达式（形式和 ASP 完全一样）。

3. JSP 代码段（JSP Scriptlet）

JSP 代码段的基本格式为：

```
<% 代码 %>或<jsp:scriptlet>代码</jsp:scriptlet>
```

在 JSP 代码段中插入用于服务的代码。在这里代码和平常的 Java 代码完全一样，每条语句需要以分号结束。使用代码段可以像在普通 Java 程序中一样声明将要用到的变量或者调用方法。本例中就是通过一段代码段来完成“Hello world”的输出。这里的 out 对象是 JSP 的一个内置对象，它是 javax.jsp.JspWriter 的一个实例，并提供了几个方法使用于向浏览器回送输出结果。除了 out 以外，还有下边几个隐含对象可供使用。

- request：表示客户请求，是 HttpServletRequest 的子类。通常一个用户有请求的话，它将会包含有参数列表。
- response：表示 JSP 页面的响应，是 HttpServletResponse 的子类。
- pageContext：表示需要通过一个统一的 API 可以访问的页面属性和隐含对象。
- session：表示与请求相联系的 HTTP 会话对象。
- application：表示通过调用 getServletConfig().getContext()返回 Servlet 的环境。
- config：表示页面的 ServletConfig 对象。
- page：表示页面引用它自身的方法（相当于 Java 代码中的 this）。
- exception：表示传递到错误页面 URL 的没有捕获到的 Throwable 的子类。

4.1.3 运行结果

确保已经启动了 Tomcat，打开浏览器，并在地址栏中输入 http://localhost:8080/test/helloworld.jsp，就会看到页面上显示“Hello world!”，效果如图 4-1 所示。其中 8080 是端口号，浏览器默认访问 80 端口，而 Tomcat 的默认端口是 8080。要想修改这个端口号，在$CATALINA_HOME\conf\server.xml 中会看到有下边一条语句：

```
<Connector className="org.apache.catalina.connector.http.HttpConnector"
           port="8080" minProcessors="5" maxProcessors="75"
           enableLookups="true" redirectPort="8443"
acceptCount="10" debug="0" connectionTimeout="60000"/>
```

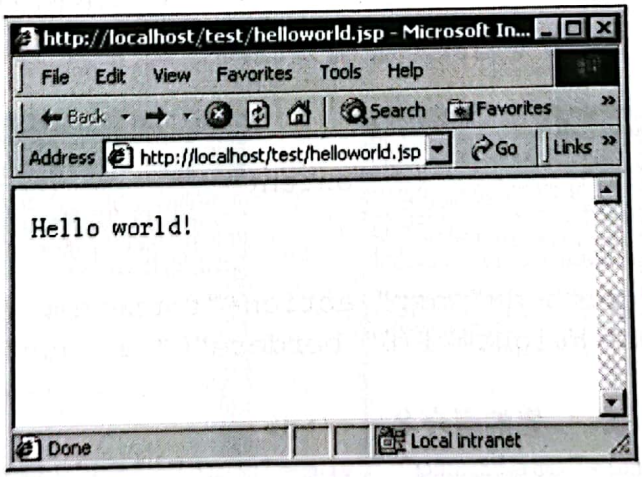

图 4-1 helloworld 显示页面

把 port="8080"中的 8080 改为 80。请确保你没有其他的程序占用 80 端口（如 IIS、Apache 等都是默认占用 80 端口），这样重新启动 Tomcat 之后，地址栏中就可以不输入端口号了。