UD 3. Servidores de Aplicaciones (II)

Índice

Estructura y recursos que componen una aplicación web.

Las aplicaciones web se basan en la versión Java EE 8 Web. Una aplicación web está compuesta de una serie de servlets, páginas jsp, ficheros html, ficheros de imágenes, ficheros de sonidos, texto, clases, etc.; de forma que todos estos recursos se pueden empaquetar y ejecutar en varios contenedores distintos. Se debe especificar como se van a estructurar los ficheros para que la aplicación se despliegue en un servidor de aplicaciones de forma óptima. Para ello se necesita un fichero denominado **descriptor de despliegue** (fichero .xml) que especifica cómo se despliega la aplicación y los recursos en el servidor.

La aplicación web está compuesta por información que se puede resumir en:

- Servlets: módulos escritos en Java utilizados en un servidor para potenciar sus capacidades de respuesta y que se ejecuta sobre el protocolo HTTP.
- Páginas jsp: tecnología de programación que permite crear páginas web dinámicas usando HTML y XML.
- PHP: lenguaje de propósito general usado en el backend.
- Perl: lenguaje de programación que generalmente se ejecuta en el servidor.
- Ruby: lenguaje interpretado de propósito general, dinámico y flexible.
- ASP: lenguaje de scripting del lado del servidor creado por Microsoft.
- ASP.NET: lenguaje diseñado para trabajar con IIS y escrito para soportar el .net Framework.
- Phyton: lenguaje interpretado de alto nivel.
- Ficheros HTML: lenguaje de marcas para mostrar información sobre el navegador.

- Ficheros de hoja de estilos CSS: usados con páginas HTML para facilitar el diseño de páginas web.
- Ficheros .java: ficheros de clases para programar clases en java.
- Ficheros de sonidos: sonidos que necesite la aplicación web.
- Ficheros de imágenes: fondos para páginas web, imágenes utilizadas.

La información anterior y la que se necesite se empaqueta y ejecuta de forma ordenada mediante el fichero descriptor de despliegue.

Servlet

Un servlet es una aplicación java encargada de realizar un servicio específico dentro de un servidor web. La versión 4.0 posibilita el uso de HTTP 2.0 que permite operaciones push. Es decir, el servidor puede enviar recursos al cliente.

A continuación se indica la estructura de directorios para los ficheros de una aplicación web. El directorio raíz debería tener el nombre de la aplicación y define la raíz de documentos para la aplicación web. Todos los ficheros debajo de esta raíz pueden servirse al cliente excepto aquellos ficheros que están bajo los directorios especiales META-INF y WEB-INF en el directorio raíz. Todos los ficheros privados, al igual que los ficheros class de los servlets, deberían almacenarse bajo el directorio WEB-INF.

Durante la etapa de desarrollo de una aplicación web se emplea la estructura de directorios, a pesar de que luego en la etapa de producción, toda la estructura de la aplicación se empaqueta en un archivo .war

El código necesario para ejecutar correctamente una aplicación web se encuentra distribuido en una estructura de directorios, agrupándose ficheros según su funcionalidad. A continuación se detallan las carpetas más importantes de una aplicación web:

- /: En la carpeta raíz del proyecto se almacenan elementos empleados en los sitios web, tipo documentos html, CSS y los elementos JSP (*.html *.jsp *.css).
- **/WEB-INF/**: Es un área privada de la aplicación donde no se puede acceder directamente desde el navegador. Aquí se encuentran los elementos de configuración del archivo .WAR

como pueden ser: la página de inicio, la ubicación de los servlets, parámetros adicionales para otros componentes. El más importante de éstos es el archivo **web.xml.**

- **/WEB-INF/classes/**: Contiene las clases Java empleadas en el archivo .WAR y, normalmente, en esta carpeta se encuentran los servlets codificados.
- **/WEB-INF/lib/**: Contiene los archivos JAR utilizados por la aplicación y que normalmente son las clases empleadas para conectarse con la base de datos o las empleadas por librerías de JSP.
- **META-INF**: directorio privado de la aplicación que contiene normalmente un fichero denominado context.xml que permite definir el contexto de la aplicación.

Un ejemplo de la estructura de carpetas de una aplicación web puede ser el siguiente:

/index.jsp
/WebContent/jsp/welcome.jsp
/WebContent/css/estilo.css
/WebContent/js/utils.js
/WebContent/img/welcome.jpg
/WEB-INF/web.xml
/WEB-INF/struts-config.xml
/WEB-INF/lib/struts.jar
/WEB-INF/src/com/empresa/proyecto/action/welcomeAction.java
/WEB-INF/classes/com/empresa/proyecto/action/welcomeAction.class

Descriptor de despliegue.

El descriptor de despliegue es un fichero .xml que describe como se ha de desplegar una aplicación web, un módulo o cualquier componente. Al se un fichero xml de puede modificar de manera fácil sin cambiar nada del código fuente.

Cualquier aplicación web debe tener un descriptor de despliegue, en caso contrario no funcionará. El descriptor de despliegue de una aplicación debe estar en la carpeta */WEB-INF/web.xml*

En Tomcat podemos encontrar un descriptor de despliegue general en la carpeta */opt/tomcat9/apache-tomcat-9.0.54/conf/web.xml*

Si editamos el archivo anterior, comprobamos como los servlets están entre las etiquetas *<web-app>* y *</web-app>*. Para acceder a un servlet, hay que teclear la siguiente URL en un navegador: http://localhost:8080/nombre_servlet
donde nombre_servlet es el nombre definido entre las etiquetas *<servlet-name>* y *</servlet-name>* del servlet.

Un ejemplo de descriptor de despliegue puede ser el siguiente archivo web.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app>
  <servlet>
    <servlet-name>nombre</servlet-name>
    <servlet-class>package.nombre.MiClass</servlet-class>
    ...
  </servlet>
  ...
</web-app>
```

Situadas entre las etiquetas *<web-app>* y *</web-app>* estarían los descriptores de despliegue de servlets.

Archivos war

Como se ha visto anteriormente, el archivo war constituye una forma alternativa de distribuir aplicaciones Web empaquetando toda la aplicación (a partir de su directorio inicial) dentro de un fichero WAR.

Arquitectura de Apache-Tomcat.

A continuación se muestra la estructura del servidor de aplicaciones Apache-Tomcat para comprender el funcionamiento a grandes rasgos de la aplicación. La arquitectura permite de forma lógica conectar sus diferentes componentes y que cada uno realice una función determinada.

Los componentes que posee el servidor de aplicaciones Tomcat son los siguientes:

- **Server**: representa todo el contenedor al completo y dentro de él se ejecutan los demás componentes. Tomcat proporciona una implementación predeterminada de la interfaz del servidor, que generalmente no requiere que los usuarios la implementen ellos mismos. El contenedor del servidor puede contener uno o más componentes de servicio.

- **Service**: es un componente intermedio que se encuentra dentro del componente server y une uno o más componentes del conector a un solo motor engine. En el servidor, puede incluir uno o más componentes de servicio. Los usuarios rara vez personalizan el Servicio. Tomcat proporciona una implementación predeterminada de la interfaz del Servicio, que es simple y puede satisfacer las aplicaciones.

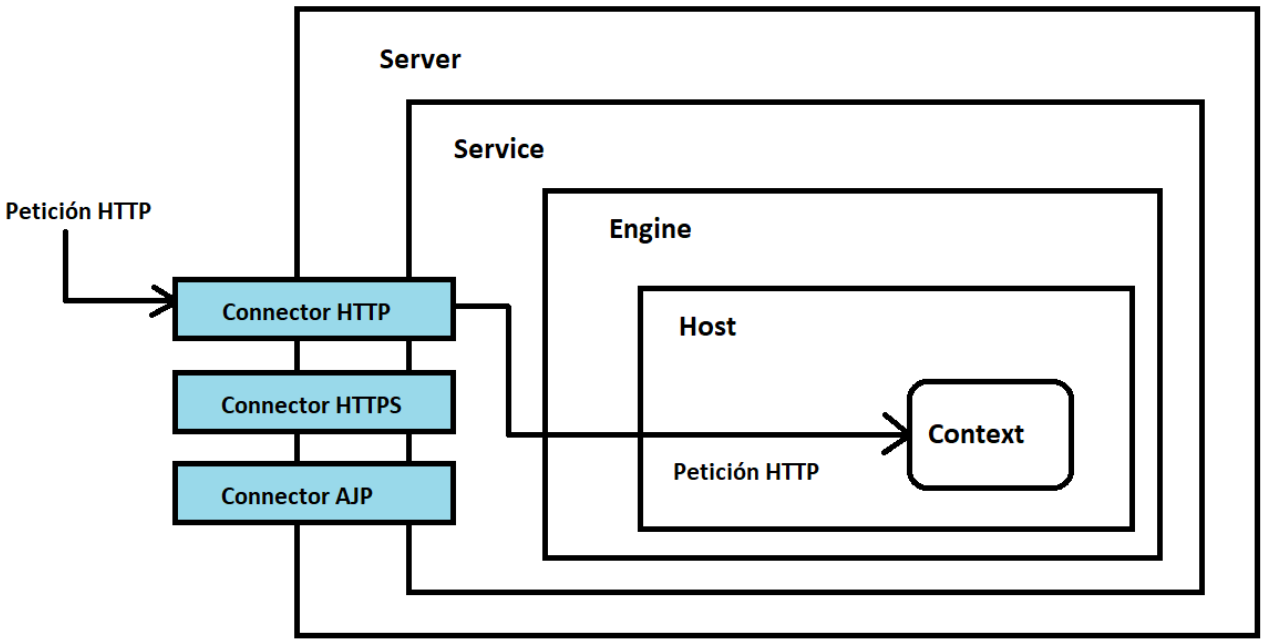
- **Engine**: en Tomcat, cada Servicio solo puede contener un Servlet Engine. Es el servicio que permite procesar las peticiones que provienen desde el exterior y responder a tales solicitudes. El motor representa la canalización de procesamiento de solicitudes para un Servicio en particular. Como servicio, puede haber varios conectores: el motor recibe y procesa todas las solicitudes del conector, devuelve la respuesta al conector apropiado y la transmite al usuario a través del conector. Los usuarios pueden proporcionar motores personalizados mediante la implementación de la interfaz del motor, pero esto generalmente no es necesario.

- **Host**: representa un host virtual, y un motor puede contener múltiples hosts. Los usuarios generalmente no necesitan crear un Host personalizado, porque la implementación de la interfaz de Host proporcionada por Tomcat (clase StandardHost) proporciona una funcionalidad adicional importante. Es un nombre de red que puede poseer varios host o alias, como www.daw.com

- **Connector**: maneja la comunicación con el cliente y es responsable de recibir la solicitud del cliente y devolver el resultado de la respuesta al cliente. En Tomcat, hay varios conectores disponibles como AJP Connector o HTTP Connector.

- **Context**: representa una aplicación web que se ejecuta en un host virtual específico. Un host puede contener múltiples contextos (que representan una aplicación web), y cada contexto tiene una ruta única. Los usuarios generalmente no necesitan crear contextos personalizados, porque la implementación de la interfaz de contexto (clase StandardContext) dada por Tomcat proporciona una funcionalidad adicional importante.

A continuación se muestra la arquitectura descrita:



En la ruta en la que hemos instalado el servidor Tomcat */opt/tomcat9/apache-tomcat-9.0.54* encontramos los directorios siguientes:

- **bin**: directorio que contiene los ficheros binarios y los scripts de inicio. Los dos ficheros importantes en Linux son *startup.sh* (para iniciar el servidor Tomcat) y *shutdown.sh* (para parar el servidor Tomcat).
- **conf**: permite la configuración global del servidor de aplicaciones. A continuación se indica los principales ficheros que contiene.
  - **catalina.policy**: la política de seguridad de Tomcat, relacionada con Java, se realiza a partir de este fichero.
  - **catalina.properties**: fichero en el que se relacionan los ficheros .jar de Java para la clase Catalina. Esta información está relacionada con los paquetes de seguridad y las rutas de las clases cargadas. Además contiene configuraciones relacionadas con la caché.
  - **context.xml**: fichero xml que contiene la información de contexto común a todas las aplicaciones web que se ejecutan en Tomcat. Además permite localizar el archivo *web.xml* de cada aplicación.

- **jaspic-providers.xml**: Java Authentication Service Provider Interface for Containers. Permite definir un módulo de autenticación.
- **jaspic-providers.xsd**: esquema xsd que define si es válido el fichero anterior.
- **logging.properties**: permite configurar las opciones de logging de todas las aplicaciones y de la actividad del servidor. Útil para solucionar errores en caso de fallo del servidor o de una aplicación.
- **server.xml**: fichero principal de configuración de Tomcat. Los principales parámetros utilizados son los siguientes:
  - **<server> </server>** Etiqueta principal. Engloba uno o más servicios. Contiene el atributo *port* que permite definir el puerto por el que se escucha.
  - **<Listener .../>** Define las extensiones JMX ("Java Management Extensions") usadas por Tomcat. Su atributo principal se denomina className.
  - **<GlobalNamingResources> </GlobalNamingResources>** Define los recursos tipo JNDI para usarlos globalmente en un servidor de aplicaciones. La etiqueta Resource especifica la localización.
  - **<Service> </Service>** Permite agrupar conectores.
  - **<Connector />** Representa las conexiones TCP desde el exterior que serán abiertas cuando arranca el servidor. Por ejemplo HTTPConnector.
  - **<Engine> </Engine>** Se usan dentro de la marca Service o de Host. Se procesan las peticiones que llegan a la marca Connector y que la cabecera disponga el Host por defecto.
  - **<Logger />** Indica donde serán enviados los registros de logs.
  - **<Host> </Host>** Permite definir varios hosts virtuales para atender las peticiones.
  - **<Context> </Context>** Indica la ruta a partir de la cual se encontrarán las aplicaciones ejecutadas en Tomcat. Habitualmente estarán por debajo del directorio webapps.

- **tomcat-users.xml**: fichero xml que contiene los usuarios, contraseñas y roles usados para acceder al servidor Tomcat.
- **tomcat-users.xsd**: esquema xsd que define si es válido el fichero anterior.
- **web-xml**: fichero estándar para las aplicaciones web, común a todas las aplicaciones web, ya que posee la configuración global a todas ellas.

- **lib**: directorio que contiene todos los ficheros .jar usados en el servidor.

- **logs**: directorio donde se almacenarán los ficheros de log.
- **temp**: directorio para almacenar los ficheros temporales de Tomcat.
- **webapps**: directorio que contiene todas las aplicaciones web.
- **work**: directorio de almacenamiento temporal de ficheros, por ejemplo de compilación.

Variables de entorno.

A continuación se indica las principales variables de entorno que utiliza Apache-Tomcat:

- **CATALINA_HOME**: indica el directorio raíz de la instalación del servidor Tomcat. En nuestro caso */opt/tomcat9/apache-tomcat-9.0.54*. En Windows tendrá la forma *C:\Program Files\omcat9\apache-tomcat-9.0.54*.
- **CATALINA_BASE**: directorio que representa la configuración de una instancia de Tomcat. Normalmente coincide con CATALINA_HOME. Solamente se diferencia cuando existan varias instancias en la misma máquina.
- **CATALINA_TMPDIR**: directorio temporal de Apache-Tomcat donde se almacenan ficheros de compilación, intermedios, etc.
- **JRE_HOME**: indica la ruta donde se encuentran los ejecutables de Java usados por Apache-Tomcat.
- **CLASSPATH**: conjunto de rutas que usa Apache-Tomcat , especialmente los ficheros .jar.

Vamos a añadir la variable de entorno CATALINA_HOME utilizando el fichero */etc/profile.d/tomcat9.sh*, en caso de que no esté ya añadida. Editamos el fichero:

```
sudo gedit /etc/profile.d/tomcat9.sh
```

Añadimos las líneas siguientes:

```
export CATALINA_HOME=/opt/tomcat9/apache-tomcat-9.0.54
export PATH="$PATH:$CATALINA_HOME"
```

Cambiamos los permisos en el fichero, para añadir permisos de ejecución, en caso de que no lo hayamos hecho anteriormente:

```
sudo chmod 755 /etc/profile.d/tomcat9.sh
```

Por ejemplo, para ejecutar ahora el fichero de arranque del servidor, bastará con que ejecutemos:

$CATALINA_HOME/bin/startup.sh