Configuración de Tomcat. Configuración de Aplicaciones Web.

Índice

En esta sesión veremos primero cómo configurar Apache Tomcat. Una vez configuradas las características comunes a todas las aplicaciones veremos cómo configurar los aspectos propios de cada aplicación. De éstos hay algunos que son estándares e independientes del servidor, pero hay otros particulares a Tomcat.

1. Configuración de Tomcat

Para comprender mejor la configuración de Tomcat antes hay que ver cuál es su arquitectura, tanto desde el punto de vista lógico como físico.

1.1. Estructura física y lógica de Tomcat

Estructura física

La distribución de Tomcat está dividida en los siguientes directorios:

- `bin`: ejecutables y scripts para arrancar y parar Tomcat.
- `common`: clases y librerías compartidas entre Tomcat y las aplicaciones web. Las clases se deben colocar en `common/classes`, mientras que las librerías en formato JAR se deben poner en `common/lib`.
- `conf`: ficheros de configuración.
- `logs`: directorio donde se guardan por defecto los logs.
- `server`: las clases que componen Tomcat.
- `shared`: clases compartidas por todas las aplicaciones web.
- `webapps`: directorio usado por defecto como raíz donde se colocan todas las aplicaciones web.
- `work` y `temp`: directorios para almacenar información temporal

Los logs del servidor

Además del *log* de accesos, en Tomcat podemos tener otros ficheros de *log* cuya finalidad primordial es registrar eventos del servidor y de las aplicaciones para poder realizar una depuración en caso de que se produzca algún error.

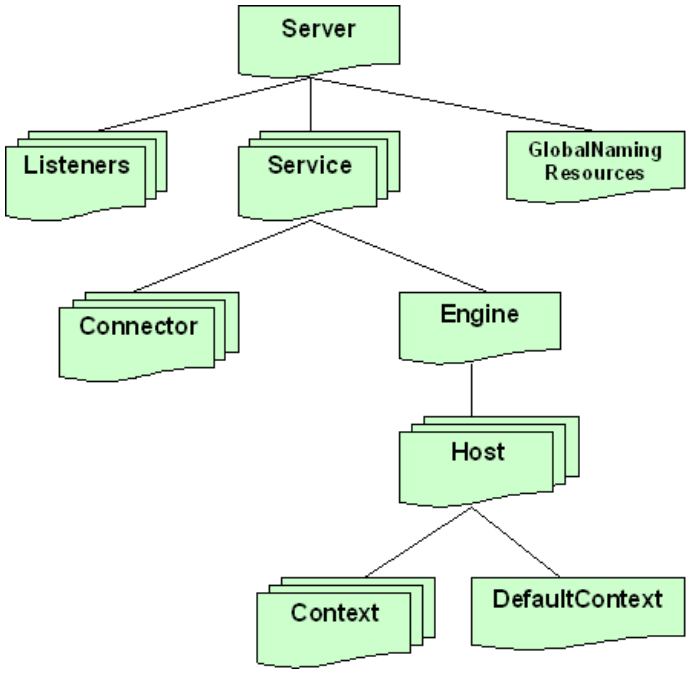
Estos ficheros se encuentran dentro de la carpeta `logs`. Algunos ya vienen por defecto. Dependiendo de la versión de Tomcat, podemos encontrarnos uno o varios de los siguientes:

- `catalina.aaaa-mm-dd.log`: *logger* global definido al nivel del `engine`.
- `localhost.aaaa-mm-dd.log`: *logger* global a todas las aplicaciones del host `localhost`.
- `manager.aaaa-mm-dd.log`: *logger* que utiliza el manager de Tomcat.
- `admin.aaaa-mm-dd.log`: *logger* que utiliza la aplicación de administración.

En estos ficheros podremos encontrar mensajes sobre el funcionamiento general y errores generales del servidor Tomcat.

Estructura lógica: módulos

Tomcat está compuesto por una serie de módulos cuyo comportamiento es altamente configurable. Incluso se pueden cambiar las clases que utiliza Tomcat por clases propias modificando el fichero `server.xml`. La estructura general de dichos módulos se muestra en la siguiente figura.



Cada módulo viene representado por su correspondiente etiqueta en el fichero `server.xml`.

- **Server**: es el propio Tomcat. Solo existe una instancia de este componente, que a su vez contiene a todos los demás elementos y subelementos.
- **Listener**: monitoriza la creación y eliminación de contenedores web
- **GlobalNamingResources:** sirve para definir mapeados de JNDI globales a todas las aplicaciones. Por ejemplo, para definir métodos de conexión a bases de datos.
- **Service**: un objeto de este tipo representa el sistema formado por un conjunto de conectores (`connector`) que reciben las peticiones de los clientes y las pasan a un `engine`, que las procesa. Por defecto viene definido el servicio llamado `Tomcat-Standalone`.
- **Connector**: acepta ciertos tipos de peticiones para pasarlas al `engine`. Por defecto, Tomcat incorpora un conector HTTP/1.1 (sin SSL) por el puerto 8080, y otro para comunicación con otros servidores (como Apache). Para cambiar el puerto por el que Tomcat acepta las peticiones HTTP basta con cambiar el atributo `port` de dicho `connector`.
- **Engine**: representa al contenedor web.
- **Host**: representa un host (o un host virtual). Mediante `appBase` se especifica el

directorio de donde "colgarán" las aplicaciones web (por defecto `webapps`)
- **Context:** representa una aplicación web. Veremos de manera más detallada su configuración.
- **DefaultContext**: se aplica por defecto a aquellas aplicaciones que no tienen `context` propio.

Hay una serie de elementos que se pueden definir a varios niveles, de modo que por ejemplo pueden afectar a todo el servidor o solo a una aplicación web:

- **Valve**: es un componente que puede "filtrar" las peticiones.
- **Logger**: define el funcionamiento de los *logs* para depuración de errores (no los *logs* de acceso, que se definen mediante un `valve`).
- **Realm**: define un conjunto de usuarios con permisos de acceso a un determinado contexto.
- **Manager**: implementa el manejo de sesiones HTTP. Se puede configurar para que las sesiones se almacenen de manera permanente cuando se apaga el servidor.
- **Loader**: cargador de clases para una aplicación web. Es raro usar uno distinto al que viene por defecto.

1.2. Formas de cambiar la configuración

La configuración de Tomcat está almacenada en cuatro ficheros que se encuentran en el directorio **`conf`**. Tres de ellos están en formato XML y el cuarto es un fichero de políticas de seguridad en el formato estándar de Java:

- `server.xml`: el fichero principal de configuración.
- `web.xml`: es un fichero en el formato estándar para aplicaciones web con servlets, que contiene la configuración global a todas las aplicaciones.
- `tomcat-users.xml`: lista de usuarios y contraseñas para autentificación.
- `catalina.policy`: políticas de seguridad para la ejecución del servidor.

Además se puede cambiar gran parte de la configuración a través de la aplicación de administración.

1.3. Configurar el host

Mediante el elemento `Host` se define la configuración para un host o host virtual

```
<Host name="localhost" debug="0" appBase="webapps" u
      npackWARs="true" autoDeploy="true">...
```

Algunos de los principales atributos de este elemento son los siguientes:

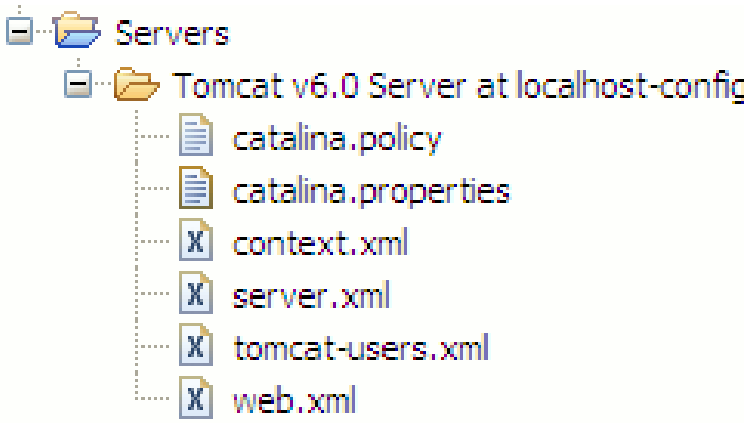
Atributo	Significado	Valor por defecto
`name`	nombre del host o host virtual	ninguno
`debug`	nivel de mensajes de	0

	depuración	
`appBase`	directorio donde se instalarán las aplicaciones de este host (si es relativa, la ruta se supone con respecto al directorio de Tomcat)	ninguno
`unpackWARs`	si es `true`, las aplicaciones empaquetadas en WAR se desempaquetan antes de ejecutarse	`true`
`autoDeploy`	si es `true`, se utiliza el despliegue automático de aplicaciones	`true`
`liveDeploy`	si es `true`, se utiliza despliegue automático sin necesidad de rearrancar Tomcat	`true`

El despliegue automático de aplicaciones es una interesante característica que permite instalarlas sin más que dejar el WAR o la estructura de directorios de la aplicación dentro del directorio `appBase`. Esta característica se puede desactivar poniendo `autoDeploy` y `liveDeploy` a `false`. Veremos más sobre ello en el punto siguiente.

1.4. Configuración en Eclipse WebTools

Al ejecutar aplicaciones mediante WebTools, Eclipse está empleando una copia de los ficheros de configuración de Tomcat, de modo que puedan cambiarse sin afectar a la instalación del servidor. Dicha configuración es accesible a través de la carpeta `Servers` que aparece en Eclipse como si fuera un proyecto más. Por cada *runtime* de servidor web configurado en Eclipse tendremos una subcarpeta dentro de `Servers`, en la que aparecerán los ficheros de configuración del servidor que ya hemos visto: `server.xml`, `tomcat-users.xml`,.... Aunque no es necesario, ni recomendable salvo que sepamos con seguridad lo que estamos haciendo, podemos modificar manualmente estos archivos si necesitamos una configuración más flexible que la que ofrecen las opciones de WebTools.



2. Configuración de aplicaciones web en Tomcat

Una vez visto cómo cambiar aspectos comunes a todas las aplicaciones, veamos cómo configurar cada aplicación de modo individual. El mecanismo estándar en aplicaciones web JavaEE es el descriptor de despliegue, aunque Tomcat aporta algo de funcionalidad extra a través del *contexto*.

2.1. El descriptor de despliegue

Como hemos dicho anteriormente, el directorio `WEB-INF` de una aplicación web con servlets y/o páginas JSP, debe haber un fichero descriptor de despliegue (llamado **`web.xml`**) que contenga la información relativa a la aplicación.

El `web.xml` es estándar en JavaEE y por tanto todo lo visto en esta sección es igualmente aplicable a cualquier servidor compatible JavaEE, aunque no sea Tomcat.

Es un fichero XML, que comienza con una cabecera XML que indica la versión y la codificación de caracteres, y un `DOCTYPE` que indica el tipo de documento, y la especificación de servlets que se sigue. La etiqueta raíz del documento XML es **`web-app`**. Así, un ejemplo de fichero podría ser:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">
        <display-name>
                Mi Aplicacion Web
        </display-name>
        <description>
```

```
                Esta es una aplicacion web sencilla a modo de
ejemplo
        </description>
</web-app>
```

En este caso se está utilizando la especificación 2.5 de servlets. Algunos servidores permiten omitir la cabecera `xml` y el `DOCTYPE`, pero sí es una buena costumbre el ponerlas.

Dentro de la etiqueta raíz `<web-app>` podemos colocar otros elementos que ayuden a establecer la configuración de nuestra aplicación web. Veremos a continuación algunos de ellos. En algunos elementos profundizaremos un poco más, por tratarse de elementos genéricos de una aplicación web (variables globales, etc). En otros (servlets, filtros, etc), simplemente se indicará qué elementos se tratan, pero su configuración se explicará en temas más específicos.

A partir de la versión 2.4 de la especificación (Tomcat 5.5), el orden de las etiquetas dentro del archivo es libre. No obstante, las discutiremos en el orden que había que seguir hasta esta versión.

1. Información general de la aplicación

Primero tenemos etiquetas con información general:

- **`<display-name>`**: nombre con que deben utilizar las aplicaciones gráficas para referenciar a la aplicación
- **`<description>`**: texto descriptivo de la aplicación

Variables globales

Podemos tener varias etiquetas:

- **`<context-param>`**: para declarar las variables globales a toda la aplicación web, y sus valores. Dentro tiene dos subetiquetas:
  - **`<param-name>`**: nombre de la variable o parámetro
  - **`<param-value>`**: valor de la variable o parámetro

Un ejemplo:

```
<context-param>
        <param-name>param1</param-name>
        <param-value>valor1</param-value>
</context-param>
```

Estos parámetros pueden leerse desde servlets con el método **`getInitParameter`** del objeto `ServletContext`.

2. Filtros

Para el tratamiento de filtros se tienen las etiquetas:

- **`<filter>`**: para asociar un nombre identificativo con la clase que implementa el filtro
- **`<filter-mapping>`**: para asociar un nombre identificativo de filtro con una URL o

patrón de URL

Se pueden tener varias de estas etiquetas, cada una para un filtro.

3. Oyentes

Se tiene la etiqueta:

- **`<listener>`**: para definir una clase oyente que responde ante eventos en sesiones y contextos (al iniciar, al cerrar, al modificar).

4. Servlets

Para definir los servlets en nuestro fichero de configuración, se tienen las etiquetas:

- **`<servlet>`**: asocia un nombre identificativo con una clase Java que implementa un servlet
- **`<servlet-mapping>`**: asocia un nombre identificativo de servlet con una URL o patrón de URL.

Se pueden tener varias de estas etiquetas, cada una para un servlet.

5. Configuración de sesión

Se tiene la etiqueta:

- **`<session-config>`**: para indicar parámetros de configuración de las sesiones.

Por ejemplo, podemos indicar el tiempo (en minutos) que le damos a una sesión de usuario antes de que el servidor la finalice:

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

6. Páginas de inicio

Se tiene la etiqueta:

- **`<welcome-file-list>`**: para indicar qué páginas debe buscar Tomcat como páginas de inicio en el caso de que en la URL se indique el directorio, pero no la página, como por ejemplo:

  ```
  http://localhost:8080/unadireccion/dir/
  ```

  Para ello, esta etiqueta tiene una o varias subetiquetas **`<welcome-file>`** para indicar cada una de las posibles páginas

Por ejemplo, podemos indicar que las páginas por defecto sean `index.html` o `index.jsp` con:

```
<welcome-file-list>
        <welcome-file>index.html</welcome-file>
```

```
        <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

Las páginas se buscan en el orden en que se especifican en esta etiqueta.

7. Librerías de tags

Se tiene la etiqueta:

- **`taglib`**: para cargar una librería de tags para utilizar en páginas JSP. Podemos tener una o varias de estas etiquetas.

8. Seguridad

Para gestionar la seguridad en las aplicaciones Web se tienen las etiquetas:

- **`security-constraint`**: permite especificar qué URLs de la aplicación deben protegerse
- **`login-config`**: indica cómo debe autorizar el servidor a los usuarios que quieran acceder a las URLs protegidas (indicadas con `security-constraint`)
- **`security-role`**: da una lista de roles en los que se encuadrarán los usuarios que intenten acceder a recursos protegidos.

Existen otras etiquetas internas, relacionadas con la seguridad, que no se encuentran detalladas aquí, ya que las veremos cuando hablemos de la seguridad en el servidor.

2.2. El contexto de la aplicación en Tomcat

Aunque el comportamiento interno de cada aplicación se define desde su propio fichero `web.xml`, se pueden especificar también sus características a nivel global. Esto se denomina en Tomcat el *contexto* de la aplicación. Se puede definir un contexto por defecto con propiedades comunes a todas las aplicaciones y además un contexto para cada aplicación de modo individual.

2.2.1. Un ejemplo

Lo recomendado es definir el contexto en formato XML en un fichero denominado precisamente `context.xml`. Este fichero se colocaría en un directorio `META-INF` dentro del .war. Por ejemplo:

```
<Context reloadable="true">
 <WatchedResource>META-INF/miConfig.xml</WatchedResource>
</Context>
```

La configuración se hace a través de atributos del elemento `Context` o bien subelementos dentro de este. En el ejemplo anterior, el atributo `reloadable` indica que Tomcat debe recargar la aplicación cuando se modifique algo de su código Java. La etiqueta `WatchedResource` hace algo parecido pero sirve para cualquier tipo de fichero. Así,

cuando modificáramos el archivo `miConfig.xml`, Tomcat recargaría la aplicación permitiendo que esta tenga en cuenta los cambios.

Nota:

Tomcat es extremadamente flexible en cuanto a dónde resida físicamente la definición del contexto. Además de lo que hemos visto, se puede colocar en el propio `server.xml`, en un archivo XML dentro de `$TOMCAT_HOME/conf/`, etc. Esto hace la configuración a veces un poco confusa. Se recomienda consultar la documentación de Tomcat para ver todas las posibilidades.

En la documentación distribuída con Tomcat se incluye una referencia de la configuración del contexto. Se recomienda consultarla para tener una información más detallada.

2.2.2. Filtrando peticiones: valves

Un **valve** es un componente que se inserta en el ciclo de procesamiento de la petición. Así, se pueden filtrar peticiones "sospechosas" de ser ataques, no aceptar peticiones salvo que sean de determinados *hosts*, etc. Esto se puede hacer a nivel global (dentro del `engine`), para un host en concreto (dentro de `host`) o para una única aplicación (dentro de `context`). Tomcat viene con varios *valves* predefinidos (que no son más que clases Java) , aunque el usuario puede escribir los suyos propios.

Registro de accesos (*access log valve*)

Crea un registro o *log* de accesos en el formato "estándar" para servidores web. Este *log* puede ser luego analizado con alguna herramienta para chequear el número de accesos, el tiempo que permanece cada usuario en el sitio, etc. Para crear un registro de accesos, introducir en el `server.xml` un elemento similar al siguiente:

```
<Valve className="org.apache.catalina.valves.AccessLogValve"
       directory="logs" prefix="localhost_access_log."
suffix=".txt"
       pattern="common" resolveHosts="false"/>
```

Según el nivel en el que se introduzca, se pueden registrar los accesos en todo el `host` o bien solo dentro de una aplicación (`context`). El nombre del fichero de *log* se compone con el prefijo asignado a través del atributo `prefix`, la fecha del sistema en el formato `aaaa-mm-dd` y el sufijo elegido con el atributo `suffix`. Cuando cambia la fecha automáticamente se crea un nuevo fichero de *log*.

La información que aparece en el *log* es configurable. Si en el atributo `pattern` se especifica el valor `common`, se utiliza el formato estándar, típico de otros servidores web como Apache. Al especificar el valor `combined`, a la información anterior se añade el valor de los campos `User-agent` y `Referer` de la petición HTTP. Si estos formatos no cubren nuestras necesidades, se puede hacer uno "a medida" empleando los códigos de formato que aparecen en la documentación de Tomcat.

Filtro de hosts y de peticiones(*remote host filter* y *remote address filter valve*)

Sirven para permitir o bloquear el acceso desde determinados host o desde determinadas direcciones IP. El nombre de los hosts o rango de direcciones se puede especificar mediante expresiones regulares. Por ejemplo, para permitir el acceso a la aplicación de administración únicamente desde el host local se introduciría en el `server.xml` algo como

```
<Context path="/admin" docBase="admin">
        ...
        <Valve
className="org.apache.catalina.valves.RemoteAddrValve"
               allow="127.0.0.1" />
        ...
</Context>
```

El atributo `allow` sirve para especificar hosts o IPs permitidas y `deny` para especificar las prohibidas. En caso de no especificar valor para `allow`, se utilizará el de `deny` para denegar peticiones y se aceptará el resto. De manera similar, si no se especifica valor para deny, se permitirán peticiones según el valor de `allow` y se rechazará el resto.

Volcado de la petición (*request dumper valve*)

Este valve se puede utilizar para depurar aplicaciones, ya que guarda toda la información de la petición HTTP del cliente. No utiliza más atributo que `className` para indicar el nombre de la clase que implementa este *valve*:

```
<Valve className="org.apache.catalina.valves.RequestDumperValve" />
```

Autentificación única (single sign-on valve)

Se puede utilizar un Valve para que el usuario se identifique en una única aplicación y automáticamente conserve la misma identidad en todas las aplicaciones del mismo host, sin necesidad de identificarse de nuevo (*single sign-on valve*). Basta con añadir a nivel de Host un Valve como:

```
<Valve className="org.apache.catalina.authenticator.SingleSignOn"
/>
```