# 第 26 章　Tomcat 与其他 HTTP 服务器集成

Tomcat 的最主要的功能是提供 Servlet/JSP 容器，尽管它也可以作为独立的 Java Web 服务器，但在对静态资源（如 HTML 文件或图像文件）的处理速度，以及提供的 Web 服务器管理功能方面 Tomcat 都不如其他专业的 HTTP 服务器，如 IIS 和 Apache 服务器。

因此在实际应用中，常常把 Tomcat 与其他 HTTP 服务器集成。对于不支持 Servlet/JSP 的 HTTP 服务器，可以通过 Tomcat 服务器来运行 Servlet/JSP 组件。

当 Tomcat 与其他 HTTP 服务器集成时，Tomcat 服务器的工作模式通常为进程外的 Servlet 容器，Tomcat 服务器与其他 HTTP 服务器之间通过专门的插件来通信。关于 Tomcat 服务器的工作模式的概念可以参考本书第 2 章的 2.4 节（Tomcat 的工作模式）。

本章首先讨论 Tomcat 与 HTTP 服务器集成的一般原理，然后介绍 Tomcat 与 Apache 服务器及 IIS 集成的详细步骤，最后介绍把由多个 Tomcat 服务器构成的集群系统与 Apache 服务器集成的方法。

## 26.1　Tomcat 与 HTTP 服务器集成的原理

Tomcat 服务器通过 Connector 连接器组件与客户程序建立连接，Connector 组件负

责接收客户的请求，以及把 Tomcat 服务器的响应结果发送给客户。在默认情况下，Tomcat 在 server.xml 中配置了两种连接器：

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector port="8080" protocol="HTTP/1.1"
       connectionTimeout="20000"
       redirectPort="8443" />

<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

第一个连接器是 HTTP 连接器，监听 8080 端口，负责建立 HTTP 连接。在通过浏览器访问 Tomcat 服务器的 Web 应用时，使用的就是这个连接器。

第二个连接器是 AJP 连接器，监听 8009 端口，负责和其他的 HTTP 服务器建立连接。在把 Tomcat 与其他 HTTP 服务器集成时，就需要用到这个连接器。

Web 客户访问 Tomcat 服务器上的 JSP 组件的两种方式如图 26-1 所示。

在图 26-1 中，Web 客户 1 直接访问 Tomcat 服务器上的 JSP 组件，他访问的 URL 为 http://localhost:8080/index.jsp。Web 客户 2 则通过 HTTP 服务器访问 Tomcat 服务器上的 JSP 组件。假定 HTTP 服务器使用的 HTTP 端口为默认的 80 端口，那么 Web 客户 2 访问的 URL 为 http://localhost:80/index.jsp 或者 http://localhost/index.jsp。

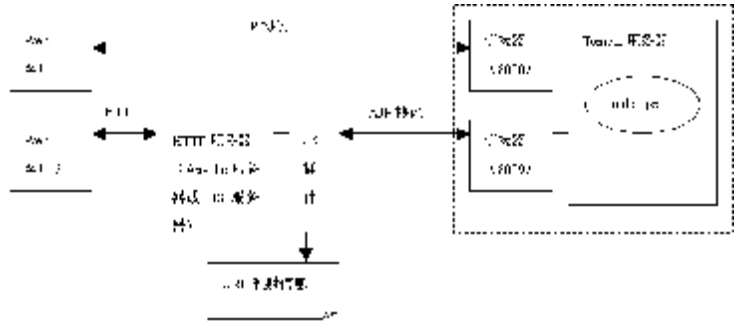

图 26-1　Web 客户访问 Tomcat 服务器上的 JSP 组件的两种方式

下面，介绍 Tomcat 与 HTTP 服务器之间是如何通信的。

### 26.1.1　JK 插件

Tomcat 提供了专门的 JK 插件来负责 Tomcat 和 HTTP 服务器的通信。应该把 JK 插件安置在对方的 HTTP 服务器上，当 HTTP 服务器接收到客户请求时，它会通过 JK 插件来过滤 URL，JK 插件根据预先配置好的 URL 映射信息，决定是否要把客户请求转发给 Tomcat 服务器处理。

假定在预先配置好的 URL 映射信息中，所有“/*.jsp”形式的 URL 都由 Tomcat 服务器来处理，那么在如图 26-1 所示的例子中，JK 插件将把 Web 客户 2 的请求转发给 Tomcat 服务器，Tomcat 服务器于是运行 index.jsp，然后把响应结果传给 HTTP 服务器，HTTP 服务器再把响应结果传给 Web 客户 2。

对于不同的 HTTP 服务器，Tomcat 提供了不同的 JK 插件的实现模块，本章将用

到以下 JK 插件。

- 与 Windows 下的 Apache HTTP 服务器集成：mod_jk.sol。
- 与 Linux 下的 Apache HTTP 服务器集成：mod_jk_linux.so。
- 与 IIS 服务器集成：isapi_redirect.dll。

### 26.1.2　AJP 协议

AJP 是为 Tomcat 与 HTTP 服务器之间通信而定制的协议，能提供较高的通信速度和效率。在配置 Tomcat 与 HTTP 服务器集成时，读者可以不必关心 AJP 协议的细节。关于 AJP 的知识可以参考以下网站：

```
http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html
```

## 26.2　在 Windows 下 Tomcat 与 Apache 服务器集成

Apache HTTP 服务器（下文简称 Apache 服务器或者 Apache）是由 Apache 软件组织提供的开放源代码软件，它是一个非常优秀的专业的 Web 服务器，为网络管理员提供了丰富多彩的 Web 管理功能，包括目录索引、目录别名、内容协商、可配置的 HTTP 错误报告、CGI 程序的 SetUID 功能、子进程资源管理、服务器端图像映射、重写 URL、URL 拼写检查及联机手册等。

Apache 服务器本身没有提供 Servlet/JSP 容器。因此，在实际应用中，把 Tomcat 与 Apache 服务器集成，可以建立具有实用价值的商业化的 Web 平台。

在 Windows XP 下 Tomcat 与 Apache 服务器集成需要准备的软件参见表 26-1。

表 26-1　在 Windows XP 下 Tomcat 与 Apache 服务器集成需要准备的软件

| 软　件 | 下 载 位 置 | 本书附赠光盘上的位置 |
|---|---|---|
| 基于 Windows XP 的 Apache HTTP 服务器软件 | http://httpd.apache.org/ | software/ apache_2.0.63-win32-x86-no_ssl.msi |
| JK 插件 | http://tomcat.apache.org/download-connectors.cgi | sourcecode/chapter26 /windows_apache/mod_jk.so |

#### 1．安装 Apache 服务器

运行 apache_2.0.63-win32-x86-no_ssl.msi，就启动了 Apache 服务器的安装程序，只要按默认设置进行安装即可。如果安装成功，会自动在 Windows 中加入 Apache HTTP 服务，如图 26-2 所示。


图 26-2　加入到 Windows 服务列表中的 Apache 服务

假定 Apache 服务器的根目录为<APACHE_HOME>，在其 conf 子目录下有一个配置文件 httpd.conf。如果 Apache 安装在本机上，并且采用默认的 80 端口作为 HTTP 端口，则在 httpd.conf 文件中会看到如下属性：

```
Listen 80
```

在操作系统的【开始】→【程序】→【Apache HTTP Server 2.0】→【Control Apache Server】菜单中，提供了重启（Restart）、启动（Start）和关闭（Stop）Apache 服务器的子菜单。

**Tips**

应该确保操作系统的 80 端口没有被占用，否则 Apache 服务器无法启动。

当 Apache 服务器启动后，就可以通过访问 Apache 的测试页来确定是否安装成功。访问 http://localhost，如果出现如图 26-3 所示的网页，就说明 Apache 已经安装成功了。

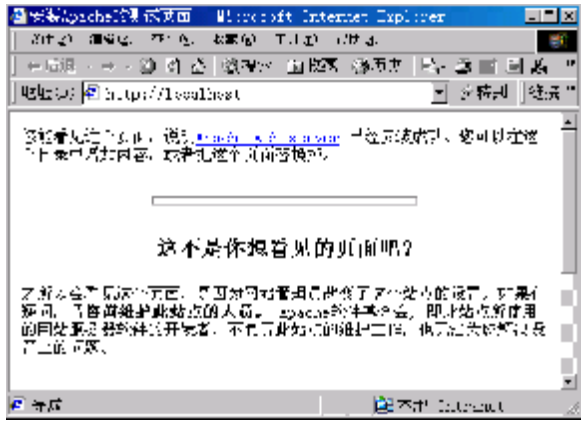

图 26-3　Apache 服务器的测试网页

2．在 Apache 服务器中加入 JK 插件

要在 Apache 服务器中加入 JK 插件，只要把 mod_jk.so 复制到<APACHE_HOME>/modules 目录下即可。

3．创建 workers.properties 文件

Apache 服务器把 Tomcat 看做是为自己工作的工人（worker）。workers.properties 文件用于配置 Tomcat 的信息，它的存放位置为<APACHE_HOME> /conf/workers.

properties。在本书附赠光盘的 sourcecode/chapter26/windows_apache 目录下提供了 workers.properties 文件，它的内容如下（“#”后面为注释信息）：

```
worker.list=worker1
worker.worker1.port=8009  #工作端口，若没占用则不用修改
worker.worker1.host=localhost  #Tomcat 服务器的地址
worker.worker1.type=ajp13  #类型
worker.worker1.lbfactor=1  #负载平衡因数
```

以上文件中的属性描述参见表 26-2。

表 26-2　workers.properties 文件的属性

| 属　　性 | 描　　述 |
|---|---|
| worker.list | 指定 Tomcat 服务器名单 |
| worker.worker1.port | 指定 Tomcat 服务器使用的 AJP 端口 |
| worker.worker1.host | 指定 Tomcat 服务器的 IP 地址 |
| worker.worker1.type | 指定 Tomcat 服务器与 Apache 服务器之间的通信协议 |
| worker.worker1.lbfactor | 指定负载平衡因数（Load Balance Factor）。只有在使用了负载平衡器(LoadBalancer)的情况下，这个属性才有意义 |

以上 worker.list 指定 Tomcat 服务器名单。例如“worker.list=worker1”表示只有一个 Tomcat 服务器，名为“worker1”。再例如“worker.list=worker1,worker2”表示有两个 Tomcat 服务器，分别名为“worker1”和“worker2”。worker.worker1.port 及 worker.worker1.host 用于设置名为“worker1”的 Tomcat 服务器的有关属性，如果要设置 worker2 的 port 属性，则可以采用“worker.worker2.port=8109”的形式。

4．修改 Apache 服务器的配置文件 httpd.conf

打开<APACHE_HOME>/conf/httpd.conf 文件，在其末尾加入以下内容：

```
# Using mod_jk.so to redirect dynamic calls to Tomcat
LoadModule  jk_module  modules/mod_jk.so
JkWorkersFile  conf/workers.properties
JkLogFile  logs/mod_jk.log
JkLogLevel  debug
JkMount  /*.jsp  worker1
JkMount  /helloapp/*  worker1
```

在本书附赠光盘的 sourcecode/chapter26/windows_apache/httpd_modify.conf 文件中提供了以上内容，它指示 Apache 服务器加载 JK 插件，并且为 JK 插件设置相关属性，这些属性的描述参见表 26-3。

表 26-3　JK 插件的相关属性

| 属　　性 | 描　　述 |
|---|---|
| LoadModule | 指定加载的 JK 插件 |
| JkWorkersFile | 指定 JK 插件的工作文件 |
| JkLogFile | 指定 JK 插件使用的日志文件，在实际配置中，可以通过查看这个日志文件，来跟踪 JK 插件的运行过程，这对排错很有用 |

| | |
|---|---|
| JkLogLevel | 指定 JK 插件的日志级别，可选值包括 debug、info 和 error 等 |
| JkMount | 指定 JK 插件处理的 URL 映射信息 |

JkMount 用来指定 URL 映射信息，“JkMount /*.jsp worker1”表示“/*.jsp”形式的 URL 都由 worker1 代表的 Tomcat 服务器来处理;“JkMount /helloapp/* worker1”表示访问 helloapp 应用的 URL 也都由 worker1 来处理。

5．测试配置

重启 Tomcat 服务器和 Apache 服务器，并通过浏览器访问 http://localhost/index.jsp，如果出现 Tomcat 的默认主页，就说明配置已经成功。此外，如果在 Tomcat 服务器上已经发布了 helloapp 应用（把本书附赠光盘的 sourcecode/chapter26 目录下的 helloapp 目录复制到<CATALINA_HOME>/webapps 目录下），则可以访问 http://localhost/helloapp/hello.htm，如果正常返回 helloapp 应用的 hello.htm 网页，说明配置已经成功。如果配置有误，可以查看 JK 插件生成的日志信息，它有助于查找错误原因。在 Apache 服务器的配置文件 httpd.conf 中设定该日志文件的存放位置为<APACHE_HOME>/logs/mod_jk.log。

## 26.3 在 Linux 下 Tomcat 与 Apache 服务器集成

在 Linux 下 Tomcat 与 Apache 服务器集成的步骤与在 Windows XP 下非常相似。在 Linux 下 Tomcat 与 Apache 服务器集成需要准备的软件参见表 26-4。

表 26-4　在 Linux 下 Tomcat 与 Apache 服务器集成需要准备的软件

| 软　　件 | 下 载 位 置 | 本书附赠光盘上的位置 |
| --- | --- | --- |
| 基于 Linux 的 Apache HTTP 服务器软件 | http://httpd.apache.org/ | software/httpd-2.0.63.tar.gz |
| JK 插件 | http://tomcat.apache.org/download-connectors.cgi | sourcecode/chapter26 /linux_apache/mod_jk_linux.so |

下文介绍在 Linux（以 RedHat 为例）下把 Tomcat 与 Apache 服务器集成的方法。

1．安装 Apache 服务器

以下是在 Linux 下安装 Apache 服务器的步骤。

（1）建立 httpd 用户，把 httpd-2.0.63.tar.gz 文件复制到/tmp 目录下。

（2）将 httpd-2.0.63.tar.gz 文件解压，命令为：

```
gzip -d httpd-2.0.63.tar.gz
tar xvf httpd-2.0.63.tar
```

（3）用超级用户账号登录 Linux，命令为：

```
su
```

（4）转到/tmp/httpd-2.0.63 目录，配置 Apache 服务器，命令为：

```
./configure --prefix =/home/httpd
```

“--prefix”选项用来设定 Apache 的安装目录。根据以上设置，Apache 将被安装到/home/httpd 目录。

（5）编译 Apache，命令为：

```
make。
```

（6）安装 Apache，命令为：

```
make install。
```

（7）在安装好以后，假定 Apache 的根目录为<APACHE_HOME>，打开<APACHE_HOME> /conf/httpd.conf 文件，配置“Listen”和“ServerName”属性：

```
Listen 80
ServerName localhost
```

（8）转到<APACHE_HOME>/bin 目录，通过运行 apachectl configtest 命令，来测试安装是否成功。如果显示 Syntax ok，则表示安装成功。

启动 Apache 服务器的命令为：

```
<APACHE_HOME>/bin/apachect1 start。
```

终止 Apache 服务器的命令为：

```
<APACHE_HOME>/bin/apachect1 stop。
```

**Tips**

应该确保操作系统的 80 端口没有被占用，否则 Apache 服务器无法启动。

也可以通过访问 Apache 的测试页来确定是否安装成功。访问 http://localhost，如果出现如本章 26.2 节的图 26-3 所示的网页，就说明 Apache 已经安装成功了。

2．在 Apache 服务器中加入 JK 插件

要在 Apache 中加入 JK 插件，只要把 mod_jk_linux.so 复制到<APACHE_HOME>/modules 目录下即可。

3．创建 workers.properties 文件

在<APACHE_HOME>/conf 目录下创建以下 workers.properties 文件。此外，在本书附赠光盘的 sourcecode/chapter26/linux_apache 目录下也提供了该文件：

```
worker.list=worker1
worker.worker1.port=8009  #工作端口,若没占用则不用修改
worker.worker1.host=localhost  #Tomcat 服务器的地址
worker.worker1.type=ajp13  #类型
worker.worker1.lbfactor=1  #负载平衡因数
```

4．修改 Apache 服务器的配置文件 httpd.conf

打开<APACHE_HOME>/conf/httpd.conf 文件，在其末尾加入以下内容：

```
LoadModule jk_module modules/ mod_jk_linux.so
JkWorkersFile conf/workers.properties
JkLogFile logs/mod_jk.log
JkLogLevel debug
JkMount /*.jsp worker1
JkMount /helloapp/* worker1
```

在本书附赠光盘的 sourcecode/chapter26/linux_apache 目录下的 httpd_modify.conf 文件中提供了以上内容。

5．测试配置

重启 Tomcat 服务器和 Apache 服务器。通过浏览器访问 http://localhost/index.jsp，如果出现 Tomcat 的默认主页，说明配置已经成功。此外，如果在 Tomcat 服务器上已经发布了 helloapp 应用（把本书附赠光盘的 sourcecode/chapter26 目录下的 helloapp 目录复制到 <CATALINA_HOME>/webapps 目录下），则可以访问 http://localhost/helloapp/hello.htm，如果正常返回 helloapp 应用的 hello.htm 网页，说明配置已经成功。如果配置有误，可以查看 JK 插件生成的日志信息，它有助于查找错误原因。在 Apache 的配置文件 httpd.conf 中设定该日志文件的存放位置为：<APACHE_HOME>/logs/mod_ jk.log。

## 26.4 Tomcat 与 IIS 服务器集成

IIS（Internet Information Service）服务器是微软开发的功能强大的 Web 服务器，它为创建和开发电子商务提供了安全的 Web 平台。把 Tomcat 与 IIS 集成，可以扩展 IIS

的功能，使它支持 Java Web 应用。

如果在 Windows 操作系统中还未安装 IIS，可以选择操作系统的【控制面板】→【添加/删除程序】→【添加/删除 Windows 组件】菜单，然后安装 IIS，在安装过程中要提供 Windows 操作系统的安装光盘。

### 26.4.1　准备相关文件

在开始本节的操作之前，假定在机器上安装了 IIS 服务器，接下来应该准备好以下 3 个文件。

1．JK 插件

在本书附赠光盘的 sourcecode/chapter26/iis 目录下提供了用于 IIS 的 JK 插件：isapi_redirect.dll，此外，也可以到以下地址下载最新的 JK 插件：

```
http://tomcat.apache.org/download-connectors.cgi
```

可以把 JK 插件 isapi_redirect.dll 复制到<CATALINA_HOME>/bin 目录下。

2．workers.properties 文件

在<CATALINA_HOME>/conf 目录下创建如下的 workers.properties 文件，在本书附赠光盘的 sourcecode/chapter26/iis 目录下也提供了该文件：

```
worker.list=worker1
worker.worker1.port=8009  #工作端口,若没占用则不用修改
worker.worker1.host=localhost  #Tomcat 服务器的地址
worker.worker1.type=ajp13  #类型
worker.worker1.lbfactor=1  #负载平衡因数
```

3．uriworkermap.properties 文件

在<CATALINA_HOME>/conf 目录下创建如下的 uriworkermap.properties 文件，它为 JK 插件指定 URL 映射。在本书附赠光盘的 sourcecode/chapter26/iis 目录下也提供了该文件：

```
/*.jsp=worker1
/helloapp/*=worker1
```

**Tips**

尽管把以上 3 个文件都放在 Tomcat 目录下，但其实 Tomcat 服务器并不会访问这些文件。以上给出的是按照惯例的一种配置，事实上，也可以把这些文件放在文件系统的其他地方。

### 26.4.2　编辑注册表

在配置 Apache 和 Tomcat 集成时，JK 插件的属性是在 Apache 的配置文件 httpd.conf 中设置的。在配置 IIS 和 Tomcat 集成时，应该在操作系统的注册表中设置 JK 插件的属性，以下是操作步骤。

（1）在 Windows XP 中通过 regedit 命令编辑注册表，创建一个新的键：

HKEY_LOCAL_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi Redirector \1.0，如图 26-4 所示。

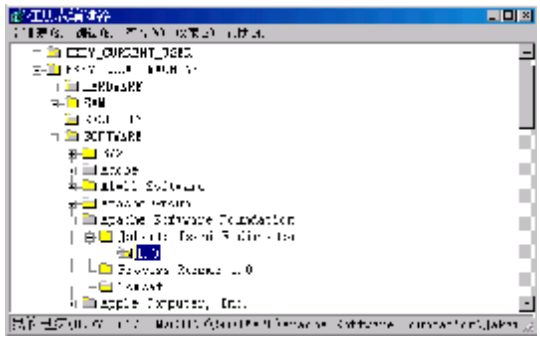

图 26-4　在注册表中创建 Jakarta Isapi Redirector\1.0 键

（2）在 Jakarta Isapi Redirector\1.0 键下面创建新的字符串，参见表 26-5，创建好之后的注册表如图 26-5 所示。

表 26-5　在 Jakarta Isapi Redirector\1.0 键下面创建的字符串

| 字　符　串 | 字 符 串 值 | 描　　述 |
|---|---|---|
| extension_uri | /jakarta/isapi_redirect.dll | 指定访问 isapi_redirect.dll 文件的 uri，在 IIS 中将创建名为 jakarta 的虚拟目录，在该目录下包含 isapi_redirect.dll 文件，参见本章 26.4.3 节 |
| log_file | C:\tomcat\logs\isapi.log | 指定 JK 插件使用的日志文件，在实际配置中，可以通过查看这个日志文件，来跟踪 JK 插件的运行过程，这对排错很有用 |
| log_level | debug | 指定 JK 插件的日志级别，可选值包括 debug、info 和 error 等 |
| worker_file | C:\tomcat\conf\workers.properties | 指定 JK 插件的工作文件 |
| worker_mount_file | C:\tomcat\conf\uriworkermap.properties | 指定 JK 插件的 URL 映射文件 |

图 26-5　在 Jakarta Isapi Redirector\1.0 键下面创建新的字符串

在本书附赠光盘的 sourcecode/chapter26/iis 目录下提供了注册表编辑文件 jk.reg，如果不想按照以上方式手工修改注册表，也可以直接运行 jk.reg 文件（选中这个文件再双击鼠标即可），它会把以上配置内容自动添加到注册表中。jk.reg 的内容如下：

```
Windows Registry Editor Version 5.00
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Apache Software Foundation\Jakarta Isapi Redirector
\1.0]
"extension_uri"="/jakarta/isapi_redirect.dll"
"log_file"="C:\\tomcat\\logs\\isapi.log"
"log_level"="debug"
"worker_file"="C:\\tomcat\\conf\\workers.properties"
"worker_mount_file"="C:\\tomcat\\conf\\uriworkermap.properties"
```

在运行 jk.reg 文件之前，应该把文件中的“C:\\tomcat”目录替换为读者本地机器上的 Tomcat 的实际安装目录。

### 26.4.3　在 IIS 中加入“jakarta”虚拟目录

当注册表修改以后，应该在 IIS 中加入名为“jakarta”的虚拟目录，它是 JK 插件所在的目录，以下是操作步骤。

（1）选择操作系统的【控制面板】→【管理工具】→【Internet 服务管理器】选项，打开 Internet 信息服务管理器，如图 26-6 所示。

（2）选中【默认 Web 站点】选项，然后单击鼠标右键，在其下拉菜单中选择【新建】→【虚拟目录】选项，如图 26-7 所示。创建一个虚拟目录，名为“jakarta”，对应的实际文件资源路径应该是 isapi_redirect.dll 文件所在的目录<CATALINA_HOME>/bin。

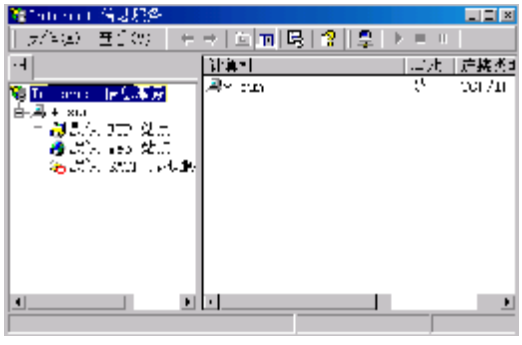

图 26-6　Internet 信息服务管理器窗口

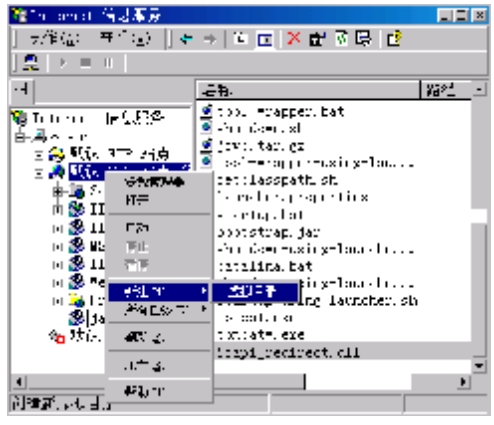

图 26-7　创建虚拟目录

（3）修改刚刚创建的 jakarta 虚拟目录的属性，将其执行许可权限设为“脚本和可执行程序”，如图 26-8 所示。这步操作很重要，它将保证在注册表中设置的 extension_uri 对应的/jakarta/isapi_redirect.dll 可以被执行。如果漏掉这步操作，会导致无法访问 Tomcat 中的 Servlet/JSP 组件。


图 26-8　修改 jakarta 虚拟目录的执行许可权限

### 26.4.4　把 JK 插件作为 ISAPI 筛选器加入到 IIS 中

在 IIS 中加入名为“jakarta”的虚拟目录后，还应该把 JK 插件作为 ISAPI 筛选器（也称为过滤器）加入到 IIS 中，以下是操作步骤。

（1）在【Internet 信息服务】主窗口的目录树中选择 IIS 主机节点，然后单击鼠标右键，在其下拉菜单中选择【属性】选项，如图 26-9 所示。在出现的窗口中单击【编辑】按钮，打开 IIS 主机的属性窗口，如图 26-10 所示。

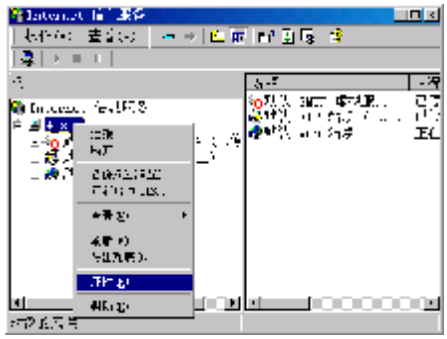

图 26-9　配置 IIS 主机的属性

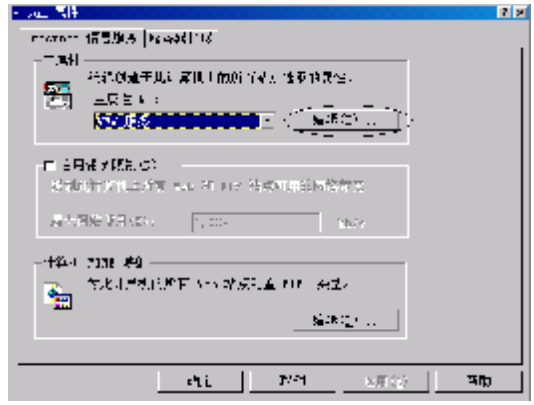

图 26-10　IIS 主机的属性窗口

（2）在主属性的 WWW 服务区域单击【编辑】按钮，打开 WWW 服务主属性窗口，增加新的 ISAPI 筛选器，筛选器名称为“jakarta”，可执行文件为<CATALINA_HOME>/bin /isapi_redirect.dll，如图 26-11 所示。

（3）重新启动 IIS 服务器，如果配置正常，在 WWW 服务主属性的 ISAPI 筛选器子窗口中，新加的 jakarta 筛选器的状态应该变为绿色向上的箭头，如图 26-12 所示。

图 26-11　增加新的 ISAPI 筛选器

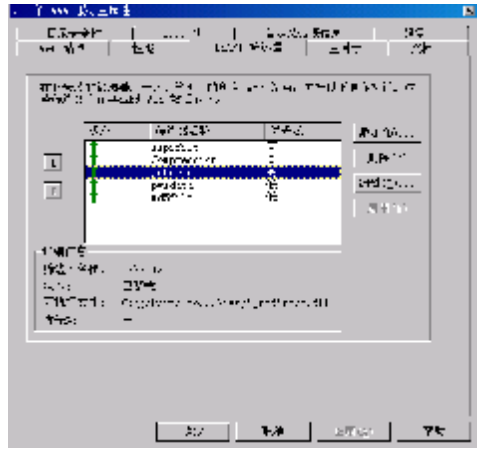

图 26-12　jakarta 筛选器被装载

如果如图 26-12 所示的 jakarta 筛选器的箭头仍然为红色，表明 JK 插件没有装载成功，这可能是由于以下原因引起的：

- 在<CATALINA_HOME>/bin 目录下不存在 isapi_redirect.dll 文件。
- 没有正确地按照本章 26.4.2 节的步骤编辑注册表。

### 26.4.5　测试配置

重启 Tomcat 服务器和 IIS 服务器，并通过浏览器访问 http://localhost/index.jsp。如果出现 Tomcat 的默认主页，说明配置已经成功。此外，如果在 Tomcat 服务器上已经发布了 helloapp 应用（把本书附赠光盘的 sourcecode/chapter26 目录下的 helloapp 目录复制到<CATALINA_HOME>/webapps 目录下），则可以访问 http://localhost/helloapp/hello.htm，如果正常返回 helloapp 应用的 hello.htm 网页，说明配置已经成功。如果配置有误，可以查看 JK 插件生成的日志信息，它有助于查找错误原因。在注册表中设定该日志文件的存放位置为<CATALINA_HOME>/logs/isapi.log。

## 25.5　Tomcat 集群

在实际应用中，如果网站的访问量非常大，为了提高访问速度，可以将多个 Tomcat 服务器与 Apache 服务器集成，让它们共同分担运行 Servlet/JSP 组件的任务。多个 Tomcat 服务器构成了一个集群（Cluster）系统，共同为客户提供服务。集群系统具有以下优点。

- 高可靠性：当一台服务器发生故障时，集群系统能够自动把工作任务转交给另一台正常运行的服务器，以便为用户提供透明的不间断的服务。
- 高性能计算：充分利用集群中的每一台服务器的软件和硬件资源，实现复杂运算的并行处理，通常用于科学计算领域，比如基因分析和化学分析等。
- 负载平衡：把负载压力根据某种算法合理分配到集群中的每一台服务器上，

以减轻单个服务器的压力，降低对单个服务器的硬件和软件要求。

图 26-13 显示了由 JK 插件和两个 Tomcat 服务器构成的集群系统。集群系统的正常运作离不开以下两个组件。

- JK 插件的 loadbalancer（负载平衡器）：根据在 workers.properties 文件中预先配置的 lbfactor（负载平衡因数），负责为集群系统中的 Tomcat 服务器分配工作负荷，以实现负载平衡。
- 每个 Tomcat 服务器上的集群管理器（SimpleTcpCluster）：每个 Tomcat 服务器上的集群管理器通过 TCP 连接与集群系统中的其他 Tomcat 服务器通信，以实现 HTTP 会话的复制，以及把 Web 应用发布到集群系统中的每个 Tomcat 服务器上。

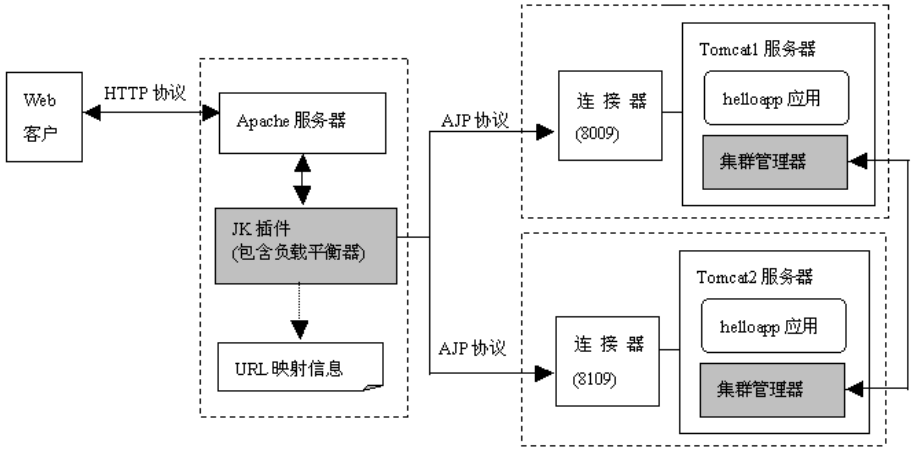


图 26-13　Tomcat 集群系统

## 26.5.1　配置集群系统的负载平衡器

假定在 Windows 中，把 Apache 服务器和两个 Tomcat 服务器集成。为了方便读者在本地机器上做实验，这两个 Tomcat 服务器和 Apache 服务器都运行在同一台机器上，Tomcat1（根目录为 C:\tomcat1）使用的 AJP 端口为 8009，Tomcat2（根目录为 C:\tomcat2）使用的 AJP 端口为 8109。如果两个 Tomcat 服务器运行在不同的机器上，那么它们可以使用相同的 AJP 端口。

以下是把 Apache 和这两个 Tomcat 服务器集成，以及配置负载平衡器的步骤。

（1）把 mod_jk.so 复制到<APACHE_HOME>/modules 目录下。

（2）在<APACHE_HOME>/conf 目录下创建如下的 workers.properties 文件（注意粗体部分的内容）：

```
worker.list=worker1,worker2,loadbalancer

worker.worker1.port=8009  #工作端口，若没占用则不用修改
worker.worker1.host=localhost  #Tomcat 服务器的地址
worker.worker1.type=ajp13  #类型
worker.worker1.lbfactor=100  #负载平衡因数
```

```
worker.worker2.port=8109  #工作端口，若没占用则不用修改
worker.worker2.host=localhost  #Tomcat 服务器的地址
worker.worker2.type=ajp13  #类型
worker.worker2.lbfactor=100  #负载平衡因数

worker.loadbalancer.type=lb
worker.loadbalancer.balanced_workers=worker1, worker2
worker.loadbalancer.sticky_session=false
worker.loadbalancer.sticky_session_force=false
```

以上文件创建了两个监听 AJP 端口的 worker：worker1 和 worker2。worker1 和 worker2 分别代表两个 Tomcat 服务器，它们由负载平衡器来进行调度。由于本实验中的 worker1 和 worker2 运行在同一个机器上，所以应该使 worker1.port 和 worker2.port 指向不同的端口。worker1 和 worker2 的 lbfactor 属性设定工作负荷，在本例中，worker1 和 worker2 的 lbfactor 属性都为 100，因此会分担同样的工作负荷。

以上文件还配置了一个名为“loadbalancer”的 worker，它是负载平衡器，有 sticky_session 和 sticky_session_force 属性，这两个属性的作用在本章 26.5.2 节的最后做了介绍。在本书附赠光盘的 sourcecode/chapter26/windows_apache/loadbalance 目录下提供了上述 workers.properties 文件。

（3）修改<APACHE_HOME>/conf/httpd.conf 文件，在文件末尾加入如下内容：

```
# Using mod_jk.so to redirect dynamic calls to Tomcat
LoadModule jk_module modules/mod_jk.so
JkWorkersFile conf/workers.properties
JkLogFile logs/mod_jk.log
JkLogLevel debug
JkMount /*.jsp loadbalancer
JkMount /helloapp/* loadbalancer
```

当客户请求为“/*.jsp”或“/helloapp/*”形式的 URL 时，该请求都由 loadbalancer 来负责转发，它根据在 workers.properties 文件中为 worker1 和 worker2 设定的 lbfactor 属性，来决定如何调度它们。在本书附赠光盘的 sourcecode/chapter26/windows_apache/loadbalance 目录下的 httpd_modify.conf 文件中提供了上述配置代码。

**Tips**

只有在使用了 loadbalancer 的情况下，workers.properties 文件中的 worker 的 lbfactor 属性才有意义，lbfactor 取值越大，表示分配给 Tomcat 服务器的工作负荷越大。

（4）分别修改两个 Tomcat 服务器的 conf/server.xml 文件中的 AJP 连接器的端口，确保它们和 workers.properties 文件中的配置对应。此外，在使用了 loadbalancer 后，要求 worker 的名字和 Tomcat 的 server.xml 文件中的<Engine>元素的 jvmRoute 属性一致。

所以应该分别修改两个 Tomcat 的 server.xml 文件，把它们的<Engine>元素的 jvmRoute 属性分别设为 worker1 和 worker2。以下是修改后的两个 Tomcat 服务器的<Engine>元素：

```
Tomcat 服务器 1:
<Engine name="Catalina" defaultHost="localhost" jmvRoute="worker1">
```

```
Tomcat 服务器 2:
<Engine name="Catalina" defaultHost="localhost" jmvRoute="worker2">
```

（5）在完成以上步骤后，分别启动两个 Tomcat 服务器和 Apache 服务器，然后访问 http://localhost/index.jsp，会出现 Tomcat 服务器的默认主页。由于此时由 loadbalancer 来调度 Tomcat 服务器，因此不能断定到底访问的是哪个 Tomcat 服务器的 index.jsp，这对于 Web 客户来说是透明的。

在进行以上实验时，两个 Tomcat 服务器都在同一台机器上运行，所以应该确保它们没有使用相同的端口。在 Tomcat 的默认的 server.xml 中，一共配置了以下 3 个端口：

```
<Server port="8005" shutdown="SHUTDOWN" >
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<Connector port="8080" … />
<!-- Define an AJP 1.3 Connector on port 8009 -->
<Connector port="8009" … />
```

由于两个 Tomcat 服务器都在同一台机器上运行，所以至少应该对其中一个 Tomcat 服务器的以上 3 个端口号都进行修改。例如把第 2 个 Tomcat 服务器的端口号改为：

```
<Server port="8105" shutdown="SHUTDOWN" >
<!--Define a non-SSL Coyote HTTP/1.1 Connector on port 8080-->
<Connector port="8180" … />
<!--Define an AJP 1.3 Connector on port 8009-->
<Connector port="8109" …/>
```

此外，在把 Tomcat 和其他 HTTP 服务器集成时，Tomcat 主要负责处理 HTTP 服务器转发过来的客户请求，并且通常不会直接接受 HTTP 请求。因此为了提高 Tomcat 的运行性能，也可以关闭 Tomcat 的 HTTP 连接器，方法为在 server.xml 中把 Tomcat 的 HTTP Connector 的配置注释掉：

```
<!-- Define a non-SSL Coyote HTTP/1.1 Connector on port 8080 -->
<!-- <Connector port="8080" … /> -->
```

### 26.5.2 配置集群管理器

假定已经按照本章 26.5.1 节的步骤使 Apache 服务器与两个 Tomcat 服务器集成，接着把本书附赠光盘的 sourcecode/chapter26 目录下的 helloapp 目录分别复制到两个 Tomcat 服务器的<CATALINA_HOME>/webapps 目录下。在 helloapp 应用中有一个 test.jsp 文件，它的源代码如下：

```
<html>
<head>
  <title>helloapp</title>
</head>
<body>
<%
System.out.println("call test.jsp"); //在 Tomcat 控制台上打印一些跟踪数据
%>
SessionID: <%=session.getId() %>

</body>
</html>
```

分别启动两个 Tomcat 服务器和 Apache 服务器，然后打开浏览器，多次访问 http://localhost/helloapp/test.jsp，浏览器及 Tomcat 服务器的打印结果如图 26-14 所示。根据 test.jsp 在 Tomcat 控制台上的打印语句，可以判断出客户端每次请求访问 test.jsp 时，由哪个 Tomcat 服务器执行 test.jsp。

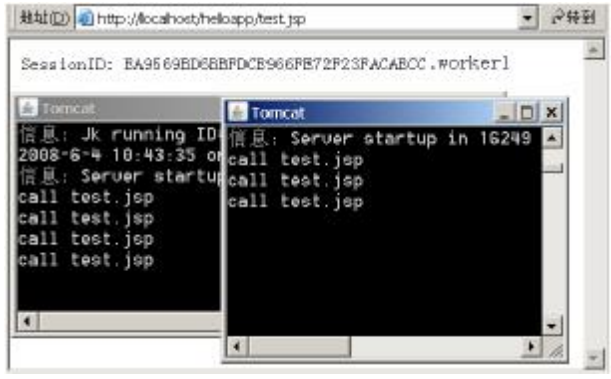


图 26-14　通过浏览器多次访问 test.jsp 的效果

本书第 9 章的 9.2 节（HttpSession 的生命周期及会话范围）已经讲过，在通过同一个浏览器进程多次访问同一个 Web 应用中支持会话的 JSP 页面时，这些请求始终是处于同一个会话中的，因此 Session ID 应该是不变的。

但是在上述实验中，在通过同一个浏览器进程多次访问 helloapp 应用的 test.jsp 时，从浏览器上看到的 Session ID 的值每次都不一样。这是因为客户端每次访问 test.jsp 的请求都由不同的 Tomcat 服务器来响应，而这两个 Tomcat 服务器之间没有进行会话的同步。

为了解决上述问题，需要启用 Tomcat 的集群管理器（SimpleTcpCluster），步骤如下。

（1）分别修改 Tomcat1 和 Tomcat2 的 conf/server.xml 文件，在其中的<Engine>元素内都加入以下<Cluster>子元素，使得 Tomcat 能够启用集群管理器。

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
             channelSendOptions="8">

  <Manager className="org.apache.catalina.ha.session.DeltaManager"
             expireSessionsOnShutdown="false"
             notifyListenersOnReplication="true"/>
  <Channel className="org.apache.catalina.tribes.group.GroupChannel">
    <Membership className="org.apache.catalina.tribes.membership.McastService"
             bind="127.0.0.1"
             address="228.0.0.4"
             port="45564"
             frequency="500"
             dropTime="3000"/>
    <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
             address="auto"
             port="4000"
             autoBind="100"
             selectorTimeout="5000"
             maxThreads="6"/>
```

```
            <Sender className="org.apache.catalina.tribes.transport.
                ReplicationTransmitter">
               <Transport className="org.apache.catalina.tribes.transport.nio.
                   PooledParallelSender"/>
            </Sender>

            <Interceptor  className="org.apache.catalina.tribes.group.interceptors.
                TcpFailureDetector"/>
            <Interceptor  className="org.apache.catalina.tribes.group.interceptors.
                MessageDispatch15Interceptor"/>
          </Channel>

          <Valve className="org.apache.catalina.ha.tcp.ReplicationValve" filter=""/>
          <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>

          <Deployer className="org.apache.catalina.ha.deploy.FarmWarDeployer"
                    tempDir="/tmp/war-temp/"
                    deployDir="/tmp/war-deploy/"
                    watchDir="/tmp/war-listen/"
                    watchEnabled="false"/>

          <ClusterListener
             className="org.apache.catalina.ha.session.JvmRouteSessionIDBinderListener"/>
          <ClusterListener className="org.apache.catalina.ha.session.
              ClusterSessionListener"/>

        </Cluster>
```

在本书附赠光盘的 sourcecode/chapter26/windows_apache/loadbalance 目录下的 server_modify.xml 文件中提供了上述配置代码。关于<Cluster>元素的详细用法可以参考以下文档：

```
<CATALINA_HOME>\webapps\docs\cluster-howto.html
<CATALINA_HOME>\webapps\docs\config\cluster.html
```

在本书附赠光盘的 sourcecode/chapter26/windows_apache/loadbalance 目录下还提供了 server1.xml 和 server2.xml 文件，它们分别是 Tomcat1 和 Tomcat2 的配置文件，这两个文件已经按照本章 26.5.1 和 26.5.2 节的步骤做了相应的设置。读者可以删除 Tomcat1 和 Tomcat2 的 conf 目录下的原有 server.xml 文件，再把 server1.xml 和 server2.xml 文件分别复制到 Tomcat1 和 Tomcat2 的 conf 目录下，并把它们改名为“server.xml”。

（2）分别修改 Tomcat1 和 Tomcat2 的 helloapp 应用的 web.xml 文件，在其中加入<distributable/>元素：

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<web-app>
  <distributable/>
</web-app>
```

在集群系统中，如果在一个 Tomcat 服务器中的一个 Web 应用的 web.xml 文件中设置了<distributable/>元素，那么当 Tomcat 服务器启动这个 Web 应用时，会为它创建由 server.xml 文件中的<Cluster>元素的<Manager>子元素指定的会话管理器。在本实验

中，会话管理器为 DeltaManager，它能够把一个服务器节点中的会话信息复制到集群系统中的所有其他服务器节点中。

（3）分别启动两个 Tomcat 服务器和 Apache 服务器，然后打开一个浏览器，多次访问 http://localhost/helloapp/test.jsp，会看到 test.jsp 页面中的 Session ID 始终保持不变。这表明无论浏览器是访问 Tomcat1 中的 test.jsp，还是访问 Tomcat2 中的 test.jsp，都始终处于同一个会话中。Tomcat1 与 Tomcat2 中的关于 helloapp 应用的会话信息是同步的。

在配置 Tomcat 集群系统时，有以下注意事项。

（1）为了保证在集群系统中，会话数据能在所有 Tomcat 服务器上正确地复制，应该保证存放在会话范围内的所有属性都实现了 java.io.Serializable 接口。

（2）集群系统中的 Tomcat 服务器之间通过组播的形式来通信。如果 Tomcat 所在的机器上有多个网卡，或者配置了虚拟网卡，可能会导致组播失败，从而无法正常复制会话。假定集群系统中的 Tomcat1 已经启动，而在启动 Tomcat2 时控制台输出如下信息：

```
信息: Manager [localhost#/helloapp]: skipping state transfer. No members active
in cluster group.
```

以上信息表明 Tomcat2 没有识别到集群系统中的 Tomcat1，说明组播失败。解决这一问题的方法是：在配置<Cluster>元素的<Membership>子元素时，确保设置了如下 bind 属性，它用于明确地设置组播绑定地址。

```
<Membership className="org.apache.catalina.tribes.membership.McastService"
            bind="127.0.0.1"  … />
```

（3）如果集群系统规模较小，即其中的 Tomcat 服务器数目不多，可以采用 DeltaManager 会话管理器，它能够把一个服务器节点中的会话信息复制到集群系统中的所有其他服务器节点中。DeltaManager 会话管理器不适合用于规模很大的集群系统中，因为它会大大增加网络通信负荷。对于规模很大的集群系统，可以采用 BackupManager 会话管理器，它只会把一个服务器节点中的会话信息备份到集群系统中的其他单个服务器节点中。例如以下 server.xml 文件中的配置代码使用了 BackupManager 会话管理器：

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
            channelSendOptions="8">

  <Manager className="org.apache.catalina.ha.session.BackupManager"
             expireSessionsOnShutdown="false"
             notifyListenersOnReplication="true"
             mapSendOptions="6"/>
    …
</Cluster>
```

（4）<Membership>元素的 address 属性设定组播地址，本实验中把它设为 228.0.0.4。在运行本实验时，应该确保 Tomcat 所在的主机连在 Internet 上，否则无法访问该组播地址。

（5）本章 26.5.1 节在 workers.properties 文件中配置负载平衡器时设置了 sticky_

session 和 sticky_session_force 属性：

```
worker.loadbalancer.type=lb
worker.loadbalancer.balanced_workers=worker1, worker2
worker.loadbalancer.sticky_session=false
worker.loadbalancer.sticky_session_force=false
```

如果 sticky_session 的值为 true，就表示会话具有“粘性”。“粘性”意味着当用户通过浏览器 A 与 Tomcat1 开始了一个会话后，以后用户从浏览器 A 中发出的请求只要处于同一个会话中，负载平衡器就会始终让 Tomcat1 来处理请求。直观地理解，可以认为一个会话始终与集群系统中的一个 Tomcat 服务器“粘”在一起。当 sticky_session 的值为 true 时，集群系统不会进行会话复制。如果希望集群系统能进行会话复制，从而使得一个浏览器能与多个 Tomcat 服务器展开同一个会话，则应该把 sticky_session 设为 false。sticky_session 的默认值为 true。

当 sticky_session 设为 false 时，sticky_session_force 对集群系统没有什么影响，通常可以把它设为默认值 false。当 sticky_session 设为 true 时，则建议把 sticky_session_force 也设为 true。

假定 sticky_session 设为 true，当浏览器已经与集群系统中的 Tomcat1 服务器“粘”在一起，展开了会话后，如果这个 Tomcat1 服务器异常终止，此时会出现什么情况呢？如果 sticky_session_force 为 true，那么服务器端会向客户端返回状态代码为 500 的错误。如果 sticky_session_force 为 false，那么负载平衡器会把请求转发给集群系统中的其他 Tomcat2 服务器，假如在 Tomcat2 服务器中不存在同一个会话的信息，则当 Web 组件试图访问会话中的有关数据时可能会导致异常。

## 26.6 小结

本章介绍了通过 JK 插件来实现 Tomcat 与 Apache 及 IIS 服务器集成的步骤。Tomcat 提供了专门的 JK 插件来负责 Tomcat 和 HTTP 服务器的通信，并且 JK 插件安置在对方 HTTP 服务器上。当 HTTP 服务器接收到客户请求时，它会通过 JK 插件来过滤 URL，JK 插件根据预先配置好的 URL 映射信息，来决定是否要把客户请求转发给 Tomcat 服务器处理。Tomcat 与 Apache 及 IIS 服务器集成的异同之处参见表 26-6。

表 26-6 Tomcat 与 Apache 及 IIS 服务器集成的异同之处

| | Tomcat 与 Apache 集成 | Tomcat 与 IIS 集成 |
|---|---|---|
| JK 插件 | mod_jk.so | isapi_redirect.dll |
| JK 插件的工作文件 | workers.properties 文件 | workers.properties 文件 |
| 设置 JK 插件属性 | 在 Apache 的配置文件 httpd.conf 中设置 | 在注册表中设置 |
| 设置 URL 映射信息 | 在 Apache 的配置文件 httpd.conf 中设置 | 在 uriworkermap.properties 文件中设置 |
| 加载 JK 插件 | 把 JK 插件复制到<APACHE_HOME>/modules 目录下，在 Apache 的配置文件 httpd.conf 中设置 LoadModule 属性 | 把 JK 插件所在的目录作为 IIS 的虚拟目录，并把 JK 插件作为 ISAPI 筛选器加入到 IIS 中 |

## 26.7　思考题

1．关于把 Tomcat 与其他 HTTP 服务器集成，以下哪些说法正确？（多选）

（a）Tomcat 与其他 HTTP 服务器之间采用 HTTP 协议通信

（b）Tomcat 与其他 HTTP 服务器之间采用 AJP 协议通信

（c）Web 客户与其他 HTTP 服务器之间采用 HTTP 协议通信

（d）要求访问 JSP/Servlet 的客户请求先到达其他 HTTP 服务器，然后由 JK 插件转发给 Tomcat

2．把 Apache 服务器与 Tomcat 集成，涉及哪些步骤？（多选）

（a）把 worker.properties 文件放在<APACHE_HOME>/conf 目录下

（b）修改<APACHE_HOME>/conf 目录下的 httpd.conf 文件，增加用于加载 JK 插件及设置 JK 插件属性的代码

（c）把 mod_jk.so 文件复制到<APACHE_HOME>/modules 目录下

（d）把 mod_jk.so 文件复制到<CATALINA_HOME>/lib 目录下

3．关于 JK 插件，以下哪些说法正确？（多选）

（a）JK 插件在 Apache 服务器中称为动态加载模块，在 IIS 服务器中称为 ISAPI 筛选器

（b）JK 插件运行在 Tomcat 服务器进程中，负责接收由其他 HTTP 服务器转发过来的客户请求

（c）在配置 JK 插件时，需要设定 JK 插件的工作文件（worker.properties）、日志文件、日志级别和 URL 映射信息等

（d）JK 插件的接口由 SUN 公司制定，各种类型的 Servlet 容器实现都可以通过 JK 插件来与其他 HTTP 服务器集成

4．关于 Tomcat 集群系统与 Apache 构成的集成环境，以下哪些说法正确？（多选）

（a）当 Tomcat 集群系统为客户端提供服务时，到底由哪个服务器节点来提供服务，这对客户端来说是透明的

（b）当 Tomcat 集群系统为客户端提供服务时，到底由哪个服务器节点来提供服务，这是由 JK 插件中的负载平衡器来调度的

（c）Tomcat 集群系统中的各个 Tomcat 服务器节点通过 AJP 协议进行通信

（d）Tomcat 集群系统中的各个 Tomcat 服务器节点通过组播的方式来进行 HTTP 会话的同步

**答案：**

1．b c d　　2．a b c　　3．a c　　4．a b d