INAP
INSTITUTO NACIONAL DE ADMINISTRACIÓN PÚBLICA

Curso online: **Instalación, Configuración y Administración de Apache + Tomcat**

Módulo 4. Cooperación entre Apache y Tomcat

Capítulo 3. mod_jk (conector JK 1.2)

Autores

Janine García Morera

Alexandra López de la Oliva Portugués

Julio Villena Román

Octubre de 2014

Índice de contenidos

CAPÍTULO 3 MOD_JK (CONECTOR JK1.2)

En este capítulo vamos a aprender a instalar, compilar y configurar el módulo de cooperación Apache-Tomcat `mod_jk`, para posteriormente definir una arquitectura de balanceo mediante el uso de `mod_jk`.

El Conector JK es un componente de servidor web que se utiliza para integrar de forma transparente Tomcat con un servidor Web tal como Apache o IIS. La comunicación entre el servidor web y Tomcat se hace sobre el protocolo JK, conocido también como AJP.

El módulo `mod_jk` es el módulo recomendado por ASF para realizar la interconexión entre el servidor Web Apache y Tomcat, ya que ofrece mejor rendimiento y es más estable que el resto de los conectores.

La configuración del módulo `mod_jk` se realiza fundamentalmente en la parte del servidor web, ya que a nivel de Tomcat el conector AJP está configurado por defecto.

Tomcat suele integrarse con un servidor web cuando queremos que el servidor web controle el contenido estático del sitio, y/o queremos aprovechar las características SSL del servidor web. En muchos entornos de aplicación, esta configuración tendrá un rendimiento mejor que si utilizamos sólo Tomcat independiente (*standalone*) con el conector Coyote.

En resumen, un servidor web espera solicitudes HTTP de los clientes. Cuando llegan las solicitudes (*requests*), el servidor hace lo necesario para satisfacerlas proporcionando el contenido adecuado. Si añadimos un contenedor de servlets cambiaremos este comportamiento.

Ahora, el servidor web, además de lo que sabe hacer, ha de:

- Cargar la librería de adaptación del contenedor de servlets e inicializarlas (antes de servir las requests)
- Cuando llega una petición, comprobar si lo que pide es un servlet. Si lo es, ha de pasarle la request al adaptador para que éste la maneja.

Mientras, el adaptador ha de saber qué peticiones tiene que servir, basándose normalmente en un patrón de la URL que le llega, y a dónde dirigir tales peticiones.

La cosa se complica si queremos configurar hosts virtuales, o si queremos que los desarrolladores puedan trabajar en el mismo servidor web, pero sobre distintas máquinas virtuales JVM de diferentes contenedores de servlets.

3.1. Terminología mod_jk

Primero hay que conocer los términos o componentes del módulo de cooperación.

- **Proceso worker**. Un worker es una instancia Tomcat que está en ejecución para cumplimentar requests de servlets que le llegan del servidor web. Normalmente hay un solo worker (el propio proceso Tomcat), pero a veces ejecutaremos workers múltiples para permitir balanceo de carga o particionamiento de los sitios web. Cada uno de los workers se identifica con el servidor web por el host donde está alojado, el puerto en el que escucha y el protocolo de comunicación que utiliza para intercambiar mensajes.
- **Proceso Worker in-process**: es un worker especial. En vez de trabajar como un proceso Tomcat que reside en otro proceso, el servidor abre una JVM y ejecuta Tomcat dentro del espacio de direcciones del proceso del servidor web.
- **Plug-in de Servidor Web / Redirector Tomcat (Web Server Plug-in / Tomcat Redirector)**. Para que Tomcat coopere con cualquier servidor web, necesita un "agente" que resida en el servidor web y le envíe a Tomcat las *requests* de servlets. Este es el plug-in de servidor, en este caso, el `mod_jk`. El redirector viene en forma de DLL o de objeto compartido, que se ponen dentro del espacio del servidor web.

3.2. ¿Cuáles son los pasos a seguir?

1. Instalación de los módulos necesarios.
2. Configuración del Plug-in. Tenemos que configurar el plug-in para que sepa donde están los distintos workers de Tomcat y a cuál de ellos ha de enviar las requests. Esta información, junto con algunos parámetros internos (como el nivel de traza), componen la configuración del plug-in.
3. Configuración del servidor Web. Todos los servidores web tienen un fichero o un modo de configuración que definen su comportamiento: en qué puerto escuchan, qué ficheros sirven, que plug-ins han de cargar, etc. Tenemos que modificar la configuración del servidor web para que cargue el redirector `mod_jk`.

3.3. Paso 1: Instalación de mod_jk

3.3.1. Windows

En primer lugar hay que obtener el binario del conector. La versión actual de `mod_jk` es la 1.2.40. En concreto, accedemos a la página <www.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/windows/> y descargamos el binario para la versión del servidor que dispongamos. Por ejemplo, el fichero `tomcat-connectors-1.2.39-windows-i386-httpd-2.4.x.zip` (<www.apache.org/dist/tomcat/tomcat-connectors/jk/binaries/windows/tomcat-connectors-1.2.39-windows-i386-httpd-2.4.x.zip>) sirve para Apache 2.4.

Descargamos el fichero en un directorio temporal, los descomprimimos y copiamos el fichero descomprimido, `mod_jk.so` en el directorio `modules` de Apache.

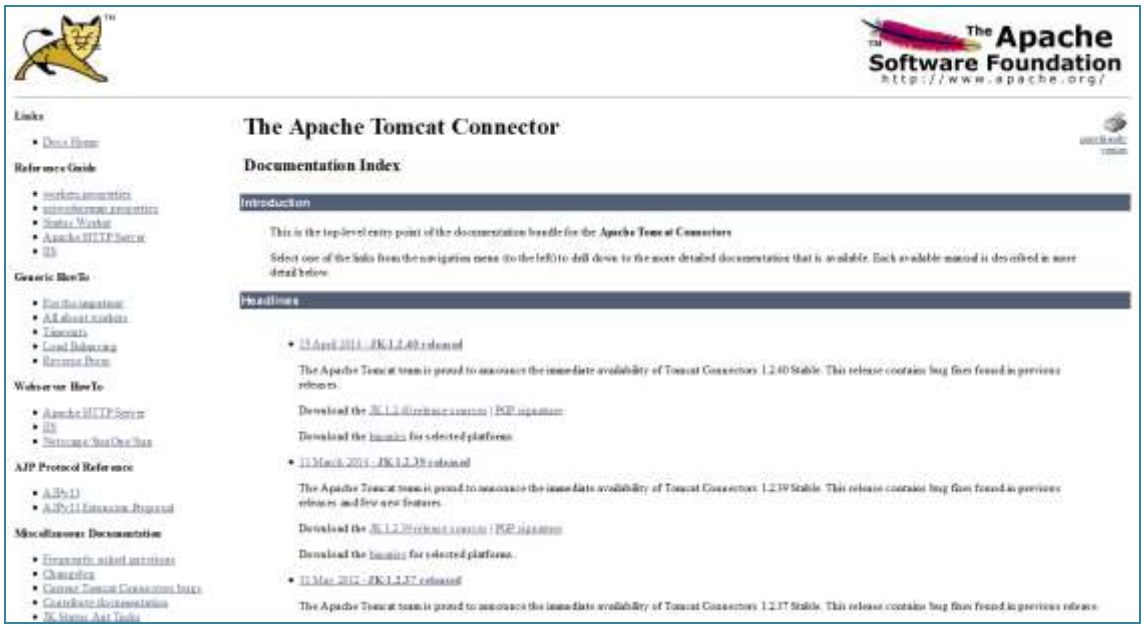

Figura 4.3.1: Página de descarga de mod_jk

3.3.2. Unix

La instalación en sistemas Unix sería mediante paquete binario de la distribución que se disponga de Unix, o bien recompilando a partir de las fuentes.

Por ejemplo, en Ubuntu sería simplemente

```
apt-get install libapache2-mod-jk
```

Si se dispone de las herramientas necesarias, se pueden bajar el código fuente de <tomcat.apache.org/tomcat/tomcat-connectors/> y compilarlo. Este proceso queda fuera del alcance de este proyecto, pero en resumen sería:

- Bajar los fuentes del conector de versión actual de <tomcat.apache.org/tomcat/tomcat-connectors/>. Descargar el fichero (tendrá un nombre como `tomcat-connectors-1.2-40-src.tar.gz`) en un directorio temporal del sistema donde se va a realizar la compilación.
- Comprobar que se tienen las herramientas necesarias para realizar el proceso: compilador ANSI C (recomendado GNU C, como en el caso de Apache) y además libtool y autoconf.
- Una vez instaladas todas las herramientas, descomprimir el fichero descargado en un directorio temporal y configurar:

```
# tar xvzf tomcat-connectors-jk1.2.15.tar.gz
# cd jk/native
./configure –with-apxs=/usr/local/apache2/bin/apxs
# make
# make install
```

Figura 4.3.2: Comandos para descomprimir el fichero y configurar

En este caso se asume que Apache está instalado en `/usr/local/apache2`. El instalador copia en el directorio `/usr/local/apache2/modules` el fichero `mod_jk.so`.

3.4. Paso 2: Configuración de Tomcat

3.4.1. Workers

Un Tomcat worker es una instancia Tomcat que está esperando para ejecutar servlets o JSPs a petición de un servidor web. Por ejemplo, podemos tener un servidor web como Apache redirigiendo las peticiones de servlets a un proceso Tomcat (el worker) que corre bajo él.

Se pueden configurar muchos workers de Tomcat para servir servlets a demanda de un servidor web. Y esto, puede ser por muchas razones, como por ejemplo:

- Queremos que diferentes contextos sean servidos por distintos workers de Tomcat para proporcionar un entorno de desarrollo, donde todos los desarrolladores comparten el mismo servidor web, pero cada uno tiene un worker Tomcat para él.
- Queremos que diferentes servidores virtuales sean servidos por distintos procesos Tomcat para proporcionar una clara separación entre sitios que pertenecen a distintas compañías.

- Queremos proporcionar balanceo de carga, lo que implica ejecutar varios workers de Tomcat cada uno en su propia máquina y distribuir las peticiones entre ellos.

3.4.2. Cómo definir Workers

Toda la información se encuentra disponible en <tomcat.apache.org/connectors-doc/generic_howto/workers.html>.

Los workers de Tomcat se definen en un fichero de propiedades que se llama **`workers.properties`** y que se ubica en el directorio `conf`. Cada directiva definida en el fichero `worker.properties` consiste en tres palabras separadas por puntos. La primera palabra es siempre "worker". La segunda palabra es el nombre del worker, que puede ser cualquier nombre arbitrario, y que coincidirá con el nombre del jvmRoute definido en el fichero `server.xml` de configuración de Tomcat (el nombre del worker puede contener sólo caracteres alfanuméricos). La tercera palabra es el nombre de la directiva.

La definición de los workers se realiza mediante la siguiente directiva:

```
worker.list=<lista de nombres de workers separados por comas>
```

Al arrancar, el plugin de web (web server plugin) inicializará los workers cuyos nombres aparezcan en la propiedad `worker.list` y estos serán también los únicos a los que se puedan asignar peticiones.

Otra directiva de definición es la **`worker.maintain`**, que define el timeout del pool de conexiones del worker en segundos. Por defecto vale 60 segundos.

3.4.3. Tipos de Workers

Cada uno de los workers listados ha de tener también algunos datos que proporcionen información adicional. Esta información ha de incluir el tipo del worker. La siguiente tabla presenta los tipos de workers que existen hasta el momento.

Tipo	Descripción
ajp13	Este worker sabe cómo redirigir peticiones a los workers Tomcat out-of-process usando el protocolo AJPv13. Es el tipo por defecto.
lb	Worker de balanceo de carga, sabe cómo proporcionar balanceado round-robin con determinado nivel de tolerancia a fallos.
status	El worker status es un tipo especial que no comunica directamente con Tomcat, sino que es responsable de la gestión del balanceador de carga.

Para definir los workers de cada tipo hay que respetar este formato:

```
worker.<worker name>.type=<tipo del worker>
```

Donde <worker name> es el nombre que se asignó al worker en la propiedad de más arriba y "type" es uno de los tipos que aparecen en la tabla anterior. Los nombres de los workers no deben contener espacios (cualquier buena convención de nombres debe seguir las reglas de las variables Java).

```
  # Define un worker llamado "remote" que usa ajpv13
worker.remote.type=ajp13

  # Define un worker llamado "loadbalancer" que balancea la
carga entre varios procesos Tomcat
  worker.loadbalancer.type=lb
```

3.4.4. Cómo configurar las propiedades del Worker

Las propiedades de los workers se definen mediante las directivas de conexión, con este formato:

```
<nombre del worker>.<propiedad>=<valor de la propiedad>
```

Cada worker tiene un juego de propiedades diferente, según su tipo.

3.4.4.1 WORKERS AJP13

Los workers del tipo ajp13 dirigen request a workers Tomcat out-of-process usando el protocolo AJPv13 sobre sockets TCP/IP.

AJPv13 es un protocolo binario y trata de comprimir algunos de los datos de la request codificando las cadenas más usadas como enteros. Además reutiliza los sockets abiertos y los deja abiertos para peticiones futuras (esto es importante cuando se tiene un Firewall entre el servidor Web y el Tomcat). Además tiene un tratamiento especial para la información SSL, para que el contenedor pueda implementar métodos de SSL como `isSecure()`.

Algunas directivas de los workers ajp13:

- **`host`**: el host donde el worker Tomcat está escuchando las peticiones ajp13. Por defecto es localhost.
- **`port`**: el puerto donde escuchará la instancia Tomcat. Para los workers ajp13, el puerto por defecto es 8009.

- **socket_timeout**: timeout en segundos del socket del canal de comunicaciones usado entre JK y el host remoto. Si el host remoto no responde antes de que se cumpla el timeout, se produce un error. Por defecto el valor es 0 (no hay timeout).
- **socket_keepalive**: esta directiva se usa cuando hay un cortafuegos entre el servidor web y Tomcat que cierra las conexiones inactivas. Este flag le dice al sistema operativo que envíe cada cierto tiempo (en función del sistema operativo) mensajes de `KEEP_ALIVE`, para evitar que el cortafuegos cierre la conexión. Para activarlo, hay que poner la directiva a un valor mayor de 0 (por defecto, es false).
- **recycle_timeout**: el número de segundos de inactividad que esperará el servidor web para cortar una conexión, tras haber sido asignada la conexión para servir una petición. Por defecto vale 0 (no recicla).
- **retries**: el número de reintentos de conexión que realizará un worker tras recibir un error por parte del Tomcat remoto. Por defecto, el valor es 3.
- **cachesize**: define el número de conexiones que pueden ser mantenidas como un pool de conexiones. Limitará el número de conexiones que cada proceso hijo del web server puede mantener. Sólo se usa en servidores web multithreaded (apache 2.x, IIS, Netscape) y debería reflejar el número de threads por proceso hijo. Por defecto, es 1 (para apache 1.3 o configuraciones prefork de Apache 2.x, este valor no puede ser superior a 1).
- **cache_timeout**: esta directiva se debería usar junto con `cachesize` para expresar cuánto tiempo debería mantener JK un socket abierto en cache antes de cerrarlo.
- **lbfactor**: esta propiedad se usa cuando se trabaja con un worker de balanceo de carga; es el factor de balanceo para este worker. Indica cuánto se espera que este worker trabaje con relación a otros workers. Por ejemplo, si un worker tiene un `lbfactor` 5 veces mayor que otro worker lb, significa que recibirá 5 veces más peticiones que el otro. Veremos más en la sección sobre los workers lb.

```
# "worker2" hablará con un Tomcat en la máquina www2.x.com, puerto 8009, usando un factor de carga de 3
worker.worker2.host=www2.x.com
worker.worker2.port=8009
worker.worker2.lbfactor=3
# "worker2" utilize conexiones que no durarán más de 10 minutos en el pool
worker.worker2.connection_pool_timeout=600
# "worker2" solicita envío de KEEP-ALIVE en la conexión
worker.worker2.socket_keepalive=1
```

3.4.4.2 PROPIEDADES DE LOS WORKERS LB

Los workers de balanceo de carga no se comunican realmente con otros workers Tomcat. Es el responsable del control de varios workers "reales". Este control incluye:

- Instanciar los workers en el servidor web.
- Usando el factor de carga del worker, ejecutar un balanceo round-robin ponderado, en el que un factor `lbfactor` alto implica una máquina más potente (que va a manejar más requests).
- Controlar que las requests que pertenecen a la misma sesión se ejecuten en el mismo worker Tomcat.
- Identificar los workers fallidos, suspender las requests para ellos y redirigirlas a los demás workers que están en el balanceo.

El resultado general es que los workers controlados por el mismo worker lb resultan balanceados (según su factor `lbfactor` y la sesión actual de usuario) y también tienen la seguridad de que la caída de un solo proceso Tomcat no matará todo el sitio (fall-back). Esta tabla especifica las propiedades que el worker lb puede aceptar:

- **`balance_workers`**: es una lista separada por comas de los workers que el balanceador tendrá que controlar. Estos workers no deben aparecer en la propiedad `worker.list`.
- **`sticky_session`**: especifica si las requests con el identificador de sesión SESSION ID han de ser devueltas al mismo worker. Si `sticky_session` es un entero diferente de cero, o es True, se configurará como `JK_TRUE` y las sesiones serán persistentes (sticky). Hay que poner esta propiedad a `JK_FALSE` cuando Tomcat utiliza un Session Manager que puede hacer persistir los datos de sesión a través de múltiples instancias de Tomcat. Por defecto, `sticky_session` es true.
- **`sticky_session_force`**: especifica si las peticiones con el identificador de sesión `SESSION_ID` para workers que están en estado de error deberían ser rechazadas. Si es True o 1, cuando un worker está en estado de error y recibe una petición con identificador de sesión `SESSION_ID`, el cliente recibirá un error 500.
- **`method`**: especifica qué método usa el balanceador para elegir el mejor worker. Si esta directiva vale R[equest], el balanceador usará el número de peticiones para encontrar el mejor worker. Si está fijada a T[raffic], el balanceador usará el tráfico de red entre JK y Tomcat para encontrar el mejor worker.

```
# The worker balance1 controlará los workers "reales"
worker1 y worker2
  worker.balance1.balance_workers=worker1, worker2
```

Hay más directivas avanzadas (`connect_timeout`, `domain`,…) pero no se van a analizar. Más información se puede encontrar en <tomcat.apache.org/connectors-doc/>.

3.5. Paso 3: Configuración de Apache

Si se hubiera configurado Apache para usar `mod_jserv` (el conector antiguo), hay que eliminar todas las directivas ApJServMount del httpd.conf. Si estábamos incluyendo `tomcat-apache.conf`, o `tomcat.conf` hay que quitarlos también. Las directivas de configuración de `mod_jserv` no son compatibles con `mod_jk`. En una instalación limpia, como es el caso de la de nuestro curso, esto no es necesario.

La información básica de esta sección se recoge en <tomcat.apache.org/connectors-doc/webserver_howto/apache.html>

El primer paso es generar un fichero de configuración del módulo `mod_jk` (`mod_jk.conf`) en el que se definen los puntos de montaje de las distintas webapp de Tomcat y el worker al que se asignarán (ver documentación). A partir de Tomcat versión 3, se puede generar el fichero `mod_jk.conf` de forma automática.

Tomcat es flexible al configurar la generación del fichero de configuración automática para Apache, aunque a costa de cierta complejidad. Cada vez que Tomcat se arranca, escribirá el fichero de configuración a `$TOMCAT_HOME_HOME/conf/auto/mod_jk.conf`, siendo `TOMCAT_HOME` el directorio de instalación de Tomcat.

Por tanto, bastará incluirlo en httpd.conf de Apache para que esté siempre al día:

```
#Añadir al final de httpd.conf
Include $TOMCAT_HOME/conf/auto/mod_jk.conf
```

Para que Tomcat genere las directivas globales para Apache, se define un `Listener` dentro del Server, en el fichero `server.xml`, dentro de la definición del <Engine>:

```
<Engine ...>
    ...
    <Listener className="org.apache.jk.config.ApacheConfig"
modJk="$APACHE_HOME/modules/ /mod_jk.so" />
    ...
</Engine>
```

siendo `$APACHE_HOME` el directorio de instalación de Apache (`ServerRoot`).

Cada vez que se reinicie Tomcat, generará un fichero `mod_jk.conf` en `$TOMCAT_HOME/conf/auto`. Como este fichero se cargará desde el fichero `httpd.conf`, se deberá iniciar Tomcat primero para que el fichero esté actualizado cuando se inicie Apache. Si la instalación es muy estática y no se añaden contextos nuevos, no tiene sentido mantener la generación del fichero `mod_jk.conf` permanentemente.

No se podrá utilizar este método de configuración cuando:

- Tomcat no está en el mismo servidor que Apache. Esto ocurre cuando, por ejemplo, Apache es el frontal de una granja de servidores Tomcat.
- Cuando cada servidor Tomcat tiene una configuración diferente (por ejemplo, en una empresa proveedora de servicios ISP)
- Cuando el webmaster quiera realizar un afinamiento de la configuración por cuestiones de rendimiento.

En cualquiera de estos casos, habrá que hacer una configuración manual. Hay que rearrancar Tomcat y Apache cada vez que se añada un contexto nuevo.

3.5.1. Configuración básica de Apache

Primero hay que decirle a Apache que cargue el módulo `mod_jk`, para lo que se usará la directiva `LoadModule`:

```
LoadModule    jk_module  modules/mod_jk.so
```

Hay que informar a `mod_jk` de dónde están los ficheros `workers.properties`. Va en la directiva `JkWorkersFile` del fichero `mod_jk.conf` (las directivas de `mod_jk` se explican en el punto siguiente):

```
JkWorkersFile c:\Curso\Apache2.4\conf\workers.properties
```

(o su directorio equivalente en Unix).

Tenemos que especificar un directorio donde colocar el fichero de log y el nivel de traza que se irá creando (directivas `JkLogFile` y `JkLogLevel` de `mod_jk`):

```
JkLogFile     C:\Curso\Apache2.4\logs\mod_jk.log
JkLogLevel    info
```

(o el equivalente en Unix)

La directiva `JkLogStampFormat` configura el formato de fecha hora para el fichero de log, usando `strftime()`. Por defecto, "[%a %b %d %H:%M:%S %Y]":

```
JkLogStampFormat "[%a %b %d %H:%M:%S %Y] "
```

Crear directivas `JkMount` para asignar URLs específicas a Tomcat. En general: `JkMount PREFIJO_URL NOMBRE_WORKER`. Pueden ponerse estas directivas en el nivel raíz o dentro de secciones <`VirtualHost`> del fichero `httpd.conf`.

```
# Enviar el contexto /examples al worker worker1
JkMount  /examples/servlet/* worker1
# Enviar los JSPs  del contexto /examples al worker worker1
JkMount  /examples/*.jsp worker1
```

3.5.2. Directivas más importantes

Para configurar el conector JK es necesario fijar unas cuantas directivas en el fichero de configuración de Apache `httpd.conf`, tal y como se ha visto anteriormente.

Las directivas más importantes son las siguientes:

- **`JkWorkersFile`**: especifica la ubicación donde `mod_jk` debería encontrar las definiciones de los workers. Por ejemplo:

```
JkWorkersFile
C:\curso\Apache2.4\conf\workers.properties
```

- **`JkLogFile`**: especifica la ubicación donde mod_jk deja los registros de acceso. Por ejemplo:

```
JkLogFile  C:\curso\Apache2.4\logs\mod_jk.log
```

- **`JkLogLevel`**: fija el nivel de registro. Las posibilidades son:
  - Info: el fichero de log contendrá la actividad estándar de `mod_jk` (es el defecto)
  - Warn: el fichero contendrá informes de errores no fatales
  - Error: el fichero contendrá también informes de errores
  - Debug: el fichero de log contendrá información de toda la actividad de `mod_jk`
  - Trace: el fichero de log contendrá toda la información de traza de la actividad de `mod_jk`

  Por ejemplo:

```
JkLogLevel info
```

- **`JkLogStampFormat`**: configurará el formato de fecha/hora que se utilizará en el fichero de log de `mod_jk`. El defecto es:

```
JkLogStampFormat  "[%a %b %d %H :%M:%S %Y]"
```

- **`JkRequestLogFormat`**: configurará el formato del registro de las peticiones individuales. Para habilitar el registro de las peticiones para un host virtual hay que añadir una directiva `JkRequestLog Format`. Por ejemplo:

```
JkRequestLogFormat          "%w %V %Y"
```

- **JkMount**: la directiva `JkMount` asigna URLs específicas a Tomcat. En general la estructura de una directiva `JkMount` es:

```
JkMount        [Prefijo URL][Nombre del Worker]
```

Por ejemplo:

```
# Envía todas las peticiones que acaban en .jsp a worker1
JkMount  /*.jsp  worker1
# Envía todas las peticiones de la URL /servlet a worker1
JkMount  /*/servlet/ worker1
# Envía todas las peticiones en /otroworker a worker2
JkMount  /otroworker/* worker2
```

La directiva `JkMount` se puede usar a nivel global o a nivel de <`VirtualHost`>

- **JkUnMount:** actúa de forma contraria a `JkMount` y bloquea el acceso a una determinada URL. El objetivo es filtrar un determinado tipo de contenido de un contexto montado. El ejemplo siguiente monta el contexto `/servlet/*`, pero todos los ficheros `*.gif` que pertenecen a dicho contexto no es servido.

```
JkMount /servlet/* worker1
JkUnMount /servlet/*.gif worker1
```

La directiva `JkUnMount` tiene prevalencia sobre la directiva `JkMount`. Por ejemplo, el siguiente grupo de directivas bloquea todos los ficheros `.gif`:

- **JkAutoAlias**: Esta directiva mapea automáticamente los directorios de contexto de las aplicaciones web en el espacio de documentos de Apache. Esto posibilita que Apache pueda servir un contexto estático mientras Tomcat sirve un contexto dinámico, sin tener que usar una directiva `Alias` por cada directorio de aplicación dentro del directorio de aplicaciones de Tomcat. El ejemplo siguiente muestra como servir un contexto dinámico por Tomcat y estático usando Apache. El directorio de aplicaciones web (`webapps`) debe ser accesible por Apache:

```
JkAutoAlias /opt/tomtact/webapps
# Monta 'servlets-examples, cuya ubicación es /opt/tomtact/webapps/servlets-examples. ajp13w es un worker definido en workers.properties
JkMount /servlets-examples/* ajp13w
# Desmonta contenido estático de la webapp servlets-examples, que será servido por Apache
JkUnMount /servlets-examples/*.gif ajp13w
JkUnMount /servlets-examples/*.jpg ajp13w
```

- **JkWorkerProperty:** nueva directiva a partir de la versión JK 1.2.7. Permite fijar directivas que normalmente se fijan en el fichero `worker.properties`. La directiva a fijar se pasa como parámetro. Por ejemplo:

```
JkWorkerProperty worker.list=worker2,worker3
```

- **JkMountFile:** se usa para modificar de forma dinámica puntos de montaje en tiempo de ejecución. Cuando el fichero de montaje es modificado, JK vuelve a cargar su contenido en un intervalo de 60 segundos.

```
JkMountFile conf/ficheromount.properties
```

3.6. Balanceo de carga mediante mod_jk

Para construir aplicaciones web rápidas y escalables, podemos configurar Apache para que delegue el servicio de peticiones para JSPs y servlets a múltiples servidores Tomcat, conectados con el módulo `mod_jk` que ejecutará balanceo de carga con afinidad de sesión, el sistema conocido como "sticky session".

Cuando un cliente (browser) solicita una página JSP por primera vez, el balanceador redirige esa petición recibida por Apache a uno de los dos servidores Tomcat; las siguientes peticiones que lleguen de esa misma sesión del cliente, serán redirigidas automáticamente al mismo servidor Tomcat, de tal forma que los datos de la sesión del cliente se conservan.

3.6.1. Configurar Apache

El grueso de la configuración va en el fichero `workers.properties`.

Hay que tener instaladas dos instancias de Tomcat que pueden estar soportadas por servidores distintos (en cuyo caso pueden escuchar en el mismo puerto) o en el mismo servidor (y entonces deben estar configurados en puertos diferentes). El servidor Apache puede compartir máquina con una de las instancias Tomcat o estar en un servidor diferente.

Los pasos a seguir son:

1. Crear un fichero llamado `workers.properties` en `$APACHE_HOME/conf` con los contenidos siguientes (atención: ajustar las rutas a las correctas en la instalación).

```
# workers.properties
#
# En Windows, las barras a la izquierda:
ps=\
# lista de workers por sus nombres
worker.list=tomcat1, tomcat2, loadbalancer
# -----------------------
# Primer servidor tomcat
# -----------------------
worker.tomcat1.tomcat_home=c:\curso\tomcat1\
worker.tomcat1.port=8009
worker.tomcat1.host=servidor1
worker.tomcat1.type=ajp13
#
#  Especificar  el  factor  de  balanceo  de  carga  para  el
balanceador
# Nota:
# ----> lbfactor ha de ser > 0
# ----> cuanto más bajo sea lbfactor, menos trabajo hará el worker
worker.tomcat1.lbfactor=1
# -----------------------
# Segundo servidor tomcat
# -----------------------
worker.tomcat2.tomcat_home=c:\curso\tomcat2\
worker.tomcat2.port=8009
worker.tomcat2.host=servidor2
worker.tomcat2.type=ajp13
#
#  Especificar  el  factor  de  balanceo  de  carga  para  el balanceador
# Nota:
# ----> lbfactor ha de ser > 0
#  ----> cuanto más bajo sea lbfactor, menos trabajo hará el worker
worker.tomcat2.lbfactor=1
# -----------------------
# Load Balancer worker
# -----------------------
#
# El worker loadbalancer es de tipo lb, y ejecuta balanceo de carga
# con ponderación round-robin mediante sticky sessions.
# Nota:
#  ----> Si un worker se cae, el balanceador comprobará su estado de tanto
# en tanto. Hasta que se levante de nuevo, todo el trabajo será redirigido
# al otro worker
worker.loadbalancer.type=lb
worker.loadbalancer.balanced_workers=tomcat1, tomcat2
```

Figura 4.3.3: Configuración Apache - Paso 1

2. Hacemos una copia del fichero httpd.conf en el directorio `$APACHE_HOME/conf`. Modificamos el fichero httpd.conf de apache para que cargue el módulo `mod_jk` y el fichero de configuración `mod_jk.conf`, y para definir las directivas de `mod_jk`.

```
LoadModule jk_module modules/mod_jk.so
#
# Conector mod_jk
Include "C:\Curso\Tomcat\conf\auto\mod_jk.conf"
#
JkWorkersFile conf\workers.properties
JkLogFile     logs\mod_jk.log
JkLogLevel    info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"
```

Figura 4.3.4: Configuración Apache - Paso 2

3. Reiniciar Apache.

3.6.2. Modificar conf/server.xml en cada instancia

Hacemos una copia de seguridad del fichero de configuración de Tomcat de la primera instancia, `server.xml`, situado en el directorio conf. Buscamos la entrada:

```
<Engine name="Catalina" defaultHost="localhost">
```

y la sustituimos por:

```
<Engine name="Catalina" defaultHost="localhost"
jvmRoute="tomcat1">
```

Lo mismo hacemos en la instancia Tomcat2, cambiando `jvmRoute="tomcat1"` por `jvmRoute="tomcat2"`. En la primera instancia, además, quitamos el listener que pusimos para generar el fichero `mod_jk.conf`, para que no lo vuelva a generar.

3.6.3. Modificar el fichero mod_jk.auto

Las llamadas a las URLs que se definían en ese fichero con la directiva `JkMount` se remitían al worker ajp13 por defecto. Ahora se trata de remitirlas al worker `loadbalancer`, que hemos definido en el `workers.properties`, y que se encargará de desviarlo alternativamente a Tomcat1 y Tomcat2.

Además, introducimos una nueva directiva `JkMount` para que todas las peticiones a `.jsp` del directorio raíz también se remitan al mismo worker. Editamos entonces el fichero `conf/auto/mod_jk.conf`, en la primera instancia de Tomcat, y lo dejamos como sigue:

```
<IfModule !mod_jk.c>
      LoadModule jk_module "modules/mod_jk.so"
</IfModule>
<VirtualHost localhost>
      ServerName localhost

      JkMount /*.jsp loadbalancer

      JkMount /admin loadbalancer
      JkMount /admin/* loadbalancer
      JkMount /webdav loadbalancer
      JkMount /webdav/* loadbalancer
      JkMount /servlets-examples loadbalancer
      JkMount /servlets-examples/* loadbalancer
      JkMount /jsp-examples loadbalancer
      JkMount /jsp-examples/* loadbalancer
      JkMount /balancer loadbalancer
      JkMount /balancer/* loadbalancer
      JkMount /host-manager loadbalancer
      JkMount /host-manager/* loadbalancer
      JkMount /tomcat-docs loadbalancer
      JkMount /tomcat-docs/* loadbalancer
      JkMount /manager loadbalancer
      JkMount /manager/* loadbalancer
</VirtualHost>
```

Figura 4.3.5: Fichero mod_jk.conf