

## 第3章 JSP开发平台的建立：Tomcat

自从JSP发布以后，推出了各式各样的JSP引擎。作为世界上用得最多的Web服务器软件——Apache的Apache Group也在进行JSP的实用研究。最初的软件产品是在Apache的Java Servlet引擎即ApacheJServ的基础上实现的GNUJSP，一直到GNUJSP1.0，基本上实现了JSP 1.0标准。另外还出现了一个被称为GSP的产品，是作为GNU体系的一个服务器端的Script语言实现的。

GNUJSP基本上是一个ApacheJServ的附属，它主要是利用Servlet将JSP源文件翻译为一个Servlet的Java语言源文件，然后经Java编译器编译后作为Servlet执行，这样做的好处前面已经说明了。

在完成GNUJSP1.0的开发以后，开发组的成员开始考虑在SUN的JSWDK基础上开发一个可以直接提供Web服务的JSP服务器，当然同时也支持Servlet。这样，Jakarta-Tomcat就诞生了。

作为一个开放源代码的软件，Jakarta-Tomcat有着自己独特的优势：

首先，它容易得到。事实上，任何人都可以从互联网上自由地下载这个软件。无论从<http://jakarta.apache.org>还是从其他网站。

其次，对于开发人员，特别是Java开发人员，Tomcat提供了全部的源代码，包括Servlet引擎、JSP引擎、HTTP服务器……，无论是对哪一方面感兴趣的程序员，都可以从这些由世界顶尖的程序员书写的代码中获得收益。

最后，由于源代码的开放及世界上许多程序员的卓有成效的工作，Tomcat已经可以和大部分的主流服务器一起工作，而且是以相当高的效率一起工作。如：以模块的形式被载入Apache，以ISAPI的形式被载入IIS或PWS，以NSAPI的形式被载入Netscape Enterprise Server……。

接下来，读者可以看到：

- 如何安装Tomcat，让它发挥作用。
- 如何让Tomcat和Apache、IIS等一起工作。
- 如何配置Tomcat，让它符合自己的要求。

下面，先来建立一个试验用的JSP页面，读者可以先将以下的代码存为HelloWorld.jsp。

```
<HTML>
  <HEAD>
    <TITLE>JSP测试页面---HelloWorld!</TITLE>
  </HEAD>
  <BODY>
    <%
      out.println("<h1>Hello World!<br>世界，你好！</h1>");
    %>
  </BODY>
</HTML>
```

### 3.1 Tomcat的安装和直接使用

在Apache的jakarta项目的主页上，可以看到有Tomcat的超连接，在这里可以找到各种版本

的下载区域，包括当前的发布（Release）版本、开发中的各种版本，其中又分为 Win32版本和 Linux版本，其实对于完全由 Java写成的 Tomcat，Win32版本和 Linux版本没有多大区别，比如 Linux版本，在 Solaris下也没有问题。这里，主要以 Win32版本作为示例。

注意：在安装使用 Tomcat之前，先安装 JDK，最好是 Sun的 JDK1.2.2或 JDK1.3。

首先，下载 jakarta-tomcat.zip包，解压缩到一个目录下，如：“c:\tomcat”。这时，会得到如下的目录结构：

tomcat

---jakarta-tomcat

---bin	Tomcat执行脚本目录
---conf	Tomcat配置文件
---doc	Tomcat文档
---lib	Tomcat运行需要的库文件（JARS）
---logs	Tomcat执行时的LOG文件
---src	Tomcat的源代码
---webapps	Tomcat的主要Web发布目录
---work	Tomcat的工作目录，

Tomcat将翻译JSP文件到的Java文件和class文件放在这里

在 Bin目录下，有一个名为 startup.bat的脚本文件，执行这个脚本文件，就可以启动 Tomcat服务器，不过，在启动服务器之前，还需要进行一些设置。

首先，设置环境变量。

Win9x在 autoexec.bat里用 set 语句来设定环境变量，如：set TOMCAT\_HOME = c:\tomcat。

在 winnt/win2000里可以选择“我的电脑”，右键点出菜单，选择属性，弹出对话框“系统特性”，选择“高级”选项页，然后点按钮“环境变量”，可以编辑系统的环境变量。

- TOMCAT\_HOME值：c:\tomcat (用TOMCAT\_HOME指示tomcat根目录)。
- JAVA\_HOME值：c:\java\jdk(用JAVA\_HOME指示jdk1.3安装目录)。
- CLASSPATH值：c:\java\jdk\lib\tools.jar。

实际上，对于 CLASSPATH也可以直接打开 tomcat.bat文件，在中间可以找到好几行 set CLASSPATH.....，将自己希望加入的库文件加入到其中即可。

另外，对于 JDK1.3，在中文系统上安装之后，系统注册表会有问题，请用 regedit打开注册表查javasoft，位置为hkey\_local\_machine->software->javasoft->，找到“Java 运行时环境”把它导出到文件 temp.reg....，然后用 notepad编辑它，把“Java 运行时环境”替换成“Java Runtime Environment”，然后导入。

同样，最好也把 javasoft注册表项中的“Java 插件”另外复制一份为“Java Plug-in”。

接下来就可以执行 TOMCAT\_HOME\bin\startup.bat，测试一下 Tomcat是否运行正常。

运行 Web浏览器，如 Netscape Navigator或 Internet Explorer。在浏览器的地址栏中键入：  
http://127.0.0.1:8080。如果看到 Tomcat的信息，那么就说明 Tomcat已经安装成功了。然后测试 Tomcat的 JSP引擎是否正常工作，即将前面建立的 HelloWorld.jsp文件拷贝到 TOMCAT\_HOME\w

ebapps\examples\jsp目录下，然后在浏览器的地址栏中键入：<http://127.0.0.1:8080/examples/jsp/HelloWorld.jsp>，这时候应该可以看到如图 3-1 所示的画面：

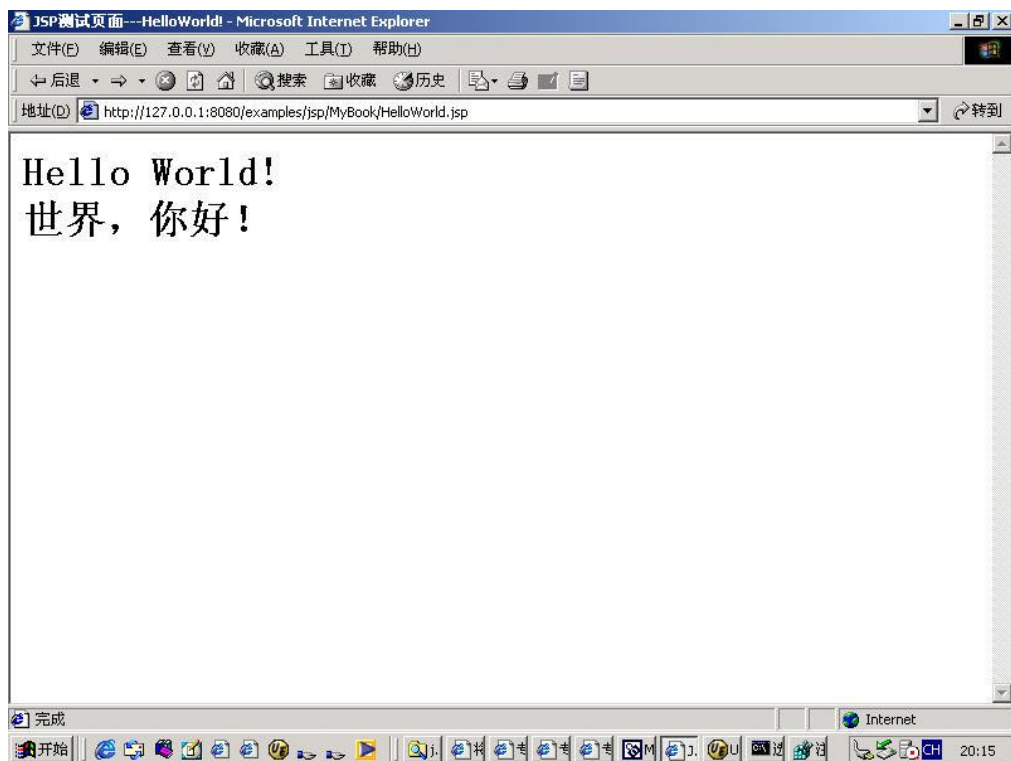


图 3-1

在启动Tomcat的过程中，可能会遇到一些问题，这里就常见问题进行一些说明。

1) 启动Tomcat失败。出现这种情况时，可能有两种现象：

第一种：执行startup.bat以后没有Java窗口出现。

第二种：有Java窗口出现，但是接着自行退出。

对于第一种情况，很可能是 TOMCAT\_HOME环境变量设置有问题，打开 startup.bat文件，观察脚本的写法，一般可以发现问题。

对于第二种情况，可能是当前系统中已经有一个服务器占用了 8080端口，这时需要把原先的服务器关闭，或者利用后面讲的 Tomcat的配置方法修改 Tomcat的服务端口。也可能是由于 CLASSPATH设置有误，这时需要检查CLASSPATH是否设置正确。

2) 启动Tomcat成功，可以看到首页，但是不能执行 JSP脚本。

这种情况一般是由于 CLASSPATH设置有误或 JAVA\_HOME设置有误，经过试验发现，当使用 Windows自带的 java.exe (c:\winnt\system32或c:\windows\system) 时可能会出现这种问题。

## 3.2 Tomcat和Apache的配合

作为 Apache的一个子项目 jakarta-tomcat当然要对 Apache提供强有力的支持，在下载

Tomcat压缩包解开后，在TOMCAGT\_HOME\conf目录下有一个tomcat-Apache.conf文件，这个文件并不是Tomcat自己的配置文件，而是提供给Apache用来使Tomcat能够和Apache一起工作的。实际上，这个文件是在Tomcat的运行过程中自动生成的。

但是，光有这个文件还不能使Apache和Tomcat一起工作，还需要一个Apache的动态载入库文件ApacheModuleJserv.dll，这个文件也可以在网站<http://jakarta.apache.org>得到，需要说明的是，对于Linux版本的tomcat，需要的是mod\_jserv.so文件，ws50为后缀的文件是Linux下的动态连链库文件。

首先，要得到Apache HTTP服务器。Apache是一个免费而且提供源代码的HTTP服务器，由于Apache强大的性能和用户可以利用源代码构造自己的HTTP服务器的特性，Apache及其衍生出来的产品已经成为世界上应用最多的HTTP服务器，甚至连著名的IBM公司为Websphere应用服务器提供的IBM HTTP Server也是由Apache改造而来的。

在<http://jakarta.apache.org>可以得到Apache服务器的最新版本，本书写作的时候，最新的发行版已经到了Apache1.3.12，而Apache2.0a6也已经提供用于测试了。

Windows下的Apache版本是一个安装文件，可以轻松地安装在计算机上。而如果在Linux下使用Apache，那么最好使用源代码包自己进行编译，需要注意的是，编译时需使用选项 enable-module=so。

双击Apache\_1\_3\_12\_Win32.exe文件进行安装，缺省安装目录为C:\Program Files\Apache Group\Apache，可以修改为自己喜欢的目录。

如果需要修改Apache服务器工作的端口号以及HTML发布目录或者其他Apache的参数，那么可以修改Apache安装目录\conf\httpd.conf，一般可以修改HTML发布目录为自己喜欢的目录。至于端口号，当计算机上还运行有其他Web服务器时可以修改之，一般Windows9x的机器上如果装有PWS就需要修改，而WindowsNT和Windows 2000的机器上如果装有IIS，也需要修改。对于没有连接到网络的机器，有时需要设置一下ServerName。Apache服务器的具体配置请见相关书籍，这里就不讲了。

另外最好将Apache作为一个服务安装在运行WindowsNT和Windows 2000的电脑上。这只需要执行开始 程序 Apache Web Server Install Apache as a service即可。

打开浏览器，在地址栏中键入<http://127.0.0.1:Apache运行的端口号>，如果能够见到Apache的欢迎页面，或者是一大堆文件让你选择，就可以认为Apache服务器已经开始工作了。

Apache HTTP 服务器配置成功以后，就可以着手让Tomcat和Apache一起工作。首先，将得到的ApacheModuleJserv.dll文件拷贝到Apache安装目录下的modules子目录下，Linux的用户将mod\_jserv.so文件拷贝到Apache安装目录的libexec目录下，然后将Apache安装目录下的httpd.conf文件用文本编辑器打开，在最后面加入下面的指令：

INCLUDE Tomcat\_Home\conf\tomcat.conf                   ——对于Windows用户。

或INCLUDE Tomcat\_Home/conf/tomcat.conf               ——对于Linu用户。

上面的Tomcat\_Home指的是Tomcat的安装目录。

最后，在httpd.conf文件中加上一行:LoadModule jserv\_module modules/ApacheModuleJserv.dll。

对于Linux下的用户，一般不需要手动加上LoadModule jserv\_module libexec/mod\_jserv.so这一行，tomcat-Apache.conf文件已经缺省加上了，如果没加，自行加上即可。

一切就绪以后，重新启动 Apache服务器和 Tomcat，在浏览器的地址栏中键入：<http://127.0.0.1:Apache运行的端口号/examples/jsp/>，如果能够看到 Tomcat的JSP示例列表，就说明 Tomcat已经和Apache一起工作了。

### 3.3 Tomcat和IIS的配合

Windows平台下最常用的Web服务器无疑是IIS（包括PWS），对于IIS，Tomcat也提供了配合工作的方法，使用这种方法，可以为本来不具有 Java Servlet和JSP功能的IIS增加处理JSP和Java Servlet的功能。

为了使Tomcat和IIS一起工作，首先要得到 isapi\_redirect.dll，这是一个IIS的插件（Plug-in），可以从<http://jakarta.apache.org/>直接下载编译好的版本，也可以自己使用 Visual C++ 编译得到。得到以后，放到一个自己喜欢的目录，例如 c:\tomcat\Jakarta-tomcat\bin\iis\i386\ 目录下。

另外，在使 IIS和Tomcat配合的过程中，还需要用到另外两个 Tomcat的配置文件，一个是 workers.properties，这个文件定义了Tomcat的工作进程使用的主机和端口。在Tomcat的conf目录中有一个示范性的workers.properties文件。另一个是uriworkermap.properties，这个文件是映射URL目录和Tomcat工作进程的。同样，在Tomcat的conf目录中有一个示范性的uriworkermap.properties文件。

首先，配置 isapi\_redirect.dll。

1) 在系统注册表中建立一个新的键值：HKEY\_LOCAL\_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi Redirector\1.0。

2) 添加一个名为extension\_uri的字符串值为/jakarta/isapi\_redirect.dll。

3) 添加一个名为log\_file的字符串值为c:\tomcat\Jakarta-tomcat\logs\isapi.log。

4) 添加一个名为log\_level的字符串值为debug、inform、error、emerg中的一个。

5) 添加一个名为worker\_file的字符串值为

6) c:\tomcat\jakarta-tomcat\conf\workers.properties。

7) 添加一个名为worker\_mount\_file的字符串值为

8) c:\tomcat\jakarta-tomcat\conf\ uriworkermap.properties。

然后，打开IIS的管理控制台，在需要使用Tomcat提供附加的JSP和Java Servlet服务的Web站点中添加一个虚拟目录。注意，一定要使用“jakarta”作为虚拟目录的名称，这个虚拟目录的实际物理位置应当是包含 isapi\_redirect.dll文件的目录，这里假设为c:\tomcat\Jakarta-tomcat\bin\iis\i386。在设定虚拟目录时注意要设此虚拟目录为可执行。如果是在PWS中，一样处理。

接着，在IIS的控制台中为此Web站点添加一个ISAPI过滤器（在此Web站点上点击鼠标右键，选择属性）。名称随意，但过滤器要设定为 isapi\_redirect.dll这个文件。如果使用的是PWS就比较麻烦了。需要使用注册表编辑器，在键 HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\W3SVC\Parameters中，有一个名为Filter DLLs的键值，在这个键值中添加 isapi\_redirect.dll，注意要包含完整的路径。

最后，重新启动IIS或PWS，最好是能够重新启动计算机。

启动Tomcat以后，打开浏览器，在地址栏中键入 `http://127.0.0.1/examples/`，如果能够看到 `jsp`和`servlet`两个目录，就说明利用 `isapi_redirect.dll`所作的重定向已经成功，可以执行一下 `jsp`目录下的例子做试验。

### 3.4 Tomcat的配置和常见问题

Tomcat为用户提供了一系列的配置文​​件来帮助用户配置自己的 Tomcat，和Apache HTTP不同，Tomcat的配置文​​件主要是基于XML的；如`server.xml`、`web.xml`.....，只有`workers.properties`和`uriworkermap.properties`等少数几个文件是传统的配置文​​件。本节将详细讨论 Tomcat的主要配置文​​件以及如何利用这些配置文​​件解决常见问题。

#### 3.4.1 Tomcat的主配置文​​件：server.xml

观察`server.xml`，可以发现其中有如表0一些元素。

表3-1

元 素	描 述
Server	Server元素是 <code>server.xml</code> 文件的最高级别的元素，Server元素描述一个Tomcat服务器，一般来说用户不用关心这个元素。一个 Server元素一般包括Logger和ContextManager两个元素
Logger	Logger元素定义了一个日志对象，一个日志对象包含有如下属性： 1) name。表示这个日志对象的名称。 2) path。表示这个日志对象包含的日志内容要输出到哪一个日志文件。 3) verbosityLevel。表示这个日志文件记录的日志的级别。 一般来说，Logger对象是对Java Servlet、JSP和Tomcat运行期事件的记录
ContextManager	ContextManager定义了一组ContextInterceptors（ContextManage的事件监听器），RequestInterceptors（ContextManage的事件监听器），Contexts（Web应用程序的上下文目录）和它们的Connectors（连接器）的结构和配置。 ContextManager包含如下一些属性： 1) debug。记录日志记录调试信息的等级。 2) home。webapps/、conf/、logs/和所有Context的根目录信息。这个属性的作用是从一个不同于TOMCAT_HOME的目录启动Tomcat。 3) workDir。Tomcat工作目录。
ContextInterceptor 和RequestInterceptor	两者都是监听ContextManager的特定事件的拦截器。ContextInterceptor监听Tomcat的启动和结束事件信息。而RequestInterceptor监听用户对服务器发出的请求信息。一般用户无需关心这些拦截器，对于开发人员，需要了解这就是全局性的操作得以实现的方法
Connector	Connector（连接器）对象描述了一个到用户的连接，不管是直接由Tomcat到用户的浏览器还是通过一个Web服务器。Tomcat的工作进程和由不同的用户建立的连接传来的读/写信息和请求/答复信息都是由连接器对象管理的。对连接器对象的配置中应当包含管理类、TCP/IP端口等内容



(续)

元 素	描 述
Context	<p>每一个Context都描述了一个Tomcat的Web应用程序的目录。这个对象包含以下属性：</p> <ul style="list-style-type: none"><li>1) docBase。这是Context的目录。可以是绝对目录也可以是基于ContextManage的根目录的相对目录。</li><li>2) path。这是Context在Web服务时的虚拟目录位置和目录名。</li><li>3) debug。日志记录的调试信息记录等级。</li><li>4) reloadable。这是为了方便Servlet的开发人员而设置的，当这个属性开关打开的时候，Tomcat将检查Servlet是否被更新而决定是否自动重新载入它</li></ul>

1. 加入自己的日志文件

添加Logger对象就可以加入自己的日志文件，添加工作相当简单，只需要将作为示例的Logger对象复制一份，然后修改一下前面介绍的几个属性就可以了。在设定了Logger以后，就可以在自己的Servlet中使用ServletContext.log()方法来建立自己的日志文件。

2. 设定新的JSP目录

设立新的JSP工作目录是比较简单的，只需要添加一个Context对象就可以了。如，要在c:\jsp目录下开发JSP项目，并且让用户可以使用/mybook/虚拟目录访问，则：

```
<Context path="/mybook" docBase="c:\jsp" debug="0" reloadable="true" >
</Context>
```

一般来说，这样就可以直接执行JSP文件了，如果进一步想要在这下面建立Web应用程序，那么还需要进一步的配置，具体方法在后面论述。

3.4.2 Windows下代码保护的问题

在Windows下使用Tomcat时有一个问题需要注意，可以做一个试验，启动Tomcat后，在浏览器的地址栏中键入：http://127.0.0.1:8080/examples/jsp/ HelloWorld.JSP（注意后缀要大写）。就会发现奇怪的现象，浏览器的窗口中什么都没有，查看HTML源文件就会发现，这个JSP文件的源代码被Tomcat完全输出到了浏览器！如果是这样，岂不是服务器端的任何源代码都会被暴露在互联网上。

实际上，解决方法很简单，把各种后缀的组合全部写到Tomcat\_Home\conf\web.xml里就可以了，这样tomcat会将不同后缀名的jsp分开对待，就不会泄露代码了。

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

```
<servlet-name>
    Jsp
</servlet-name>
<url-pattern>
    *.Jsp
</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>
        JSp
    </servlet-name>
    <url-pattern>
        *.JSp
    </url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>
        Jsp
    </servlet-name>
    <url-pattern>
        *.JSP
    </url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>
        JSP
    </servlet-name>
    <url-pattern>
        *.JSP
    </url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>
        jSp
    </servlet-name>
    <url-pattern>
        *.jSp
    </url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>
        jSP
    </servlet-name>
    <url-pattern>
        *.jSP
    </url-pattern>
</servlet-mapping>
<servlet-mapping>
```



```

<servlet-name>
    jsP
</servlet-name>
<url-pattern>
    *.jsp
</url-pattern>
</servlet-mapping>

```

### 3.4.3 Apache、IIS和Tomcat协作时工作目录的添加

#### 1. Apache

由于Jakarta-Tomcat项目是Apache的一个子项目，所以向Tomcat-Apache协作的Web服务器添加工作目录时只需要修改Tomcat-Apache.conf文件就可以了。也许读者会觉得奇怪，Tomcat-Apache.conf文件不是在Tomcat启动时自动生成的吗？的确如此，但是Tomcat自动生成的Tomcat-Apache.conf文件仅仅是Tomcat提供的一个缺省配置文件而已，如果需要，可以修改，然后存放在另外的目录中，或是更名，再在httpd.conf文件中将这个新的文件包含进来就可以了。

为什么Tomcat不自动修改Tomcat-Apache.conf文件以适应工作目录添加的需要呢？Tomcat的确修改了Tomcat-Apache.conf文件，但是修改的结果显然是不正确的。如，前面在server.xml中添加了工作目录/mybook-->c:\jsp后，Tomcat修改Tomcat-Apache.conf文件，添加了这么几行：

```

Alias /mybook C:\tomcat\jakarta-tomcat\webapps\mybook
<Directory "C:\tomcat\jakarta-tomcat\webapps\mybook">
    Options Indexes FollowSymLinks
</Directory>
ApJServMount /mybook/servlet /mybook
<Location /mybook/Web-INF/ >
    AllowOverride Non
    deny from all
</Location>

```

这显然是有问题的，尽管虚拟目录是/mybook，但是实际的目录并不是C:\tomcat\jakarta-tomcat\webapps\mybook。查看Tomcat的源代码C:\tomcat\jakarta-tomcat\src\org\Apache\tomcat\task\ApacheConfig.java文件可以发现，Tomcat在生成Tomcat-Apache.conf文件的时候，简单地在虚拟目录前面加上原先Tomcat的缺省webapp目录作为新的工作目录：

```

pw.println("Alias " + path + " " +
    FileUtil.patch(tomcatHome + "/webapps" + path));
pw.println("<Directory \" " +
    FileUtil.patch(tomcatHome + "/webapps" + path) +
    "\">");
pw.println("    Options Indexes FollowSymLinks");
pw.println("</Directory>");

```

那么如何解决这个问题呢？修改Tomcat的源代码也可以，不过，对于一般的用户，如前所述直接修改Tomcat-Apache.conf文件更现实一些。修改的方法举例如下：

```
Alias /mybook C:\jsp
```

```
<Directory "C:\jsp">
    Options Indexes FollowSymLinks
</Directory>
ApJServMount /mybook/servlet /mybook
<Location /mybook/Web-INF/ >
    AllowOverride None
    deny from all
</Location>
```

也就是简单地将原先错误的实际目录 C:\tomcat\jakarta-tomcat\webapps\mybook 修改为 c:\jsp 就可以了。

修改完文件后，一定要存为另一个文件并修改 httpd.conf 文件将这个文件包含进来，否则，重新启动 Tomcat 后，这个正确的 Tomcat-Apache.conf 文件会被 Tomcat 重新生成的 Tomcat-Apache.conf 文件覆盖掉。最后，重启 Apache 和 Tomcat 就可以了。

## 2. IIS

与 Apache 和 Tomcat 几乎无缝的配合不一样，IIS 和 Tomcat 的配合多少有些复杂，需要向 ISAPI Redirect 添加新的内容。不过还好，Tomcat 的 uniworkermap.conf 文件将这个过程简单化了。

需要做的事情共分两步：

- 1) 向 Tomcat 中添加一个工作目录。前面已经讲述了如何实现，这里依然使用 /mybook-->c:\jsp 这个例子。
- 2) 向 ISAPI Redirect 添加工作目录。使用文本编辑器打开文件 uniworkermap.conf，添加一行：

```
/mybook/*=ajp12
```

然后重新启动 IIS 和 Tomcat 就可以了。

### 3.4.4 设定 Tomcat 作为 Windows 的服务而启动

手工启动 Tomcat 显然不是一个合适的使用 Tomcat 作为 Web 服务的方法，在 Linux 下可以通过修改启动脚本自动启动 Tomcat，在 Windows 下则可以设定 Tomcat 作为 Windows 的服务而启动。

Tomcat 作为 Windows NT/2000 的一个服务是需要借助工具的

- 1) 下载工具，这里作为例子的是 gservany ---- 将 NT 下的一般应用程序作为服务运行的工具。下载网址为 <http://www.advok.com/gservany.html>。将 zip 文件解压缩，将 gservany.exe 放入 winnt\system32 目录下，(以防以后被误删)。

- 2) 在 NT 的 Command (命令行模式) 下输入：gservany -i tomcat "C:\jakarta-tomcat\bin" "startup.bat" "C:\jakarta-tomcat\bin" "shutdown.bat"。

- 3) 启动 service 管理器，会看到 tomcat service 被装上，加些注释说明这个 service 实际干什么，再修改启动类型为“自动”。然后再启动它。

这样，就成功地将 Tomcat 作为 Service 安装在 NT 下了。

其实，将 NT 下的应用程序作为服务安装在 NT 中的工具还有很多，任何一种都应该可以将

Tomcat加入到NT的服务中。

### 3.4.5 在Tomcat中建立新的Web应用程序

JSP主要是为建立Web网站而开发的技术，这种技术由 Web应用程序的一整套 Web文件（jsp，servlet，html，jpg，gif，class.....）所组成。Tomcat为Web应用程序的建立提供了一系列的帮助，下面分步骤描述。

#### 1. 应用程序的目录和结构

按照Tomcat的规范，从/example例子目录来看，Tomcat的Web应用程序应该由如表 3-2所示目录组成的。

表3-2

*.html, *.jsp, etc.	这里可以有許多目录，由用户的网站结构而定，实现的功能应该是网站的界面，也就是用户主要的可见部分。除了 HTML文件、JSP文件外，还有 js（JavaScript）文件和css（样式表）文件以及其他多媒体文件等等
Web-INF/web.xml	这是一个Web应用程序的描述文件。这个文件是一个 XML文件，描述了Servlet和这个Web应用程序的其他组件信息，此外还包括一些初始化信息和安全约束等等
Web-INF/classes/	这个目录及其下的子目录应该包括这个 Web应用程序的所有 Servlet文件，以及没有被压缩打入JAR包的其他class文件和相关资源。注意，在这个目录下的 Java类应该按照其所属的包组织目录
Web-INF/lib/	这个目录下包含了所有压缩到 JAR文件中的类文件和相关文件。比如：第三方提供的Java库文件、JDBC驱动程序等等 .....

#### 2. web.xml文件

web.xml文件包含了描述整个 Web应用程序的信息。下面以一个 web.xml文件为例，讲解里面的各个对象。

```
web.xml :
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
<!--
Web应用程序的主要描述
-->
<display-name>My Web Application</display-name>
<description>
    在这里加入Web应用程序的描述信息
</description>
<!--
下面定义了Web应用程序的初始化参数，
在JSP或Servlet文件中使用下面的语句
来得到初始化参数
String value =
    getServletContext().getInitParameter("name");
这里可以定义任意多的初始化参数
-->
```

```

<context-param>
  <param-name>webmaster</param-name>
  <param-value>myaddress@mycompany.com</param-value>
  <description>
    这里包含了初始化参数的描述
  </description>
</context-param>
<!--

```

下面的定义描述了组成这个Web应用程序的Servlet，还包含初始化参数。在Tomcat中，也可以将放在Web-INF/classes中的Servlet直接以servlet/Servlet名访问，但是一般来说，不推荐这样使用。而且这样的使用方法还会导致Servlet的相关资源组织的复杂性。所以一般来说推荐将所有的Servlet在这里定义出来。初始化参数可以在Servlet中一这种语句的到：

```

    String value =
        getServletConfig().getInitParameter("name");
-->
<servlet>
  <servlet-name>controller</servlet-name>
  <description>
    这里加入这个Servlet的描述
  </description>
  <servlet-class>com.mycompany.mypackage.ControllerServlet</servlet-class>
  <init-param>
    <param-name>listOrders</paramName>
    <param-value>com.mycompany.myactions.ListOrdersAction</param-value>
  </init-param>
  <init-param>
    <param-name>saveCustomer</paramName>
    <param-value>com.mycompany.myactions.SaveCustomerAction</param-value>
  </init-param>
<!--

```

服务器启动后这个Servlet加载的时间

```

-->
  <load-on-startup>5</load-on-startup>
</servlet>
<servlet>
  <servlet-name>graph</servlet-name>
  <description>
    这个Servlet的描述
  </description>
</servlet>
<!--
Servlet映射对应了一个特殊的URI请求
到一个特殊的Servlet的关系
-->

```

```

<servlet-mapping>
  <servlet-name>controller</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
<servlet-mapping>
  <servlet-name>graph</servlet-name>
  <url-pattern>/graph</url-pattern>
</servlet-mapping>
<!--
设定缺省的Session过期时间
-->
<session-config>
  <session-timeout>30</session-timeout>      <!-- 30 minutes -->
</session-config>
</web-app>

```

### 3. 将应用程序打包为WAR文件

WAR文件是包装Web应用程序的一种方法，使用WAR文件，既方便了管理各种相关文件，又减小了整个应用程序的体积。

下面先来看一看将Web应用程序打包为WAR文件的语法：

```

packager -webArchive[-classpath servletorjspbean/classes [ -
  classFiles package/MyClass1.class: package/MyClass2.class ] ]
  <content-root> [-contentFiles login.jsp:index.html:images/me.gif]
  web.xml myWebApp.war

```

下面是一个简单的应用示例，将 myWebPage.xml 的配置和 myWebPageDir/ 下的文件打入包 myWebPage.war 中：

```
packager -webArchive myWebPageDir myWebPage.xml myWebPage.war
```

使用 -contentFiles 标志可以添加个别的目录文件

```
packager -webArchive myWebPageDir -contentFiles Hello.jsp
  myWebPage.xml myWebPage.war
```

```
packager -webArchive myWebPageDir -contentFiles Hello.jsp:Hello.html
  myWebPage.xml myWebPage.war
```

假定Servlet文件在 classes/package/Servlet1.class，指定Servlet和JSP文件：

```
packager -webArchive -classpath classes myWebPageDir -contentFiles
  Hello.jsp myWebPage.xml myWebPage.war.
```

下面示例如何仅仅包含 package/Servlet1.class 和 packageB/Servlet.class 两个文件到WAR文件中：

```
packager -webArchive -classpath classes -classFiles package/
  Servlet1.class:packageB/Servlet.class myWebPageDir -contentFiles
  Hello.jsp myWebPage.xml myWebPage.war
```

最后，需要说明的是，每个.war文件前面的.xml文件就是前面讲过的web.xml文件。生成的WAR文件可以直接放在包含Web应用程序的目录下使用。