第 3 章　第一个 JavaWeb 应用

本书第 2 章已经讲过，Tomcat 是符合 Servlet 规范的优秀 Servlet 容器。JavaWeb 应用运行在 Servlet 容器中，Servlet 容器能够动态调用 JavaWeb 应用中的 Servlet。

本章以一个简单的 helloapp 应用为例，让初学者迅速获得开发 JavaWeb 应用的实际经验。读者将通过这个例子学习以下内容：

- JavaWeb 应用的基本组成内容和目录结构。
- 在 web.xml 文件中配置 Servlet。
- 在 Tomcat 中发布 JavaWeb 应用。
- 配置 Tomcat 的虚拟主机。
- 创建、发布和使用自定义 JSP 标签。

本章侧重于介绍 JavaWeb 应用的组成和发布方法，所以没有对范例中的 Servlet 实现类以及 JSP 代码进行详细解释，关于 Servlet 和 JSP 的技术可以参考本书第 4 章、第 5 章和第 6 章的内容。

3.1　JavaWeb 应用简介

Oracle 公司的 Servlet 规范对 JavaWeb 应用做了这样的定义：“JavaWeb 应用由一组 Servlet/JSP、HTML 文件、相关 Java 类，以及其他可以被绑定的资源构成。它可以在由各种供应商提供的符合 Servlet 规范的 Servlet 容器中运行。”

从 JavaWeb 应用的定义可以看出，JavaWeb 应用不仅可以在 Tomcat 中运行，还可以在其他符合 Servlet 规范的 Servlet 容器中运行。在 JavaWeb 应用中可以包含如下内容：

- Servlet 组件

标准 Servlet 接口的实现类，运行在服务器端，包含了被 Servlet 容器动态调用的程序代码。

- JSP 组件

包含 Java 程序代码的 HTML 文档。运行在服务器端，当客户端请求访问 JSP 文件时，Servlet 容器先把它编译成 Servlet 类，然后动态调用它的程序代码。

- 相关的 Java 类

开发人员自定义的与 Web 应用相关的 Java 类。

- 静态文档

存放在服务器端的文件系统中，如 HTML 文件、图片文件和声音文件等。当客户端请求访问这些文件时，Servlet 容器从本地文件系统中读取这些文件的数据，再把它发送到客户端。

- 客户端脚本程序

是由客户端来运行的程序，JavaScript 是典型的客户端脚本程序，本书第 1 章的 1.6.3 节（提供浏览器端与用户的动态交互功能）已经介绍了它的运行机制。

- web.xml 文件

JavaWeb 应用的配置文件，该文件采用 XML 格式。该文件必须位于 Web 应用的 WEB-INF 子目录下。

Servlet 容器能够运行 JavaWeb 应用，而 JavaWeb 应用的最主要的组件就是 Servlet 和 JSP。因此 Servlet 容器也称为 JavaWeb 应用容器或者 Servlet/JSP 容器。

3.2 创建 JavaWeb 应用

JavaWeb 应用中可以包含 HTML 文档、Servlet、JSP 和相关 Java 类等。为了让 Servlet 容器能顺利地找到 JavaWeb 应用中的各个组件，Servlet 规范规定，JavaWeb 应用必须采用固定的目录结构，每种类型的组件在 Web 应用中都有固定的存放目录。Servlet 规范还规定，JavaWeb 应用的配置信息存放在 WEB-INF/web.xml 文件中，Servlet 容器从该文件中读取配置信息。在发布某些 Web 组件（如 Servlet）时，需要在 web.xml 文件中添加相应的关于这些 Web 组件的配置信息。

3.2.1 JavaWeb 应用的目录结构

JavaWeb 应用具有固定的目录结构，假定开发一个名为 helloapp 的 JavaWeb 应用。首先，应该创建这个 Web 应用的目录结构，参见表 3-1。

表 3-1 JavaWeb 应用的目录结构

目 录	描 述
/helloapp	Web 应用的根目录，所有的 JSP 和 HTML 文件都存放于此目录或子目录下。
/helloapp/WEB-INF	存放 Web 应用的配置文件 web.xml。
/helloapp/WEB-INF/classes	存放各种.class 文件，Servlet 类的.class 文件也放于此目录下。
/helloapp/WEB-INF/lib	存放 Web 应用所需的各种 JAR 文件（类库文件）。例如，在这个目录下可以存放 JDBC 驱动程序的 JAR 文件，参见本书第 8 章（访问数据库）。

从表 3-1 可以看出，在 WEB-INF 目录的 classes 以及 lib 子目录下，都可以存放 Java 类文件。在运行时，Servlet 容器的类加载器先加载 classes 目录下的类，再加载 lib 目录下的 JAR 文件（Java 类库的打包文件）中的类。因此，如果两个目录下存在同名的类，classes 目录下的类具有优先权。另外，浏览器端不可以直接请求访问 WEB-INF 目录下的文件，这些文件只能被服务器端的组件访问。

在开发阶段，为了便于调试和运行 helloapp 应用，可以直接在 Tomcat 的<CATALINA_HOME>/webapps 目录下创建该 Web 应用的目录结构。

本章介绍的 helloapp 应用的完整的目录结构如图 3-1 所示。

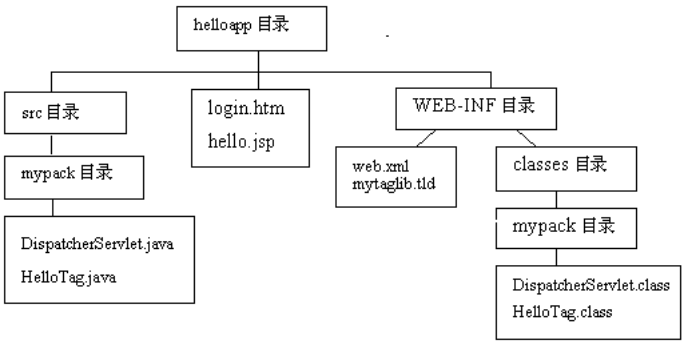

图 3-1　helloapp 应用的目录结构

图 3-1 中有一个 src 目录，这是在开发 helloapp 应用阶段，开发人员自定义的目录，该目录用来存放所有 Java 类的源文件。到了 Web 应用产品正式发布阶段，一般都不希望对外公开 Java 源代码，所以届时应该将 src 目录转移到其他地方。

在 helloapp 应用中包含如下组件：

- HTML 组件：login.htm
- Servlet 组件：DispatcherServlet 类
- JSP 组件：　hello.jsp

这些组件之间的关系如图 3-2 所示。login.htm 与 DispatcherServlet 类之间为超级链接关系，DispatcherServlet 类与 hello.jsp 之间为请求转发关系，本书第 5 章的 5.6.1 节（请求转发）对 Web 组件之间的请求转发关系作了深入介绍。

图 3-2　helloapp 应用中组件之间的关系

图 3-1 中还包含 HelloTag.java、mytaglib.tld 和 HelloTag.class 文件，这些文件用于创建、发布和使用自定义 JSP 标签，本章 3.4 节会对此作介绍。

3.2.2　创建 HTML 文件

在 helloapp 目录下加入 login.htm 文件（参见例程 3-1），它包含一个名为“loginForm”的登录表单，要求输入用户名和口令。当用户提交表单，将由 URI 为“dispatcher”的 Servlet 来处理，这个 Servlet 的 Java 类名为 mypack.DispatcherServlet，参见本章 3.2.3 节。

例程 3-1　login.htm

```
<html>
<head>
<title>helloapp</title>
</head>
<body >
  <form name="loginForm" method="POST" action="dispatcher">
    <table>
      <tr>
        <td><div align="right">User Name:</div></td>
        <td><input type="text" name="username"></td>
      </tr>
      <tr>
        <td><div align="right">Password:</div></td>
        <td><input type="password" name="password"></td>
      </tr>
      <tr>
        <td><input type="submit" name="submit" value="submit"></td>
        <td><input type="reset" name="reset" value="reset"></td>
      </tr>
    </table>
  </form>
</body>
</html>
```

访问 login.htm 的 URL 为 http://localhost:8080/helloapp/login.htm，该页面的显示结果如图 3-3 所示。

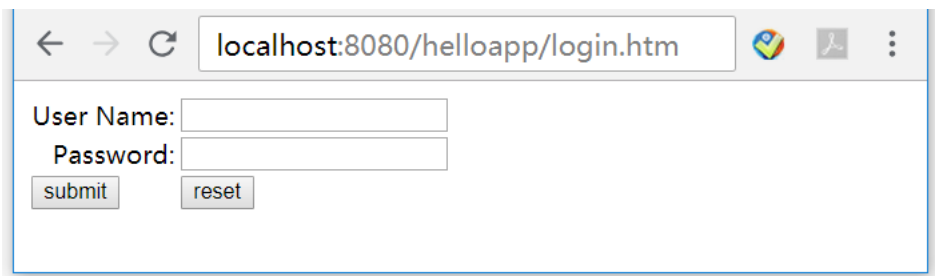

图 3-3　login.htm

3.2.3　创建 Servlet 类

下面创建 DispatcherServlet 类（参见例程 3-2），它调用 ServletRequest 对象的 getParameter()方法读取客户提交的 loginForm 表单数据，获取用户名和口令，然后将用户名和口令作为属性保存在 ServletRequest 对象中，再把请求转发给 hello.jsp。

例程 3-2　DispatcherServlet.java

```
package mypack;

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class DispatcherServlet extends GenericServlet {
  private String target = "/hello.jsp";

  /** 响应客户请求*/
  public void service(ServletRequest request,
                      ServletResponse response)
    throws ServletException, IOException {

    //读取表单中的用户名
    String username = request.getParameter("username");
    //读取表单中的口令
    String password = request.getParameter("password");

    //在 request 对象中添加 USER 属性
    request.setAttribute("USER", username);
    //在 request 对象中添加 PASSWORD 属性
    request.setAttribute("PASSWORD", password);

    /*把请求转发给 hello.jsp */
    ServletContext context = getServletContext();
    RequestDispatcher dispatcher =
               context.getRequestDispatcher(target);
    dispatcher.forward(request, response);
  }
}
```

编译 Servlet 时，需要将 Servlet API 的 JAR 文件（servlet-api.jar）加入到 classpath 中。servlet-api.jar 文件位于<CATALINA_HOME>/ lib 目录下。此外，从 Oracle 的官方网址（http://www.oracle.com/technetwork/java/index.html）也可以下载该文件。

在 DOS 下先转到 helloapp 目录下，然后用如下 javac 命令编译 DispacherServlet 类：

```
C:\chapter03\helloapp>javac   -sourcepath  src
                      -classpath  C:\tomcat\lib\servlet-api.jar
                      -d  WEB-INF\classes
                      src\mypack\DispatcherServlet.java
```

以上命令编译 DispatcherServlet.java 文件，生成的 DispatcherServlet.class 文件的存放位置为 helloapp/WEB-INF/classes/mypack/DispatcherServlet.class。

Java 编译器编译 Java 类时，编译出来的.class 文件的存放路径与类的包名存在对应关系。例如 DispatcherServlet 类声明位于 mypack 包下，因此 DispatcherServlet.class 文件的存放目录为 helloapp/WEB-INF/classes/mypack。

创建好 DispatcherServlet 后，还必须在 web.xml 文件中对其进行配置，这样客户端才能访问该 Servlet。本章 3.2.5 节介绍了对 DispatcherServlet 的配置。

3.2.4 创建 JSP 文件

接下来创建 hello.jsp，参见例程 3-3。hello.jsp 件放在 helloapp 根目录下。

例程 3-3 hello.jsp

```
<html>
<head>
  <title>helloapp</title>
</head>
<body>
  <b>Hello: <%= request.getAttribute("USER") %></b>
</body>
</html>
```

hello.jsp 和 HTML 文档看上去很相似，仅仅在一处地方使用了 JSP 语法：

```
<%= request.getAttribute("USER") %>
```

以上代码的作用是从 request（即 ServletRequest）对象中获得 USER 属性值，然后输出该 USER 属性值。本章 3.2.3 节介绍的 DispacherServlet 先在 request 对象中设置了 USER 属性，再把请求转发给 hello.jsp，所以接下来 hello.jsp 就能从 request 对象中获取 USER 属性值，hello.jsp 生成的页面如图 3-4 所示。JSP 的详细语法将在本书第 6 章（JSP 技术）详细讨论，本章侧重于介绍 JSP 的发布过程。

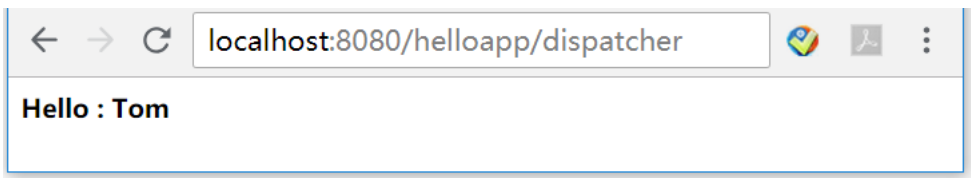

图 3-4 hello.jsp 生成的网页

3.2.5 创建 web.xml 文件

web.xml 文件是 JavaWeb 应用的 XML 格式的配置文件，存放于 WEB-INF 子目录下。web.xml 文件由开发人员编写，供 Servlet 容器访问。web.xml 文件也称为 JavaWeb 应用的发布描述符文件，Servlet 容器在加载和启动 JavaWeb 应用时会读取它的 web.xml 文件，从中获得关于当前 Web 应用的发布信息。本书附录 B 详细介绍了 web.xml 的配置方法，附录 C 则介绍了 XML 的基本知识。在 web.xml 文件中可包含如下配置信息：

- Servlet 的定义。

- Servlet 的初始化参数。
- Servlet 以及 JSP 的映射。
- 安全域配置参数。
- welcome 文件清单。
- 资源引用。
- 环境变量的定义。

现在，先创建一个默认的 web.xml 文件，并把这个文件放到 WEB-INF 目录中。

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
      http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  version="4.0" >

  ......
</web-app>
```

以上 web.xml 文件的第一行指定了 XML 的版本和字符编码，接下来声明了一个<web-app>元素，它是根元素，所有关于 JavaWeb 应用的具体配置元素都将加入到这个<web-app>元素中。

接下来在 web.xml 中为 DispatcherServlet 类加上<servlet>和<servlet-mapping>元素。

```
<web-app xmlns=…… >
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>mypack.DispatcherServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/dispatcher</url-pattern>
  </servlet-mapping>
</web-app>
```

<servlet>元素用于为 Servlet 类定义一个名字，它的子元素的说明参见表 3-2。在本例配置中，没有为 DispatcherServlet 设置<servlet>元素的<load-on-startup>子元素，因此当 Web 应用启动时，Servlet 容器不会初始化这个 Servlet，只有当客户端首次访问这个 Servlet 时，Servlet 容器才初始化它。本书第 4 章的 4.3.1 节（初始化阶段）在介绍 Servlet 生命周期时，进一步介绍了 Servlet 容器初始化 Servlet 的细节。

表 3-2　<servlet>元素的子元素

子元素	说　明
<servlet-name>	定义 Servlet 的名字。
<servlet-class>	指定 Servlet 的完整类名（包括包的名字）。
<init-param>	定义 Servlet 的初始化参数（包括参数名和参数值），一个<servlet>元素中可以有多个<init-param>。<init-param>的具体用法参见本书第 4 章的 4.1.8 节（ServletConfig 接口）。
<load-on-startup>	指定当 Servlet 容器启动 Web 应用时，加载各个 Servlet 的次序。当这个值为正数或零，Servlet 容器先加载数值小的 Servlet，再依次加载其他数值大的 Servlet。如果这个值为负数或者没有设定，那么 Servlet 容器将在客户端首次访问这个 Servlet 时加载它。

<servlet-mapping>元素用于为 Servlet 映射一个 URL。<servlet-name>子元素指定待映射的 Servlet 的名字；<url-pattern>子元素指定访问 Servlet 的相对 URL 路径。根据以上范例中的 <url-pattern> 子元素的值，可以确定访问 DispatcherServlet 的 URL 为 http://localhost:8080/helloapp/dispatcher。本章 3.3.3 节会介绍 Tomcat 根据 Servlet 的 URL 来查找相应的.class 文件的步骤。

在 web.xml 文件中还可以加入<welcome-file-list>元素，它用来为 Web 应用设置默认的主页。例如，当客户端提供的 URL 为 http://localhost:8080/helloapp 时，如果希望服务器端自动返回 login.htm 页面，则可以在 web.xml 文件中加入如下<welcome-file-list>元素：

```
<welcome-file-list>
  <welcome-file>login.htm </welcome-file>
</welcome-file-list>
```

本书附录 B 进一步介绍了<welcome-file-list>元素及<welcome-file>子元素的用法。

3.3　在 Tomcat 中发布 JavaWeb 应用

JavaWeb 应用可以运行在各种符合 Servlet 规范的 Servlet 容器中。本节介绍在 Tomcat 中发布 JavaWeb 应用的步骤。尽管发布的具体细节依赖于 Tomcat 本身的实现，但是以下关于发布 JavaWeb 应用的基本思想适用于所有 Servlet 容器：

- 把 Web 应用的所有文件拷贝到 Servlet 容器的特定目录下，这是发布 Web 应用的最快捷的一种方式。
- 各种 Servlet 容器实现都会从 Web 应用的 web.xml 配置文件中读取有关 Web 组件的配置信息。
- 为了使用户能更加灵活自如地控制 Servlet 容器发布和运行 Web 应用的行为，并且为了使 Servlet 容器与 Web 应用能进行更紧密地协作，许多 Servlet 容器还允许用户使用额外的配置文件及配置元素，这些配置文件及配置元素的语法由 Servlet 容器的实现决定，与 Oracle 公司的 Servlet 规范无关。

3.3.1 Tomcat 的目录结构

在 Tomcat 上发布 JavaWeb 应用之前，首先要了解 Tomcat 的目录结构。Tomcat 的目录结构是由自身的实现决定的，与 Oracle 公司的 Servlet 规范无关。Tomcat9.x 的目录结构参见表 3-3，表中的目录都是<CATALINA_HOME>的子目录。

表 3-3　Tomcat9.x 的目录结构

目　　录	描　　述
/bin	存放在 Window 平台以及 Linux 平台上启动和关闭 Tomcat 的脚本文件。
/conf	存放 Tomcat 服务器的各种配置文件，其中最重要的配置文件是 server.xml。
/lib	存放 Tomcat 服务器以及所有 Web 应用都可以访问的 JAR 文件。
/logs	存放 Tomcat 的日志文件。
/webapps	在 Tomcat 上发布 JavaWeb 应用时，默认情况下把 Web 应用文件放于此目录下。
/work	Tomcat 的工作目录，Tomcat 在运行时把生成的一些工作文件放于此目录下。例如默认情况下，Tomcat 把编译 JSP 而生成的 Servlet 类文件放于此目录下，参见本书第 6 章的 6.1.3 节（用 JSP 动态生成 HTML 页面）。

以上 lib 目录用于存放 JAR 文件，另外，本章 3.2.1 节已经提到，在 JavaWeb 应用的 WEB-INF 目录下也可以包含 lib 子目录。这两个 lib 目录的区别在于：

- Tomcat 的 lib 子目录：存放的 JAR 文件不仅能被 Tomcat 访问，还能被所有在 Tomcat 中发布的 JavaWeb 应用访问。
- JavaWeb 应用的 lib 子目录：存放的 JAR 文件只能被当前 JavaWeb 应用访问。

Tomcat 的类加载器负责为 Tomcat 本身以及 JavaWeb 应用加载相关的类。假如 Tomcat 的类加载器要为一个 JavaWeb 应用加载一个名为 Sample 的类，类加载器会按照以下先后顺序到各个目录中去查找 Sample 类的.class 文件，直到找到为止，如果所有目录中都不存在 Sample.class，则会抛出异常：

（1）在 JavaWeb 应用的 WEB-INF/classes 目录下查找 Sample.class 文件。
（2）在 JavaWeb 应用的 WEB-INF/lib 目录下的 JAR 文件中查找 Sample.class 文件。
（3）在 Tomcat 的 lib 子目录下直接查找 Sample.class 文件。
（4）在 Tomcat 的 lib 子目录下的 JAR 文件中查找 Sample.class 文件。

如果想进一步了解 Tomcat 的类加载器的工作过程，请参考 Tomcat 的有关文档，地址为：<CATALINA_HOME>/webapps/docs/class-loader-howto.html。

3.3.2 按照默认方式发布 JavaWeb 应用

在 Tomcat 中发布 JavaWeb 应用的最快捷的方式，就是直接把 JavaWeb 应用的所有文件拷贝到 Tomcat 的<CATALINA_HOME>/webapps 目录下。默认情况下，<CATALINA_HOME>/webapps 中的所有 Web 应用运行在名为 localhost 的虚拟主机中，而 localhost 虚拟主机则运行在名为 Catalina 的 Engine 组件中。

Tomcat 既可以运行采用开放式目录结构的 Web 应用，也可以运行 Web 应用的打包文件（简称为 WAR 文件）。

在 sourcecode/chapter03/helloapp 目录下提供了本章范例的所有源文件，只要把整个 helloapp 目录拷贝到<CATALINA_HOME>/webapps 目录下，即可运行开放式目录结构的 helloapp 应用。

在 Web 应用的开发阶段，为了便于调试，通常采用开放式的目录结构来发布 Web 应用，这样可以方便地更新或替换文件。如果开发完毕，进入产品发布阶段，应该将整个 Web 应用打包为 WAR 文件，再进行发布。在本例中，按如下步骤发布 helloapp。

（1）在 DOS 中进入 helloapp 应用的根目录 helloapp。

（2）把整个 Web 应用打包为 helloapp.war 文件，命令如下：

```
C:\chapter03\helloapp>jar cvf  C:\chapter03\helloapp.war  *
```

以上 jar 命令会把 C:\chapter03\helloapp 目录下（包括其子目录下）的所有文件打包为 helloapp.war 文件，把该文件存放在 C:\chapter03 目录下。

在 JDK 的 bin 目录下提供了打包程序 jar.exe。如果要展开 helloapp.war 文件，命令为：jar　xvf　C:\chapter03\helloapp.war。

（3）如果在 Tomcat 的 webapps 目录下已经有 helloapp 子目录，先将 helloapp 子目录删除。

（4）把 helloapp.war 文件拷贝到<CATALINA_HOME>/webapps 目录下。

（5）启动 Tomcat 服务器。

Tomcat 服务器启动时，会把 webapps 目录下的所有 WAR 文件自动展开为开放式的目录结构。因此服务器启动后，读者会发现服务器自动在 webapps 目录下创建了一个 helloapp 子目录，并把 helloapp.war 中的所有内容展开到 helloapp 子目录中。

3.3.3　Web 组件的 URL

无论按照开放式目录结构还是按照打包文件方式发布 Web 应用，Web 应用的默认 URL 入口都是 Web 应用的根目录名。例如对于 helloapp 应用，它的 URL 入口为“/helloapp”。

对于 HTML 或 JSP 文件，它们的 URL 与文件路径之间存在对应关系。例如 login.htm 的文件路径为 helloapp/login.htm，因此它的 URL 为 http://localhost:8080/helloapp/login.htm。同样，hello.jsp 的文件路径为 helloapp/hello.jsp，因此它的 URL 为 http://localhost:8080/helloapp/hello.jsp。

HTML 或 JSP 文件不仅可以放在 Web 应用的根目录下，也可以放到自定义的子目录下。例如，假定 hello.jsp 的文件路径为 helloapp/dir1/dir2/hello.jsp，那么它的 URL 为：http://localhost:8080/helloapp/dir1/dir2/hello.jsp。

浏览器无法直接访问 Web 应用的 WEB-INF 目录下的文件。因此，如果出于安全的原因，不希望浏览器直接访问某个 JSP 文件，可以把它放到 WEB-INF 目录或其子目录下。在这种情况下，只有服务器端的组件才能访问该 JSP 文件，例如 Servlet 可以给把请求转发给 JSP 文件。

对于 Servlet，对其映射 URL 有两种方式：

- 在 web.xml 文件中映射 URL。
- 用@WebServlet 标注来映射 URL，参见第 4 章的 4.7 节（ 使用 Annotation 标注配置 Servlet）。

本节介绍如何在 web.xml 中配置 Servlet。Servlet 的 URL 由 web.xml 文件中的<url-pattern>元素来指定。在本范例中，web.xml 文件对 mypack.DispatcherServlet 类作了如下配置：

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>mypack.DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/dispatcher</url-pattern>
</servlet-mapping>
```

以上<url-pattern>元素的值为“/dispatcher”，因此访问 mypack.DispatcherServlet 类的 URL 为：http://localhost:8080/helloapp/dispatcher。当浏览器端首次通过该 URL 请求访问 DispatcherServlet 时，Tomcat 需要先找到文件系统中的 DispatcherServlet.class 文件，从而能加载 DispatcherServlet 类。Tomcat 查找 DispatcherServlet.class 文件的步骤如下。

（1）参考 helloapp 应用的 web.xml 文件，找到<url-pattern>子元素的值为“/dispatcher”的<servlet-mapping>元素。

（2）读取<servlet-mapping>元素的<servlet-name>子元素的值，由此确定 Servlet 的名字为 dispatcher。

（3）找到<servlet-name>子元素的值为“dispatcher”的<servlet>元素。

（4）读取<servlet>元素的<servlet-class>子元素的值，由此确定 Servlet 的类名为 mypack.DispatcherServlet。

（5）到<CATALINA_HOME>/webapps/helloapp/WEB-INF/classes/mypack 目录下查找 DispatcherServlet.class 文件。

以下图 3-5 显示了 DispatcherServlet 类的 URL、Servlet 配置元素以及 DispatcherServlet.class 文件之间的对应关系。

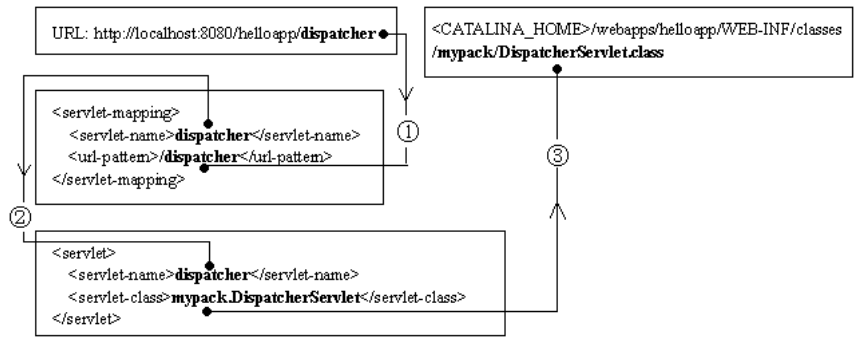

图 3-5　DispatcherServlet 的 URL、配置元素以及.class 文件之间的对应关系

Tomcat 在加载 Web 应用时，就会把相应的 web.xml 文件中的数据读入到内存中。因此当 Tomcat 需要参考 web.xml 文件时，实际上只需要从内存中读取相关数据就行了，无需再到文件系统中读取 web.xml 文件。

初学者发布了 Servlet 后，再通过浏览器访问该 Servlet 时，服务器端可能会返回“该文件不存在”的错误。这可能是由以下原因导致的：

（1）提供的 URL 不正确，例如以下 URL 都无法访问 DispatcherServlet 类：

```
http://localhost:8080/helloapp/DispatcherServlet.class
http://localhost:8080/helloapp/DispatcherServlet
http://localhost:8080/helloapp/mypack/DispatcherServlet.class
http://localhost:8080/dispatcher
```

（2）web.xml 作为 XML 格式的文件，需要区分大小写。如果不注意大小写，可能会导致对 Servlet 的配置不正确。例如以下<servlet>元素把 mypack.DispatcherServlet 类命名为“Dispatcher”。而<servlet-mapping>元素对一个名为“dispatcher”的 Servlet 进行了 URL 映射。这使得名为“Dispatcher”的 Servlet 实际上没有进行 URL 映射。

```
<servlet>
  <servlet-name>Dispatcher</servlet-name>
  <servlet-class>mypack.DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/dispatcher</url-pattern>
</servlet-mapping>
```

（3）Servlet 的.class 文件的存放路径不正确。Servlet 的.class 文件必须位于 Web 应用的 WEB-INF/classes 目录下，并且类的包名与文件路径匹配。例如对于一个完整类名为 mypack1.mypack2.MyServlet 的类，它的文件路径应该为：

```
WEB-INF/classes/mypack1/mypack2/MyServlet.class
```

也许你会问：Servlet 规范为什么规定要对 Servlet 进行 URL 映射呢？如果能直接根据 Servlet 的文件路径来访问 Servlet，不是更方便吗？假如这种设想成立，访问 DispatherServlet 类的 URL 将变为：

```
http://localhost:8080/helloapp/mypack/DispatcherServlet.class
```

Servlet 规范之所以规定要对 Servlet 进行 URL 映射，主要有两个原因：

（1）为一个 Servlet 对应多个 URL 提供了方便的设置途径。假如有个 Web 应用规定所有以“.DO”结尾的 URL 都由 ActionServlet 来处理，那么只需在 web.xml 文件中对 ActionServlet 进行如下配置：

```
<servlet>
  <servlet-name>action</servlet-name>
  <servlet-class>mypack1.mypack2.ActionServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

这样，所有以“.DO”结尾的 URL 都对应 ActionServlet，例如以下 URL 都映射到 ActionServlet：

```
http://localhost:8080/mywebapp/login.DO
http://localhost:8080/mywebapp/logout.DO
http://localhost:8080/mywebapp/checkout.DO
```

再例如一个<servlet>还可以对应多个<servlet-mapping>元素：

```
<servlet>
  <servlet-name>Manager</servlet-name>
  <servlet-class>
    org.apache.catalina.manager.ManagerServlet
  </servlet-class>
  <init-param>
    <param-name>debug</param-name>
    <param-value>2</param-value>
  </init-param>
</servlet>

<servlet-mapping>
  <servlet-name>Manager</servlet-name>
  <url-pattern>/list</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>Manager</servlet-name>
```

```
    <url-pattern>/expire</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>Manager</servlet-name>
    <url-pattern>/sessions</url-pattern>
  </servlet-mapping>

  <servlet-mapping>
    <servlet-name>Manager</servlet-name>
    <url-pattern>/start</url-pattern>
  </servlet-mapping>
```

（2）简化 Servlet 的 URL，并且可以向客户端隐藏 Web 应用的实现细节。如果在 URL 中暴露 Servlet 的完整类名，会让不懂 JavaWeb 开发的普通客户觉得 URL 很复杂，不容易理解和记忆。通过对 Servlet 进行 URL 映射，则可以提供一个简洁易懂的 URL。

JSP 文件和 Servlet 一样，也可以在 web.xml 文件中映射 URL。本书第 6 章的 6.7 节（再谈发布 JSP）对此作了介绍。

3.3.4 配置 Tomcat 的<Context>元素

本章 3.3.2 节已经介绍了在 Tomcat 中发布 JavaWeb 应用的最快捷的方式，只需把 JavaWeb 应用的所有文件拷贝到<CATALINA_HOME>/webapps 目录下即可，Tomcat 会按照默认的方式来发布和运行 JavaWeb 应用。如果需要更加灵活地发布 Web 应用，则需要为 Web 应用配置 Tomcat 的<Context>元素。

<Context>元素是 Tomcat 中使用最频繁的元素，它代表了运行在虚拟主机<Host>上的单个 Web 应用。本书第 2 章的 2.3 节（Tomcat 的组成结构）在介绍 Tomcat 的组成结构时已经简单介绍了<Context>元素、<Host>元素和<Engine>元素。一个<Engine>中可以有多个<Host>，一个<Host>中可以有多个<Context>。<Context>元素的主要属性的说明参见表 3-4。

表 3-4　Context 元素的主要属性

属　性	描　述
path	指定访问该 Web 应用的 URL 入口。
docBase	指定 Web 应用的文件路径，可以给定绝对路径，也可以给定相对于<Host>的 appBase 属性的相对路径（关于<Host>的 appBase 属性参见本章 3.3.5 节）。如果 Web 应用采用开放目录结构，则指定 Web 应用的根目录；如果 Web 应用是个 WAR 文件，则指定 WAR 文件的路径。
className	指定实现 Context 组件的 Java 类的名字，这个 Java 类必须实现 org.apache.catalina.Context 接口。该属性的默认值为 org.apache.catalina.core.StandardContext。
reloadable	如果这个属性设为 true，Tomcat 服务器在运行状态下会监视在 WEB-INF/classes 和 WEB-INF/lib 目录下.class 类文件或.jar 类库文件的改动。如果监测到有类文件或类库文件按被更新，服务器会自动重新加载 Web 应用。该属性的默认值为 false。在 Web 应用的开发和调试阶段，把 reloadable 设为 true，可以方便对 Web 应用的调试。在 Web 应用正式发布阶段，把 reloadable 设为 false，可以降低 Tomcat 的运行负荷，提高 Tomcat 的运行性能。

一般情况下，<Context>元素都会使用默认的标准 Context 组件，即 className 属性采用默认值 org.apache.catalina.core.StandardContext。标准 Context 组件除了具有表 3-4 列出的属性，还具有以下表 3-5 所示的属性。

表 3-5　标准 Context 组件的专有属性

属　性	描　述
unloadDelay	设定 Tomcat 等待 Servlet 卸载的毫秒数。该属性的默认值为 2000 毫秒。
workDir	指定 Web 应用的工作目录。Tomcat 运行时会把与这个 Web 应用相关的临时文件放在此目录下。
uppackWar	如果此项设为 true，表示将把 Web 应用的 WAR 文件先展开为开放目录结构后再运行。如果设为 false，则直接运行 WAR 文件。该属性的默认值为 true。

在 Tomcat 低版本中，允许直接在<CATALINA_HOME>/conf/server.xml 文件中配置<Context>元素。这种配置方式有一个弊端：如果在 Tomcat 运行时修改 server.xml 文件，比如添加<Context>元素，那么所作的修改不会立即生效，而必须重新启动 Tomcat，才能使所作的修改生效。

因此 Tomcat6.x 开始的高版本尽管也允许直接在 server.xml 文件中配置<Context>元素，但不提倡采用这种方式。如今的 Tomcat 提供了多种配置<Context>元素的途径。当 Tomcat 加载一个 Web 应用时，会按照以下顺序查找 Web 应用的<Context>元素：

（1）到<CATALINA_HOME>/conf/context.xml 文件中查找<Context>元素。这个文件中的<Context>元素的信息适用于所有 Web 应用。

（2）到<CATALINA_HOME>/conf/[enginename]/[hostname]/context.xml.default 文件中查找<Context>元素。[enginename]表示<Engine>的 name 属性，[hostname]表示<Host>的 name 属性。在 context.xml.default 文件中的<Context>元素的信息适用于当前虚拟主机中的所有 Web 应用。 例如以下文件中的<Context>元素适用于名为 Catalina 的 Engine 下的 localhost 主机中的所有 Web 应用：

```
<CATALINA_HOME>/conf/Catalina/localhost/context.xml.default
```

（3）到<CATALINA_HOME>/conf/[enginename]/[hostname]/[contextpath].xml 文件中查找<Context>元素。[contextpath]表示单个 Web 应用的 URL 入口。在[contextpath].xml 文件中的<Context>元素的信息只适用于单个 Web 应用。例如以下文件中的<Context>元素适用于名为 Catalina 的 Engine 下的 localhost 主机中的 helloapp 应用：

```
<CATALINA_HOME>/conf/Catalina/localhost/helloapp.xml
```

（4）到 Web 应用的 META-INF/context.xml 文件中查找<Context>元素。这个文件中的<Context>元素的信息适用于当前 Web 应用。

（5）到<CATALINA_HOME>/conf/server.xml 文件中的<Host>元素中查找<Context>子元素。该<Context>元素的信息只适用于单个 Web 应用。

如果仅仅为单个 Web 应用配置<Context>元素，可以优先选择第三种或第四种方式。第三种方式要求在 Tomcat 的相关目录下增加一个包含<Context>元素的配置文件，而第四种方式则要求在 Web 应用的相关目录下增加一个包含<Context>元素的配置文件。对于这两种

方式，Tomcat 在运行时会监测包含<Context>元素的配置文件是否被更新，如果被更新，Tomcat 会自动重新加载并启动 Web 应用，使对<Context>元素所作的修改生效。

下面先采用第四种方式配置<Context>元素。在 helloapp 目录下新建一个 META-INF 子目录，然后在其中创建一个 context.xml 文件，它的内容如下：

```
<Context path="/helloapp" docBase="helloapp" reloadable="true" />
```

以上<Context>元素的 docBase 属性表明，helloapp 应用的文件路径为<CATALINA_HOME>/webapps/helloapp；path 属性表明访问 helloapp 应用的 URL 入口为“/helloapp”。

下面再采用第三种方式配置<Context>元素。假定 helloapp 应用的文件路径为 C:\chapter03\helloapp，并且在<CATALINA_HOME>/webapps 目录下没有发布 helloapp 应用。在<CATALINA_HOME>/conf 目录下先创建 Catalina 目录，接着在 Catalina 目录下再创建 localhost 目录，然后在<CATALINA_HOME>/conf/Catalina/localhost 目录下创建 helloapp.xml 文件，它的内容如下：

```
<Context path="/helloapp"
       docBase="C:\chapter03\helloapp"
       reloadable="true" />
```

以上<Context>元素的 docBase 属性指定了 helloapp 应用的绝对路径，为 C:\chapter03\helloapp；path 属性表明访问 helloapp 应用的 URL 入口为“/helloapp”。由于 helloapp.xml 文件位于 Catalina/localhost/子目录下，因此 helloapp 应用将运行在名为 Catalina 的 Engine 组件的 localhost 虚拟主机中。访问 helloapp 应用中的 login.htm 和 hello.jsp 的 URL 分别为：

```
http://localhost:8080/helloapp/login.htm
http://localhost:8080/helloapp/hello.jsp
```

在 server.xml 文件中已经有一个名为 localhost 的<Host>元素，如果采用第五种方式配置<Context>元素，最常见的做法是在该<Host>元素中插入<Context>子元素，例如：

```
<Host name="localhost"  appBase="webapps"
          unpackWARs="true" autoDeploy="true"
          xmlValidation="false" xmlNamespaceAware="false">
        …
  <Context path="/helloapp" docBase="helloapp" reloadable="true" />
</Host>
```

如果没有为 Web 应用配置 Tomcat 的 Context 元素，那么 Tomcat 会为 Web 应用提供一个默认的 Context 组件。例如按照本章 3.3.2 节的方式发布 helloapp 应用时，Tomcat 就给它提供了默认的 Context 组件。

3.3.5 配置 Tomcat 的虚拟主机

在 Tomcat 的配置文件 server.xml 中，<Host>元素代表虚拟主机，在同一个<Engine>元

素下可以配置多个虚拟主机。例如，有两个公司的 Web 应用都发布在同一个 Tomcat 服务器上，可以为每家公司分别创建一个虚拟主机，它们的虚拟主机名分别为：

```
www.javathinkerok.com
www.javathinkerext.com
```

尽管以上两个虚拟主机实际上对应同一个主机，但是当客户端通过以上两个不同的虚拟主机名访问 Web 应用时，客户端会感觉这两个应用分别拥有独立的主机。

此外，还可以为虚拟主机建立别名，例如，如果希望客户端访问 www.javathinkerok.com 或 javathinkerok.com 都能对应到同一个 Web 应用，那么可以把 javathinkerok.com 作为虚拟主机的别名来处理。

下面介绍如何配置 www.javathinkerok.com 虚拟主机。

（1）打开<CATALINA_HOME>/conf/server.xml 文件，会发现在<Engine>元素中已经有一个名为 localhost 的<Host >元素，可以在它的后面（即</Host>标记后面）加入如下<Host>元素：

```
<Host name="www.javathinkerok.com" appBase="C:\javathinkerok"
unpackWARs="true" autoDeploy="true">

  <Alias>javathinkerok.com</Alias>
  <Alias>javathinkerok</Alias>

</Host>
```

以上配置代码位于 sourcecode/chapter03/virtualhost-configure.xml 文件中。<Host>元素的属性描述参见表 3-6。<Host>元素还有一个子元素<Alias>，它用于指定虚拟主机的别名。<Host>元素允许包含多个<Alias>子元素，因此可以指定多个别名。

表 3-6　<Host>元素的属性

属　性	描　述
name	指定虚拟主机的名字。
className	指定实现虚拟主机的 Java 类的名字，这个 Java 类必须实现 org.apache.catalina.Host 接口。该属性的默认值为 org.apache.catalina.core.StandardHost。
appBase	指定虚拟主机的目录，可以指定绝对目录，也可以指定相对于<CATALINA_HOME>的相对目录。如果此项没有设定，默认值为<CATALINA_HOME>/webapps。
autoDeploy	如果此项设为 true，表示当 Tomcat 服务器处于运行状态时，能够监测 appBase 下的文件，如果有新的 Web 应用加入进来，则会自动发布这个 Web 应用。
deployOnStartup	如果此项设为 true，则表示 Tomcat 启动时会自动发布 appBase 目录下所有的 Web 应用。如果 Web 应用没有相应的<Context>元素，那么 Tomcat 会提供一个默认的 Context 组件。deployOnStartup 的默认值为 true。

一般情况下，<Host>元素都会使用默认的标准虚拟主机，即 className 属性使用默认值 org.apache.catalina.core.StandardHost。标准虚拟主机除了具有表 3-6 列出的属性，还具有以下表 3-7 所示的属性。

表 3-7　标准虚拟主机的专有属性

属　性	描　述
unpackWARS	如果此项设为 true，表示将把 appBase 属性指定的目录下的 Web 应用的 WAR 文件先展开为开放目录结构后再运行。如果设为 false，则直接运行 WAR 文件。
workDir	指定虚拟主机的工作目录。Tomcat 运行时会把与这个虚拟主机的所有 Web 应用相关的临时文件放在此目录下。它的默认值为<CATALINA_HOME>/work。如果<Host>元素下的一个<Context>元素也设置了 workDir 属性，那么<Context>元素的 workDir 属性会覆盖<Host>元素的 workDir 属性。
deployXML	如果设为 false，那么 Tomcat 不会解析 Web 应用中的用于设置 Context 元素的 META-INF/context.xml 文件。出于安全原因，如果不希望 Web 应用中包含 Tomcat 的配置元素，就可以把这个属性设为 false，在这种情况下，应该在<CATALINA_HOME>/conf/[enginename]/[hostname]下设置 Context 元素。该属性的默认值为 true。

（2）把 helloapp 应用（helloapp.war 文件或者是整个 helloapp 目录）拷贝到<Host>元素的 appBase 属性指定的目录 C:\javathinkerok 下。

（3）为了使以上配置的虚拟主机生效，必须在 DNS 服务器中注册以上虚拟主机名和别名，使它们和 Tomcat 服务器所在的主机的 IP 地址进行映射。本节末尾会介绍在 Windows 中配置 DNS 映射的步骤。必须在 DNS 服务器中注册以下虚拟主机名字和别名：

```
www.javathinkerok.com
javathinkerok.com
javathinkerok
```

（4）重启 Tomcat 服务器，然后通过浏览器访问：

```
http://www.javathinkerok.com:8080/helloapp/login.htm
```

如果返回正常的页面就说明配置成功。还可以通过虚拟机的别名来访问 helloapp 应用：

```
http://javathinkerok.com:8080/helloapp/login.htm
http://javathinkerok:8080/helloapp/login.htm
```

每个虚拟主机都可以有一个默认 Web 应用，它的默认根目录为 ROOT。例如在<CATALINA_HOME>/webapps 目录下有一个 ROOT 目录，它是 localhost 虚拟主机的默认 Web 应用，访问 http://localhost:8080/index.jsp，就会显示这个 Web 应用的 index.jsp 页面。

对于 www.javathinkerok.com 虚拟主机，也可以提供默认的 Web 应用。把 C:\javathinkerok 下的 helloapp 目录改名为 ROOT 目录，这个虚拟主机就有了一个默认 Web 应用。访问 http://www.javathinkerok.com:8080/login.htm，就会显示这个 Web 应用的 login.htm 页面。

如果要设置虚拟主机的默认 Web 应用的<Context>元素，那么它的 path 属性的值应该为一个空的字符串（即 path=""）。

如果要了解更多关于配置 Tomcat 的虚拟主机的信息，可以参考 Tomcat 的相关文档，地址为：<CATALINA_HOME>/webapps/docs/virtual-hosting-howto.html

以上步骤(3)提到要进行虚拟主机名和 IP 地址之间的 DNS 映射。下面介绍在 Windows 操作系统中进行 DNS 映射的步骤。

（1）在文件资源管理器中找到文件：C:\WINDOS\system32\drivers\etc\hosts。选中 hosts 文件，按下鼠标右键，在弹出的菜单中，选择【属性】→【安全】→【编辑】，在 hosts 文件的访问权限编辑窗口中，设置 Windows 用户具有“完全控制”权限，参见图 3-6.

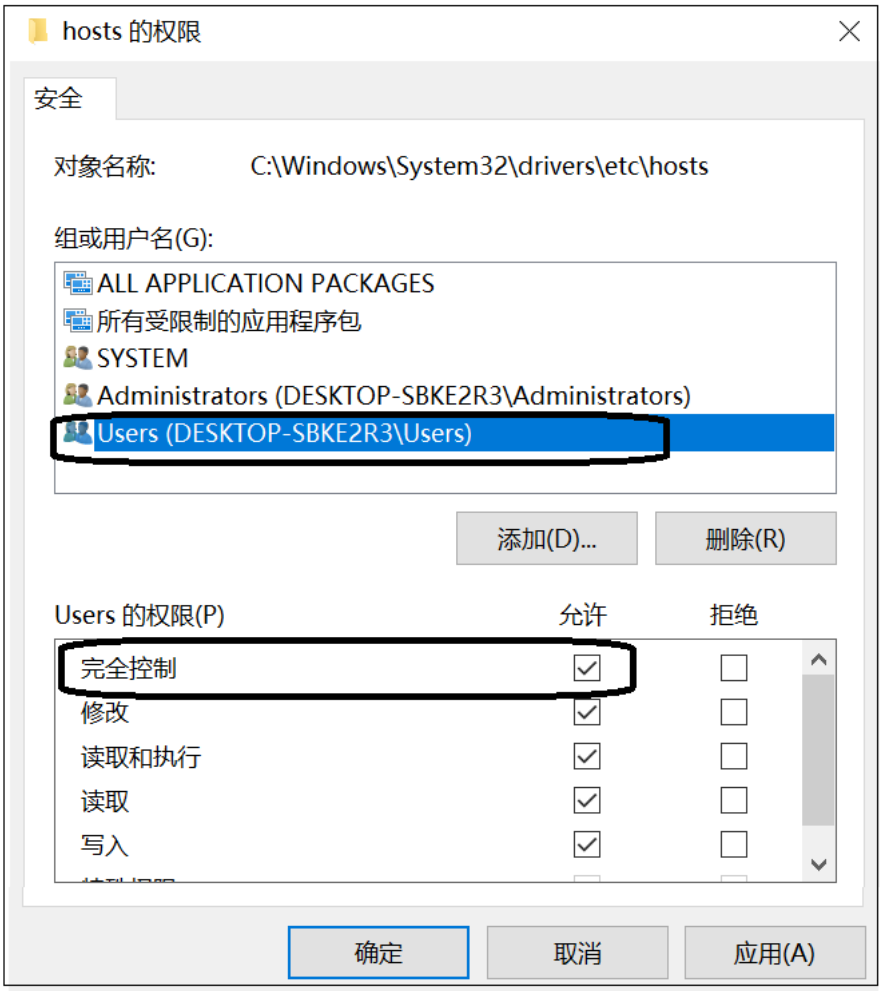

图 3-6　编辑 hosts 文件的访问权限

（2）用 Windows 记事本打开 hosts 文件，在文件中加入如下内容，使得“www.javathinkerok.com”等虚拟主机名和本地主机的 IP 地址映射：

```
127.0.0.1   www.javathinkerok.com
127.0.0.1   javathinkerok.com
127.0.0.1   javathinkerok

::1   www.javathinkerok.com
::1   javathinkerok.com
```

```
::1   javathinkerok
```

以上代码中“127.0.0.1”和“::1”分别是本地主机的 IPv4 格式和 IPv6 格式的 IP 地址。

3.4 创建、配置和使用自定义 JSP 标签

接下来创建一个名为 hello 的简单的自定义 JSP 标签，它的作用是输出字符串“Hello”。hello 标签位于一个名为 mytaglib 的标签库（Tag Library）中。hello.jsp 会使用 mytaglib 标签库中的 hello 标签。本章侧重介绍 hello 标签的发布和使用。本书第 13 章（自定义 JSP 标签）还会更深入地介绍创建自定义 JSP 标签的过程。

以下是创建、配置和使用 hello 标签的步骤。

（1）编写用于处理 hello 标签的类，名为 HelloTag 类，例程 3-4 列出了 HelloTag.java 的源代码。

例程 3-4　HelloTag.java

```
package mypack;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspTagException;
import javax.servlet.jsp.tagext.TagSupport;

public class HelloTag extends TagSupport{
  /** 当 JSP 解析器遇到 hello 标签的结束标志时，调用此方法 */
  public int doEndTag() throws JspException{
    try{
      //打印字符串“Hello”
      pageContext.getOut().print("Hello");
    }catch (Exception e) {
      throw new JspTagException(e.getMessage());
    }
    return EVAL_PAGE;
  }
}
```

编译 HelloTag.java 时，需要将 Servlet API 的类库文件（servlet-api.jar）以及 JSP API 的类库文件（jsp-api.jar）添加到 classpath 中，这两个 JAR 文件位于<CATALINA_HOME>/lib 目录下。此外，在 Oracle 的官方网址（http://www.oracle.com/technetwork/java/index.html）也可以下载 JSP API 的类库文件。编译生成的 HelloTag.class 存放位置为 WEB-INF/classes/mypack/HelloTag.class。

（2）创建一个 TLD（Tag Library Descriptor，标签库描述符）文件。假定 hello 标签位于 mytaglib 标签库中，因此创建一个名为 mytaglib.tld 的 TLD 文件。在这个文件中定义

mytaglib 标签库和 hello 标签。这个文件的存放位置为 WEB-INF/mytaglib.tld。例程 3-5 列出了 mytaglib.tld 的源代码。

例程 3-5　mytaglib.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
       PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
    "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_2.dtd">

<!-- a tag library descriptor -->

<taglib>
  <tlib-version>1.1</tlib-version>
  <jsp-version>2.4</jsp-version>
  <short-name>mytaglib</short-name>
  <uri>/mytaglib</uri>

  <tag>
    <name>hello</name>
    <tag-class>mypack.HelloTag</tag-class>
    <body-content>empty</body-content>
    <description>Just Says Hello</description>
  </tag>

</taglib>
```

Servlet 规范规定，TLD 文件在 Web 应用中必须存放在 WEB-INF 目录或者自定义的子目录下，但不能放在 WEB-INF\classes 目录和 WEB-INF\lib 目录下。web.xml 文件中的<taglib>元素的<taglib-location>子元素用来设置标签库描述文件的存放路径，应该保证<taglib-location>子元素的取值与 TLD 文件的实际存放位置相符。

（3）在 web.xml 文件中配置<taglib>元素，例程 3-6 列出了修改后的 web.xml 文件。

例程 3-6　加入<taglib>元素的 web.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
      http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  version="4.0"
  metadata-complete="true">
```

```
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>mypack.DispatcherServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/dispatcher</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>login.htm </welcome-file>
  </welcome-file-list>

  <jsp-config>
    <taglib>
      <taglib-uri>
        /mytaglib
      </taglib-uri>

      <taglib-location>
         /WEB-INF/mytaglib.tld
      </taglib-location>
    </taglib>
  </jsp-config>

</web-app>
```

<taglib>元素位于<jsp-config>元素中。<taglib>元素中包含两个子元素：<taglib-uri>和<taglib-location>。其中<taglib-uri>指定标签库的 URI；<taglib-location>指定标签库的 TLD 文件的存放位置。

（4）在 hello.jsp 文件中使用 hello 标签。首先，在 hello.jsp 中加入引用 mytaglib 标签库的 taglib 标签的指令：

```
<%@ taglib uri="/mytaglib" prefix="mm" %>
```

以上 taglib 指令中，prefix 属性用来为 mytaglib 标签库指定一个前缀“mm”。接下来，hello.jsp 就可以用<mm:hello/>的形式来使用 hello 标签。修改后的 hello.jsp 文件参见例程 3-7。

例程 3-7　加入 hello 标签的 hello.jsp

```
<%@ taglib uri="/mytaglib" prefix="mm" %>
<html>
<head>
  <title>helloapp</title>
</head>
```

```
    <b><mm:hello/> : <%= request.getAttribute("USER") %></b>
  </body>
  </html>
```

hello.jsp 修改后，再依次访问 login.htm→DispatcherServlet→hello.jsp，最后生成的网页如图 3-7 所示。这个网页中的“Hello”字符串就是由<mm:hello/>标签输出来的。

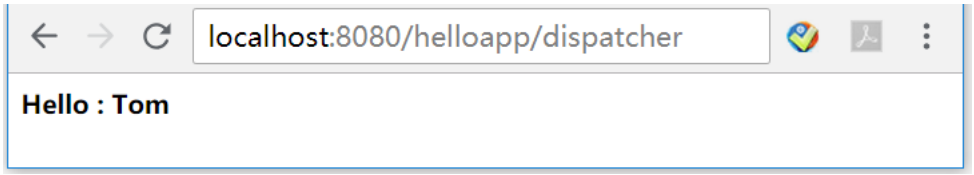

图 3-7　带 hello 标签的 hello.jsp 生成的网页

当客户端请求访问 hello.jsp 时，Servlet 容器会按照如下步骤处理 hello.jsp 中的<mm:hello/>标签。

（1）由于<mm:hello/>的前缀为“mm”，与 hello.jsp 中的如下 taglib 指令匹配：

```
<%@ taglib uri="/mytaglib" prefix="mm" %>
```

由此得知 hello 标签来自 URI 为“/mytaglib”的标签库。

（2）在 web.xml 文件中对 URI 为“/mytaglib”的标签库的配置如下：

```
<taglib>
  <taglib-uri>/mytaglib</taglib-uri>
  <taglib-location>/WEB-INF/mytaglib.tld</taglib-location>
</taglib>
```

由此得知 URI 为“/mytaglib”的标签库的 TLD 文件为 WEB-INF/mytaglib.tld。

（3）在 WEB-INF/mytaglib.tld 文件中对名为 hello 的标签的定义如下：

```
<tag>
  <name>hello</name>
  <tagclass>mypack.HelloTag</tagclass>
  <bodycontent>empty</bodycontent>
  <info>Just Says Hello</info>
</tag>
```

由此得知 hello 标签的处理类为 mypack.HelloTag 类。因此当 Servlet 容器运行 hello.jsp 时，如果遇到<mm:hello/>标签，就会加载 WEB-INF/classes/mypack 目录下的 HelloTag.class 文件。遇到<mm:hello/>标签的结束标志时，就会调用 HelloTag 类的 doEndTag()方法。

以下图 3-8 显示了 hello.jsp 中的 hello 标签与 web.xml 文件中的<taglib>元素、mytaglib.tld 文件中的<tag>元素，以及 HelloTag.class 文件之间的对应关系。

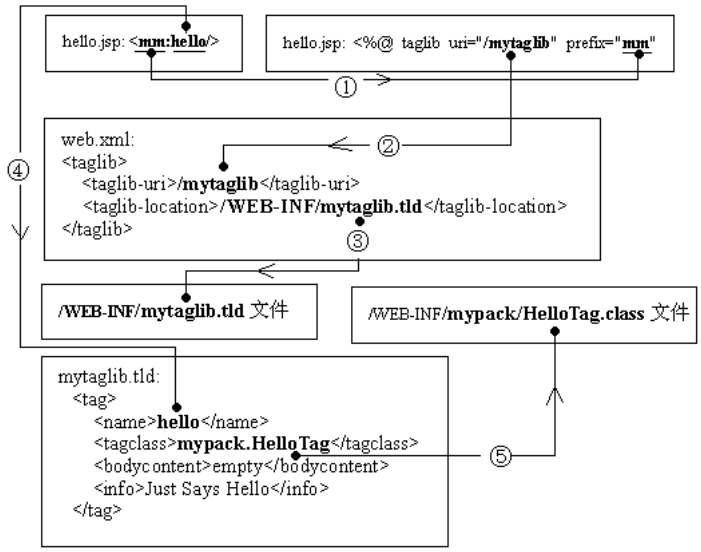

图 3-8　hello 标签与相关配置代码及处理类的对应关系

3.5　用批处理文件或 ANT 编译范例

为了便于读者编译源程序，在 sourcecode/chapter03 目录下提供了编译本章 Java 程序的批处理文件 compile.bat，它的内容如下：

```
set catalina_home=C:\tomcat
set path=%path%;C:\jdk\bin

set currpath=.\
if "%OS%" == "Windows_NT" set currpath=%~dp0%

set src=%currpath%helloapp\src
set dest=%currpath%helloapp\WEB-INF\classes
set classpath=%catalina_home%\lib\servlet-api.jar;
           %catalina_home%\lib\jsp-api.jar

javac   -sourcepath %src%
       -d %dest% %src%\mypack\DispatcherServlet.java
javac   -sourcepath %src%
       -d %dest% %src%\mypack\HelloTag.java
```

运行这个批处理文件时，只要先重新设置以上 Tomcat 目录和 JDK 的根目录即可。在以上 javac 命令中，-sourcepath 选项设定 Java 源文件的路径，-d 选项设定编译生成的类的存放路径。 javac 命令的-classpath 选项设定 classpath 路径，如果此项没有设定，将参照操作系统中 classpath 环境变量的设置。在 compile.bat 文件中，“set classpath”命令用来设置 classpath 环境变量。

除了用上述批处理文件编译范例，还可以使用 ANT 工具来编译范例，本书第 30 章（用 ANT 工具管理 Web 应用）介绍了 ANT 工具的用法。在本书提供的每个范例 Web 应用的根目录下，都有一个 build.xml 文件，它是 ANT 的工程管理文件。例程 3-8 是本章 helloapp 应用下的 build.xml 文件的内容。

例程 3-8　build.xml

```
<project name="helloapp" default="compile" basedir=".">
  <property name="tomcat.home" value="C:/tomcat" />
  <property name="app.home"   value="." />

  <property name="src.home"    value="${app.home}/src"/>
  <property name="classes.home"
          value="${app.home}/WEB-INF/classes"/>


  <path id="compile.classpath">  <!-- 设置 classpath -->

    <pathelement location="${classes.home}"/>
    <fileset dir="${tomcat.home}/lib">
      <include name="*.jar"/>
    </fileset>
  </path>

  <target name="compile" > <!--编译任务 -->
    <javac srcdir="${src.home}" destdir="${classes.home}"
          debug="yes" includeAntRuntime="false">
       <classpath refid="compile.classpath"/>
    </javac>
  </target>
</project>
```

用 ANT 工具来编译范例的步骤如下。

（1）参照本书第 30 章的 30.1 节（安装配置 ANT），安装和配置 ANT。

（2）打开 helloapp 应用下的 build.xml 文件，确认 tomcat.home 属性的值为本地 Tomcat 的根目录：

```
<property name="tomcat.home" value="C:/tomcat" />
```

（3）在 DOS 下，转到 helloapp 目录下，运行命令“ant”，该命令会执行 build.xml 文件中的“compile”任务，编译 src 子目录下的所有 Java 源文件，编译生成的.class 文件位于 WEB-INF/classes 子目录下。

3.6 小结

本章通过 helloapp 应用例子，介绍了 JavaWeb 应用的目录结构和开发过程，还介绍了在 Tomcat 上发布 JavaWeb 应用的步骤。

JavaWeb 应用有着固定的目录结构，各种类型的 Web 组件都有专门的存放目录。此外 JavaWeb 应用还有一个 web.xml 文件，当 JavaWeb 应用被发布到一个 Servlet 容器中时，Servlet 容器需要从 web.xml 文件中来获取有关 Web 组件的配置信息。本章介绍了在 web.xml 文件中为 Servlet 映射 URL 的配置方法。

Tomcat 允许为 Web 应用配置<Context>元素，从而进一步控制 Tomcat 发布和运行 Web 应用的行为。<Context>元素有多种配置方式。如果为单个 Web 应用配置<Context>元素，可以把它放在 Web 应用的 META-INF/context.xml 文件中，或者把它放在 Tomcat 的 <CATALINTA_HOME>/conf/[enginename]/[hostname]/[contextpath].xml 文件中。

要在 Tomcat 中发布 Web 应用，最简单的做法就是把 Web 应用的所有文件拷贝到 <CATALINTA_HOME>/webapps 目录下。默认情况下，Tomcat 启动时会自动加载 webapps 目录下的 Web 应用，并把它发布到名为 Catalina 的 Engine 组件的 localhost 虚拟主机中。此外，通过在<CATALINTA_HOME>/conf/server.xml 文件中配置<Host>元素，也可以把 Web 应用发布到其他虚拟主机中。

本章还介绍了自定义 JSP 标签的创建、发布和使用方法。每个自定义标签都对应一个标签处理类，当 Servlet 容器遇到 JSP 文件中的自定义标签时，就会调用标签处理类的相关方法。自定义标签放在标签库中，基于 XML 格式的 TLD 文件是标签库的定义文件，它包含了对标签库的定义，以及对库中所有标签的定义。

如果一个 JavaWeb 应用使用了某个标签库中的标签，必须先在 web.xml 文件中配置这个标签库，指定它的 TLD 文件的存放位置以及它的 URI。接下来在 JSP 文件中先为这个标签库指定一个前缀，然后就可以通过<前缀名：标签名/>的形式来使用这个标签。

3.7 思考题

1．关于本章提到的 web.xml、context.xml 和 server.xml 文件，以下哪些说法正确？（多选）

（a）web.xml 文件的根元素为<Context>元素。

（b）Servlet 规范规定 JavaWeb 应用的配置文件为 web.xml 文件。

（c）Servlet 规范规定 Servlet 容器的配置文件为 server.xml 文件。

（d）web.xml 文件和 server.xml 文件都是 XML 格式的配置文件。

（e）Servlet 规范规定 JavaWeb 应用的的配置文件为 context.xml 文件。

2．关于 JavaWeb 应用的目录结构，以下哪些说法正确？（多选）

（a）JavaWeb 应用的目录结构完全由开发人员自行决定。

（b）JavaWeb 应用中的 JSP 文件只能存放在 Web 应用的根目录下。

（c）web.xml 文件存放在 WEB-INF 目录下。

（d）JavaWeb 应用中的.class 文件存放在 WEB-INF/classes 目录或其子目录下。

3．假设在 helloapp 应用中有一个 hello.jsp，它的文件路径如下:

```
<CATALINA_HOME>/webapps/helloapp/hello/hello.jsp
```

在 web.xml 文件中没有对 hello.jsp 作任何配置。浏览器端访问 hello.jsp 的 URL 是什么？（单选）

（a）http://localhost:8080/hello.jsp

（b）http://localhost:8080/helloapp/hello.jsp

（c）http://localhost:8080/helloapp/hello/hello.jsp

（d）http://localhost:8080/hello

4．假设在 helloapp 应用中有一个 HelloServlet 类，它位于 net.javathinker 包中，那么这个类的.class 文件的存放路径应该是什么？（单选）

（a）helloapp/HelloServlet.class

（b）helloapp/WEB-INF/HelloServlet.class

（c）helloapp/WEB-INF/classes/HelloServlet.class

（d）helloapp/WEB-INF/classes/net/javathinker/HelloServlet.class

5．假设在 helloapp 应用中有一个 net.javathinker.HelloServlet 类，它在 web.xml 文件中的配置如下:

```
<servlet>
  <servlet-name> HelloServlet </servlet-name>
  <servlet-class>net.javathinker.HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name> HelloServlet </servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

那么在浏览器端访问 HelloServlet 的 URL 是什么？（单选）

（a）http://localhost:8080/HelloServlet

（b）http://localhost:8080/helloapp/HelloServlet

（c）http://localhost:8080/helloapp/net/javathinker/hello

（d）http://localhost:8080/helloapp/hello

（e）http://localhost:8080/helloapp/net/javathinker//HelloServlet.class

6．以下配置代码定义了一个名为 hello 的标签：

```
<tag>
  <name>hello</name>
  <tagclass>mypack.HelloTag</tagclass>
```

```
    <bodycontent>empty</bodycontent>
    <info>Just Says Hello</info>
  </tag>
```

以上代码可能位于哪个配置文件中？（单选）

（a）一个 TLD 文件中。

（b）Tomcat 的 conf/server.xml 文件中。

（c）JavaWeb 应用的 WEB-INF/web.xml 文件中。

（d）JavaWeb 应用的 META-INF/context.xml 文件中。

7. 实验题：修改本章范例 helloapp 应用，在 helloapp/META-INF 目录下创建 context.xml 文件，使得访问 helloapp 应用的 URL 入口为“/hello”。例如：通过“http://localhost:8080/hello/login.htm”就可以访问 login.htm 页面。

参考答案

1．b,d　2．c,d　3．c　4．d　5．d　6．a

7．提示：context.xml 文件中需要加入如下<Context>元素：

```
<Context path="/hello" docBase="helloapp" reloadable="true" />
```