

Instalar nuevas aplicaciones en Tomcat.

Fecha de creación: 01.08.2001

Revisión 1.0.3 (05.03.2003)

Alberto Molpeceres (al AT javahispano DOT org)



<http://www.javaHispano.org>

Copyright (c) 2002, Alberto Molpeceres. Este documento puede ser distribuido solo bajo los términos y condiciones de la licencia de Documentación de javaHispano v1.0 o posterior (la última versión se encuentra en <http://www.javahispano.org/licencias/>).

Instalar nuevas aplicaciones en Tomcat.

Supongo que si estas metido un poco en este de Java es difícil que, al menos alguna vez, no hayas oído hablar de Tomcat [1]. Tomcat es un aclamado contenedor de Servlets y páginas JSP, totalmente open source y bastante potente. Seguro que has oído muchas cosas buenas de él, pero seguro que alguna vez también has oído que su documentación es un poco escasa, y si ha eso añadimos las pocas ganas que tenemos normalmente de leer la documentación, pues sabrás que instalar una nueva aplicación, al menos hacerlo bien, tampoco lo hacen muchos. Yo tampoco soy un experto, lo he hecho varias veces pero siempre limitándome a hacer lo mínimo para que funcione. Por alguna razón, ahora me he decidido a explicar como se hace, dando toda la información que pueda, pero perdonadme si no doy un método perfecto y me dejo varias cosas.

Existen tres formas de instalar una aplicación en Tomcat:

- » *Instalar un archivo WAR (Web ARchive) en la jerarquía de ficheros de Tomcat.*
- » *Instalar la aplicación descomprimida en la jerarquia de ficheros de Tomcat.*
- » *Instalar la aplicación donde queramos y manipular el fichero TOMCAT_HOME\conf\server.xml*

1. Formas de instalar aplicaciones en Tomcat.

Instalar un archivo WAR.

Esta es la manera más sencilla de instalar una aplicación para tomcat. Un fichero WAR (Web Archive) no es más que un fichero comprimido (al igual que un JAR) que contiene todos los archivos necesarios para la aplicación web. Este fichero lo puedes generar tu mismo, pero es más fácil si dejas que lo cree alguna de las herramientas que existen para ello, pertenecientes a J2EE [2].

Lo único que hay que hacer es copiar este fichero WAR al directorio TOMCAT_HOME\webapps. Luego será Tomcat el que se encargará de descomprimir el archivo cuando se arranque el servidor. Normalmente esto solo se hace en el momento del despliegue final, cuando quieres instalar la aplicación y hacerla productiva.

Instalar la aplicación descomprimida.

Este método es posiblemente el más usado, al menos en tiempo de desarrollo. Lo que hay que hacer para instalar una aplicación web según este método es copiar todos los ficheros de nuestra aplicación con una estructura de directorios dada, de forma que Tomcat los encuentre y los entienda. Esta estructura de directorios viene indicada por la especificación del API J2EE.

Estructura de directorios.

Empezamos creando un subdirectorio en la carpeta `TOMCAT_HOME\webapps` con el nombre de nuestra aplicación, por ejemplo `mi_aplicacion`. Este directorio será la raíz de nuestra aplicación web. En él es donde se pueden colocar los ficheros HTML, JSP, de imagenes, etc, que componen nuestro sitio, como se hace normalmente, permitiendose, por supuesto, jerarquias de directorios para una mejor organización. A esta carpeta será a la que se vaya cuando el cliente haga una petición a nuestra aplicación por medio de la URL `http://mi_servidor[.PUERTO]/mi_aplicacion/`.

Otros de los directorios que tenemos que crear para seguir la especificacion de J2EE y de Servlets 2.2 (y posteriores), son, dentro de nuestro directorio raíz, el que hemos llamado `mi_aplicacion`, uno llamado `WEB-INF` (así exactamente, **todo en mayúsculas**), y dentro de este otro llamado `classes`, en el que colocaremos nuestras clases compiladas (`.class`). En este subdirectorio `classes` tendremos que poner las clases de nuestros servlets así como las clases que ellos utilicen, es decir, las clases de las que dependen. Estas clases no tienen que estar combinadas en un fichero JAR (estos se colocan en otro sitio) y tienen que respetar la estructura de subdirectorios que indiquen sus paquetes, como ocurre siempre en Java, esto no es nuevo.

Dentro de la carpeta `WEB-INF` todavía podemos crear otro subdirectorio estandar, si nos hace falta. Este subdirectorio se llama `lib`, y es el encargado de almacenar los ficheros JAR especificos de nuestra aplicación. Esto pueden ser cualquier tipo de clases empaquetadas, nuestras o de terceros, como por ejemplo un parser XML, o drivers JDBC.

Para acabar con este directorio especial `WEB-INF`, nos falta un fichero (aunque no es obligatorio), `web.xml`, descriptor de despliegue de la aplicación web, que en cristiano, que es lo que hablamos nosotros, significa que es el fichero donde se describe nuestra aplicación web, es decir, donde decimos que servlets la componen, que parámetros de inicialización tienen estos, restricciones de seguridad de la aplicación, etc.

El fichero web.xml.

Podéis encontrar toda la información detallada de este fichero en la especificación del API Servlets 2.2 (y en las posteriores, obviamente), pero aquí teneis un ejemplo (es una traducción resumida del ejemplo que viene con la documentación de Tomcat), con todo lo necesario para nuestra aplicación.

```
<!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>

<!-- Descripción de la aplicación (opcional) -->

<display-name>Mi aplicacion</display-name>
<description>
  Esta es la primera aplicación que escribo y despliego
  en Tomcat.
  Soy alberto, y mi direccion de correo es
  al at javahispano.com
</description>

<!-- Parametros de inicializacion del contexto de
  la aplicación.
  Estos valores son los que devuelven en los servlets
  y JSP las funciones del estilo:

      String value =
```

Instalar nuevas aplicaciones en Tomcat.

```
        getServletContext().getInitParameter("name");

        donde "name" corresponde al parametro .

        Son opcionales
-->

<context-param>
  <param-name>webmaster</param-name>
  <param-value>webmaster@javahispano.com</param-value>
  <description>
    Dirección del webmaster
  </description>
</context-param>

<!-- Aquí vienen las declaraciones de los servlets
      que intervienen en nuestra aplicación, con sus
      parametros iniciales.

      Llamaremos a nuestros servlets de la forma:

      http://www.miservidor.com:8080/{context-path}/servlet/{classname}

      ejemplo:

      http://www.javahispano.com/sitioweb/servlet/com.javahispano.web.HolaMundo

      aunque esto hace difícil la portabilidad y las
      referencias relativas. Es recomendable usar
      la etiqueta que viene después.

      Puede hacer cualquier numero de servlets, incluso 0.

      Por supuesto no son necesarios todos los campos.
-->

<servlet>
  <servlet-name>HolaMundo</servlet-name>
  <description>
    Este es mi primer servlet, y por tanto sólo dice el
    -obligatorio- "Hola Mundo!".
    Tiene un parámetro que le dice en que color escribir.
  </description>
  <servlet-class>com.javahispano.web.HolaMundo</servlet-class>
  <init-param>
  <!-- este parámetro se carga de la forma
        String value =
            getServletConfig().getInitParameter("color");
-->
    <param-name>color</param-name>
    <param-value>azul</param-value>
  </init-param>
  <!-- Cargar este servlet al arrancar el servidor -->
  <load-on-startup>5</load-on-startup>
</servlet>

<!-- Mapeos definidos que usará el contenedor de servlets para
      traducir a una petición URI (relativa al contexto) a un
      servlet en particular. En el siguiente ejemplo, el URI:

          http://www.miservidor.com:8080/{contextpath}/grafico

      se redirigira al servlet "grafico", y el URI:

      http://www.miservidor.com:8080/{contextpath}/guardarCliente.bd

      sera redirigido al servlet "controlador".
```

```
        Se pueden definir 0 o mas mapeos, e incluso varios para
        el mismo servlet.
-->

<servlet-mapping>
  <servlet-name>controlador</servlet-name>
  <url-pattern>*.bd</url-pattern>
</servlet-mapping>

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

<!-- Define el tiempo (en minutos) durante el que nuestra
      aplicación mantendrá las sesiones.
      Se puede cambiar desde el código con
      HttpSession.getMaxInactiveInterval(). -->

<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>

</web-app>
```

Ampliación del fichero web.xml

Esta es una versión muy limitada del fichero web.xml. Según la especificación del API se pueden detallar muchos más aspectos de nuestra aplicación, pero son eso, aspectos configurables de nuestra aplicación, no necesarios para la ejecución de Tomcat. Aunque una explicación detallada de estos elementos sería más adecuada para un artículo sobre la creación de aplicaciones J2EE (quizás sería interesante, ¿algún interesado en escribirlo?), detallaré algunos de los que pueden resultar más interesantes.

Uno de los elementos adicionales para el descriptor web.xml es `<error-page>`, que indica que página se mostrará cada vez que se produzca un error, que en Java se llaman excepciones. El siguiente elemento determinará que cada vez que se produzca una excepción `Exception` (o una de sus descendientes) se muestra la página de error.

```
<error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/error.jsp</location>
</error-page>
```

Otro elemento útil puede ser `<taglib>`, que permite mapear un biblioteca de etiquetas JSP a una dirección relativa o URI. Por ejemplo, podemos mapear nuestra biblioteca `etiquetas_utilidad_v1_3.tld` a `utilidades` de la siguiente forma:

```
<taglib>
  <taglib-uri>
    /utilidades
  </taglib-uri>
  <taglib-location>
    /WEB-INF/tlds/etiquetas_utilidad_v1_3.tld
  </taglib-uri>
```

```
</taglib>
```

Ahora podemos usar dicha librería de forma más sencilla:

```
<%@ taglib uri=../utilidades_ prefix=_util_ %>
```

También se pueden declarar algunos ficheros o direcciones como privados, de forma que cada vez que se quiera acceder a ellos nos aparezca automáticamente un petición de autenticación, normalmente un formulario para que introduzcamos nombre y contraseña. Esto se hace por medio de los elementos `<security-constraint>` y `<login-config>` (y subelementos), pero bueno, esto solo lo cuento para mostraros un poco las posibilidades que hay (actualmente ya está escrito ese artículo, llamado Autenticación controlada por el contenedor en Tomcat [3]), porque hablar de usuarios, roles y restricciones de seguridad se escapa de la idea de este artículo. Ya sabéis donde hay más información, en las especificaciones de la plataforma J2EE.

Manipular el fichero TOMCAT_HOME\conf\server.xml.

Este último modo de instalar un aplicación en Tomcat, es básicamente igual que el anterior, en el sentido que tenemos que seguir la estructura de directorios que se nos marca y demás. La diferencia está en dónde colocamos los ficheros de nuestra aplicación. Podemos decidir colocar nuestra aplicación fuera de la jerarquía de directorios de Tomcat (por ejemplo en la de Apache), pero entonces, para que este encuentre nuestra aplicación, le tendremos que decir dónde está, y para ellos tenemos que manipular el fichero TOMCAT_HOME\conf\server.xml.

Este proceso de manipulación del fichero server.xml también se puede (debe) usar cuando queremos que nuestra aplicación tenga algunos valores en su contexto que no son lo que vienen por defecto, por ejemplo, para hacer que Tomcat recargue las clases automáticamente cada vez que cambiamos algo en nuestros servlets o clases auxiliares.

De lo que se trata es de añadir un nuevo elemento `<context>` a este fichero XML. Para ello tenemos que editarlo, colocarnos al final de dicho fichero, dentro de un servidor `<host>` justo antes de `</host>` (esta ubicación no es exactamente lo mismo en las distintas versiones de Tomcat, y también está en función de si tenemos configurados servidores virtuales), y añadir nuestra entrada. Por ejemplo:

```
<Context
  path="/mi_aplicacion"
  docBase="/usuario/aplic"
  crossContext="true"
  debug="0"
  reloadable="true"
  trusted="false"
/>
```

Los atributos que acepta este elemento son:

» `path`: este es el nombre por el que se llegará a nuestra aplicación. En nuestro caso `http://miservidor.com/mi_aplicacion`. Es obligatorio ponerlo.

- » `docBase`: esta es la dirección del directorio raíz de nuestra aplicación. Puedes ser un path relativo a `TOMCAT_HOME` (`../web`) o absoluto (`c:\web` en Windows, o `/web` en *nix). Este atributo también es obligatorio.
- » `debug`: este atributo opcional le indica a Tomcat cuanta información de depuración debe dar. Admite valores entre 0 y 9, y cuanto más alto sea mayor información dará. Por defecto es 0 (información mínima).
- » `reloadable`: Este atributo opcional y booleano le indica a Tomcat si tiene que comprobar las clases compiladas (`.class`) en busca de cambios. Por defecto vale `false`, lo que significa que Tomcat no comprueba los cambios en las clases compiladas, por lo que sigue trabajando con las versiones antiguas. En tiempo de desarrollo es recomendable ponerlo a `true` para no tener que reiniciar manualmente el servidor cada vez que cambiamos nuestro Servlet o una clase usada por él.
- » `trusted`: Por defecto este atributo opcional es `false`, e indica si nuestra aplicación tiene permiso para acceder a las clases internas de Tomcat. Más o menos, la única aplicación que lo hace es el administrador que viene con Tomcat.

2. Atributos del elemento `Context`

Por supuesto este elemento (`context`) admite subelementos, para definir cosas como los ficheros de log de la aplicación, pero puesto que eso cambia con cada versión de Tomcat, es mejor que os guíeis por la documentación de nuestra versión y por los ejemplos que trae.

Informando a Apache.

Lo que sucede a continuación, es que nosotros no usamos casi nunca llamadas directas a Tomcat, si no que hacemos las llamadas desde nuestro navegador al servidor web Apache, y este le pasa a Tomcat el trabajo de tratar con Servlets y páginas JSP. Para que Apache se de por enterado de que nuestra aplicación existe, y que queremos que la trate Tomcat, tenemos que cambiar aún un fichero. Si utilizamos el módulo **mod_jk** para realizar la conexión entre Apache y Tomcat este fichero es el `mod_jk.conf` situado en el directorio `TOMCAT_HOME\conf` generalmente, si utilizamos **mod_webapp** tendremos que cambiar el final de `APACHE_HOME/conf/httpd.conf`.

Para versiones anteriores, si habeis optado por **mod_jk**, podeis adivinar que fichero concreto mirando la configuración de Apache. Cuando instalasteis Apache y Tomcat y les hicisteis colaborar (en esta web teneis artículos para la version 3.x [4] y 4.x[5]), tuvisteis que incluir un fichero de Tomcat en el fichero `APACHE_HOME\conf\httpd.conf`. Al final de este fichero seguro que escribisteis algo así como (Tomcat 3.x):

```
include c:/tomcat/conf/mod_jk.conf
```

Ese es el nombre del fichero que tendréis que editar. Lo único que hay que hacer para que Apache le pase esas llamadas a nuestra aplicación a Tomcat es añadir los datos de autoconfiguración de nuestra aplicación. Tan sencillo como:

```
#####
```

Instalar nuevas aplicaciones en Tomcat.

```
# Inicio autoconfiguracion del contexto de /mi_aplicacion
#####

#
# Esta línea informa a Apache de la existencia de un
# sitio web fuera de su jerarquia de directorios.
# En Tomcat hacemos algo similar si moviamos nuestra aplicación
#
Alias /mi_aplicacion "c:/tomcat/webapps/mi_aplicacion"
<Directory "c:/tomcat/webapps/mi_aplicacion">
Options Indexes FollowSymLinks
</Directory>

#
# Ahora informamos a Apache de que le mande las peticiones
# de Servlets y JSP a Tomcat.
# Solo le envia estas peticiones, ya que Apache sirve
# mas rapidamente el contenido estático.
# Ademas le decimos que Protocolo usar, si tenemos el
# 13 mejor que el 12 ;- ) (en realidad es 1.2 y 1.3).
# Ver como el "como instalar Tomcat" en nuestra web
# para mas información.
#
JkMount /mi_aplicacion/servlet/* ajp13
JkMount /mi_aplicacion/*.jsp ajp13

#####
# Fin autoconfiguracion del contexto de /mi_aplicacion
#####
```

Con esto sería suficiente, pero podemos añadir otros datos de configuración, sabiendo que son cosas de Apache, nada que ver con Tomcat.

Por ejemplo impediremos que nadie lea nuestra carpeta /WEB-INF, a fin de cuentas eso no es el interfaz de usuario, ¿no?

```
#
# Esto es para todas las plataformas salvo Windows
#
<Location "/mi_aplicacion/WEB-INF/">
    AllowOverride None
    deny from all
</Location>
#
# Y esto es para Windows.
# Location no funciona bien en Windows
#
<Directory "c:/web/tomcat/webapps/mi_aplicacion/WEB-INF/">
    AllowOverride None
    deny from all
</Directory>
```

Si habeis optado por comunciar Apache y Tomcat por medio del módulo **mod_webapp** lo mejor que podeis hacer es consultar el final del artículo dedicado a la instalación de Apache con Tomcat 4 [\[5\]](#), donde viene explicado en detalle el proceso.

Conclusión

Y eso es todo, punto y final, quizás no sea la cosa más sencilla del mundo, pero tampoco es imposible, ¿no?. No os asusteis, después de hacerlo un par de veces le perdereis el miedo.

Este pequeño *como* esta basado en gran medida en la documentación de Tomcat, que quizás alguno de vosotros os hayais atrevido a leer. Pero no se puede considerar una traducción literal, es una simple ampliación detallada de las notas que he ido tomando (creedme que no todo esta en la documentación) a lo largo de de las veces que he tenido que crear y desplegar aplicaciones para Tomcat. También he añadido información procedente de la especificación de J2EE y Servlets. Puesto que no es una traducción literal, para cualquier duda os remito a la documentación original y a los FAQ existentes sobre Tomcat, ya sea en la propia fundación Apache o en jguru [6].

Recursos y referencias

- [1] Sitio de Tomcat, <http://jakarta.apache.org/tomcat/>
- [2] Sitio de Java Enterprise Edition, <http://java.sun.com/j2ee/>
- [3] Autenticación manejada por el contenedor en Tomcat, http://www.javahispano.org/articulos/ver_articulo.jsp?id=24
- [4] Instalación de Apache + Tomcat 3, http://www.javahispano.org/articulos/ver_articulo.jsp?id=18
- [5] Instalación de Apache + Tomcat 4, http://www.javahispano.org/articulos/ver_articulo.jsp?id=47
- [6] jGurú, <http://www.jguru.com>

Acerca del autor

Alberto es ahora mismo desarrollador de aplicaciones en ámbito cliente/servidor para la empresa T-Systems - debis Systemhaus en Munich (Alemania). Cuando no está trabajando o "metiendo caña" al resto de los integrantes de javaHispano intenta pasear con su novia, buscar la desaparecida lógica del idioma alemán o intentar olvidar la pesadilla que es buscar piso en Munich.

Copyright (c) 2002, Alberto Molpeceres. Este documento puede ser distribuido solo bajo los términos y condiciones de la licencia de Documentación de javaHispano v1.0 o posterior (la última versión se encuentra en <http://www.javahispano.org/licencias/>).