Tema 3: Administración de servidores de aplicaciones

Una aplicación web es aquella que funciona a través de una red. Amazon, Google, Facebook o Twitter son ejemplos de aplicaciones web. Una aplicación web funciona en HTTP sobre TCP/IP y los clientes acceden a ella mediante un navegador web. Generalmente es una aplicación que se ajusta al modelo cliente servidor y consta de tres capas ya que hace uso de una base de datos.

El término Servidor de Aplicaciones está asociado históricamente a la plataforma Java Enterprise Edition aunque en la actualidad engloba un conjunto de lenguajes y características mucho más amplio.

Un servidor de aplicaciones es una plataforma que provee aplicaciones software proporcionando servicios como seguridad, transacciones, acceso a datos, etc.

Existen muchos servidores de aplicaciones, siendo los más frecuentes *WebLogic Application Server* de *Oracle*, *WebSphere Application Server* de *IBM*, *Tomcat* de *Apache*. Muy relacionado con éste último está *GlassFish*, que fue desarrollado por *Sun Microsystems* y actualmente pertenece a *Oracle*.

Nosotros veremos *Tomcat* aunque *GlassFish* es más completo. ¿Por qué existen dos basados en lo mismo? *Tomcat* solo soporta los **contenedores web** de la plataforma J2EE mientras que *GlassFish* soporta todos los tipos por lo es realmente un servidor de aplicaciones en un sentido más completo.

Vamos a ver Tomcat por su simplicidad, facilidad de uso y configuración, bajo consumo de recursos y alto rendimiento y por que es muy utilizado, pero en este caso no existe un paralelismo con lo que veíamos con Apache y el mercado está mucho más fragmento. Es muy

probable que *Tomcat* vaya dejando paso a *GlassFish*. Para pequeñas aplicaciones se sigue recomendando *Tomcat*, pero si necesitamos J2EE al completo tendremos que buscar otra opción.

Con el tiempo Java ha ido perdiendo pujanza como lenguaje de programación para la web. Últimamente se ha ido quedando para aplicaciones grandes por características como el balanceo de carga, fácil escalabilidad o las posibilidades de comunicación entre sistemas de backend. Sin embargo se considera muy pesada para las aplicaciones más pequeñas y está siendo rápidamente sustituido por lenguajes como PHP, Ruby o Python. Un ejemplo de esto es Twitter que fue desarrollado con Ruby on Rails pero cuando vieron las dificultades que tenían para escalar la aplicación la migraron a Java.

Por si fuera poco los últimos problemas de seguridad de Java han llevado a las grandes compañías de Sistemas Operativos y de desarrollo de navegadores web a recomendar deshabilitar Java de los navegadores. Incluso “Homeland Security” un organismo gubernamental americano ha hecho esta recomendación sugiriendo que se deshabilite Java de todos los navegadores web excepto donde sea necesario. (Diciembre 2012 - Enero 2013). Oracle está intentando reaccionar sacando parches, pero por ejemplo el último (14/01/2013) lo único que hace es aumentar el nivel de seguridad por defecto y dejar en manos del usuario la decisión de ejecutar o no código no firmado.

¿Puede ser esto el fin de Java en la web? Difícilmente. Antes o después Oracle reaccionará y hay demasiadas aplicaciones enormes realizadas en Java para que desaparezca del mapa en los próximos años. Lo que sí es posible es que deje de usarse como lenguaje de desarrollo de aplicaciones web de tamaño pequeño o medio.

En mi opinión Java quedará más para el *backend* de las aplicaciones mientras que el *frontend* será desarrollado en lenguajes más ligeros. La integración de Tomcat con Apache va en esta línea.

Arquitectura

Existen varias versiones de Apache Tomcat. La diferencia entre unas versiones y otras es la versión de *Servlets* y *JSPs* que pueden manejar. En este enlace se pueden consultar las diferentes versiones, para qué especificaciones sirven y qué versión de *Java* es necesaria para su uso.

Apache Tomcat utiliza una estructura en la que varios componentes se organizan me manera jerárquica. Estos componentes son *Server, Service, Engine, Host, Connector y Context*.

> Lee la página de la documentación de Tomcat que explica estos conceptos. Entiende muy someramente qué hace cada uno (ya que luego los veremos en más profundidad) y cuál es la relación entre unos y otros. ¿Cuáles tendremos que modificar durante nuestro trabajo? Realizar un dibujo puede ayudarte en la tarea. Posteriormente discute en clase los conceptos aprendidos.

Cuando hayamos instalado Tomcat podremos ver ejemplos de todos estos elementos. Excepto Context, pueden consultarse todos en

```
sudo gedit /etc/tomcat7/server.xml
```

> Echa un vistazo al archivo /etc/tomcat7/server.xml

La estructura de directorios de Tomcat

Los directorios que se incluyen en una instalación de Tomcat se ubican generalmente en el directorio donde hemos descomprimido Tomcat. Sin embargo, en esta instalación están definidos en dos ubicaciones: */usr/share/tomcat7* y */var/lib/tomcat7.* Sin embargo hay otro directorio afectado en la instalación */etc/tomcat7*

Los directorios comunes son:

- **bin**: contiene los binarios y scripts de inicio de Tomcat.
- **conf**: la configuración global de Tomcat. En esta instalación es un enlace simbólico a */etc/tomcat7*. Tiene algunos archivos que merece la pena destacar.
  - **catalina.policy**: Este archivo contiene la política de seguridad relacionada con Java e impide que los Servlets o JSPs la sobrescriban por motivos de seguridad. En esta instalación se encuentra en policy.d/03catalina.policy
  - **catalina.properties**: Contiene los archivos .JAR que no pueden sobrescribirse por motivos de seguridad y otros de uso común.
  - **context.xml**: El archivo de contexto común a todas las aplicaciones. Se utiliza principalmente para informar de dónde se puede encontrar el archivo web.xml de las propias aplicaciones. Contiene la configuración que será común a todos los elementos Context.
  - **logging.properties**: Establece las políticas generales para el registro de actividad del servidor, aplicaciones o paquetes.
  - **server.xml**: ya hemos visto que es el fichero principal de configuración de Tomcat y que tiene mucho que ver con su arquitectura.
  - **tomcat-users.xml**: Contiene los usuarios, contraseñas y roles usados para el control de acceso. Es el archivo donde se encuentra la información de seguridad para las aplicaciones de administración de Tomcat. Todos los valores son por defecto así que deben cambiarse en caso de “descomentar” las líneas.
  - **web.xml**: Un descriptor de despliegue por defecto con la configuración compartida por todas las aplicaciones. Es un archivo con directivas de funcionamiento de las aplicaciones.
  - Además de todos estos archivos, existe un subdirectorio para cada motor con un subdirectorio localhost donde irá otro archivo de contexto específico para cada aplicación. Este archivo tiene la forma nombre-de-aplicación.xml. En este caso el

motor es Catalina así que están en *Catalina/localhost*. Puedes ver que hay uno por cada paquete adicional que instalamos, documentación, ejemplos y aplicaciones de administración.

- **lib**: continen todos los .JAR comunes a todas las aplicaciones. Aquí van archivos de Tomcat, APIs de JSPs, etc y en él podemos ubicar archivos comunes a las diferentes aplicaciones como MySQL JDBC.
- **logs**: aquí van los archivos de registro. En esta instalación es un enlace simbólico a */var/log/tomcat7*
- **temp**: este directorio es opcional. En esta instalación no está creado ni activado por defecto. Se usa para los archivos temporales que necesita Tomcat durante su ejecución.
- **webapps**: aquí se ubican las aplicaciones propiamente dichas. Ahora solo hay una que se denomina ROOT. Podría haber más pero como hemos instalado la documentación, ejemplos y administradores aparte se encuentran en su propio directorio en /usr/share/tomcat7-admin, tomcat7-docs y tomcat7-examples.
- **work**: es un directorio para los archivos en uso, cuando se compilan los JSPs, etc.

Busca estos directorios en la instalación de Tomcat que realizamos en el tema 1.

Un vistazo más profundo a la arquitectura de Tomcat

El término arquitectura de Tomcat también se refiere a la estructura XML que sigue su organización. Se compone de un conjunto de elementos representados por etiquetas. En la imagen puede verse cómo se relacionan unos con otros. Son los elementos que visteis antes, y vamos a repasarlos.

Server

Es el primer elemento superior y representa una instancia de Tomcat. Es equivalente al servidor en si con un puerto asociado. Pueden existir varios en diferentes puertos y a veces se hace para que si una aplicación falla arrastrando al servidor esto no afecte a otras aplicaciones.

Contiene varios *Listeners* que escuchan y responden a eventos.

También usa GlobalNamingResources que sirven para permitir a clientes software hechos en Java encontrar objetos y datos mediante su nombre. Por ejemplo aquí se podría indicar un recurso global como una base de datos MySQL.

Service

El servicio agrupa un contenedor de tipo *Engine* con un conjunto de conectores. El motor suele ser Catalina y los conectores por defecto HTTP y AJP.

Connector

Los conectores sirven para comunicar las aplicaciones con clientes (por ejemplo un navegador web u otros servidores). Representan el punto donde se reciben las peticiones y se les asigna un puerto IP en el servidor.

Containers

Tomcat se refiere a Engine, Host, Context, y Cluster como contenedores. El de nivel más alto es Engine y el más bajo Context. Algunos componentes como Realm o Valve pueden ubicarse dentro de un contenedor.

Engine

El motor procesa las peticiones y es un componente que representa el motor de Servlets Catalina. Examina las cabeceras (por ejemplo de las tramas HTTP) para determinar a host (o virtual host) o context se le debe pasar cada petición.

Cuando Tomcat se utiliza como servidor autónomo se usa el motor por defecto. Cuando Tomcat se usa dando soporte a un servidor web se sobrescribe porque el servidor web ya ha determinado el destino correcto para las peticiones.

Un motor puede contener Hosts que representan un grupo de aplicaciones web o Context que representa a una única aplicación.

Tomcat se puede configurar con un único host o múltiples hosts virtuales como Apache.

Host

Define un host por defecto o múltiples hosts virtuales en Tomcat. En Tomcat los hosts virtuales se diferencian por nombres de dominio distintos, por ejemplo www.aplicacion1.es y www.aplicacion2.es. Cada uno soporta varios Context.

Context

Este elemento es equivalente a una aplicación web. Hay que informar al motor y al host de la localización de la carpeta raíz de aplicación. También se puede habilitar la recarga dinámica (dynamic reload) para que al modificar alguna clase de la aplicación se modifique en la ejecución. Esta opción carga mucho el servidor por lo que se recomienda para pruebas pero no en producción.

En un contexto también se pueden establecer páginas de error específicas para armonizarlas con la apariencia de la aplicación.

Puede contener parámetros de inicio para establecer control de acceso en la aplicación.

Cluster

En caso de que tengamos más de un servidor Tomcat atendiendo las peticiones, este elemento nos permite configurarlo. Es capaz de replicar las sesiones y los parámetros de cada Context. Queda muy por encima del contenido del curso.

Realm

Se puede aplicar al nivel de Engine, Host o Context. Se utiliza para autenticación y autorización de usuarios o grupos. Pueden usarse con archivos de texto, servidores LDAP o bases de datos por ejemplo.

> ¿Qué es un servidor LDAP?

Si lo aplicamos a nivel de motor, los usuario o grupos tendrán idénticos permisos en todas las aplicaciones a la hora de acceder a objetos o recursos. Si lo aplicamos en un Host todas las aplicaciones los permisos serán para todas las aplicaciones de dicho entorno mientras que si lo aplicamos en un Context se establecerán por cada aplicación. Sin embargo, aunque un usuario tenga los mismos permisos para todas las aplicaciones debe seguir autenticándose en cada una de ellas por separado.

Los permisos se heredan y por eso se da la situación descrita en el párrafo anterior, pero si los permisos de un usuario o grupo se sobrescriben en un entorno de rango inferior los últimos serán los que se apliquen. Es similar a lo que veíamos con ciertas directivas (por ejemplo de directorios) en Apache.

Valve

Se usa para interceptar peticiones antes de pasárselas a las aplicaciones. Esto nos permite pre-procesar las peticiones para bloquear algunas, registrar accesos, registrar detalles de la conexión (en archivos log), o establecer un único punto de acceso para todas las aplicaciones de un host o para todos los hosts de un servidor. Afectan al tiempo de respuesta a la petición.

Puede establecerse para cualquier contenedor Engine, Host, and Context, y/o Cluster.

Hay un concepto similar en los servlets que se denomina Filtros (Filters).

Aunque en la sección siguiente veremos los archivos de configuración en detalle, echa un vistazo al archivo en /etc/tomcat/server.xml para afianzar estos conceptos.

Compara este archivo con el de la instalación de Tomcat que hicimos en el tema 1.

Configuración básica del servidor de aplicaciones

La configuración de Tomcat se realiza a través de uno o más ficheros XML. Hemos visto por encima los ficheros de configuración en los apartados anteriores. Para este propósito, los principales son: server.xml, context.xml y web.xml.

Tomcat busca estos archivos en el directorio especificado por $CATALINA_BASE, en un subdirectorio /conf. En nuestro caso establecimos

```
CATALINA_BASE=/var/lib/tomcat7
```

Pero si visitamos este directorio veremos que el subdirectorio /conf es un enlace simbólico a /etc/tomcat/ que es donde realmente se encuentran los archivos de configuración.

En el caso de que no se especifique $CATALINA_BASE se usa $CATALINA_HOME que es obligatoria para el arranque de Tomcat.

La configuración de Tomcat incluye infinidad de parámetros y no es el objetivo de este curso ser un experto administrador de él. Vamos a ver por encima los archivos de configuración principales y luego estudiaremos las configuraciones particulares más importantes.

La documentación de Tomcat incluye muchísima más información de todos estos elementos y sus posibles configuraciones.

server.xml

En el archivo de configuración por defecto podemos ver que se establece un único servicio y una única instancia del servidor. El elemento <Server> tiene la siguiente forma:

```
<Server port="8005" shutdown="SHUTDOWN">
```

Puede sorprendernos que el puerto de Tomcat sea 8080 y sin embargo esté configurado en el puerto 8005. Este puerto es en el que el que se inicia una instancia del servidor (JVM) y en el que escucha por si llegan señales de apagado (shutdown). Esta señal no se puede mandar desde otra máquina por motivos de seguridad, pero evidentemente se puede ejecutar el comando de apagado desde otra máquina y la señal ya llegará desde la misma en la que esté el servidor.

<Server>

El elemento <Server> puede contener otros tres:

- <Service>: Un grupo de conectores asociados con un motor. Es necesario al menos uno.
- <Listener>: Clases que tienen que escuchan y manejan eventos que tienen que ver con el ciclo de vida del servidor, por ejemplo después de arrancar, etc.
- <GlobalNamingResources>: Recursos globales que pueden ser usados en esta instancia del servidor por los componentes que los necesiten, por ejemplo una base de datos.

```
<GlobalNamingResources>
  <!-- Editable user database that can also be used by
       UserDatabaseRealm to authenticate users
  -->
  <Resource name="UserDatabase" auth="Container"
            type="org.apache.catalina.UserDatabase"
            description="User database that can be updated and saved"
            factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
            pathname="conf/tomcat-users.xml" />
</GlobalNamingResources>
```

<Service>

El propósito de un servicio es agrupar un motor que procese las peticiones con uno o más conectores que gestionen los protocolos de comunicación. El servicio por defecto es el motor Catalina.

```
    <Service name="Catalina">
```

Aquí se le ha dado al servicio el nombre del motor, pero no es necesario.

Un <Service> contiene al menos un <Connector> y solo un <Engine> que es obligatorio.

<Connector>

Este elemento tiene mucho que ver con los dos modos de funcionamiento de Tomcat:

- Como servidor único: En el que Tomcat realiza las funciones de servidor de aplicaciones y de servidor web.
- Como servidor de aplicaciones: En el que Tomcat colabora con un servidor web que hace de *frontend*. El servidor web dirige todas peticiones de JSPs y Servlets a Tomcat.

En un entorno con acceso público se suele usar la segunda configuración ya que el servidor web está mucho más preparado en términos de seguridad y privacidad.

Los dos conectores más comunes son HTTP y AJP. El segundo es usado para conectar con los servidores en el como colaborativo (Apache u otros). Ambos pueden funcionar con SSL para mejorar la seguridad.

El puerto por defecto para HTTP es 8080. Se puede cambiar y si por ejemplo Tomcat va a estar en producción como un servidor en solitario podríamos modificarlo al puerto estándar de HTTP, 80.

El conector HTTP está habilitado:

```
        <Connector port="8080" protocol="HTTP/1.1"
                   connectionTimeout="20000"
                   URIEncoding="UTF-8"
                   redirectPort="8443" />
```

mientras que el AJP no:

```
    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

Ninguno de los dos usa SSL por defecto.

<Engine>

El motor es quien procesa las peticiones realmente. El nombre es el que le demos a la instancia y defaultHost indica a qué host virtual se le pasará una petición en caso de que no se especifique ninguno ya que el mismo motor puede procesar peticiones dirigidas a múltiples hosts virtuales de los especificados en este archivo.

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

Un motor contiene uno o más <Host>, uno o ningún <Context>, uno o ningún <Realm>, multiples <Valve> y <Listener> aunque puede no tener ninguno.

<Realm>

Es un mecanismo de seguridad que sirve para autenticar usuarios y establecer seguridad a nivel de contenedor (recuerda qué elementos consideraba Tomcat contenedores según vimos en el apartado arquitectura). La configuración por defecto hace que Tomcat lea los usuarios del archivo tomcat-users.xml pero obviamente sería mejor configurarlo para que se usara una base de datos o servidor LDAP.

```
            <!-- Use the LockOutRealm to prevent attempts to guess user passwords
                 via a brute-force attack -->
            <Realm className="org.apache.catalina.realm.LockOutRealm">
              <!-- This Realm uses the UserDatabase configured in the global JNDI
                   resources under the key "UserDatabase".  Any edits
                   that are performed against this UserDatabase are immediately
                   available for use by the Realm.  -->
              <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                     resourceName="UserDatabase"/>
            </Realm>
```

<Host>

Este elemento representa un host virtual en Tomcat. La configuración por defecto solo define *localhost*. Si tenemos configurado el servidor DNS con un nombre para nuestro servidor usaremos éste.

El atributo *appBase* establece el directorio raíz de las aplicaciones. Se establece a partir de <CATALINA_BASE> si no se indica lo contrario. Por defecto la URL de cada aplicación es la resultante de añadir su directorio raíz a la del servidor. En nuestro ejemplo hemos instalado cuatro aplicaciones docs, examples, host-manager y manager. También se establece ROOT que indica la aplicación por defecto si no añadimos otras cada una en su directorio.

El atributo *unpackWARs* indica si este tipo de archivos debe ser descomprimido o no. En caso de no descomprimirlos, la ejecución será un poco más lenta.

autoDeploy indica si el despliegue de una aplicación que situemos en el directorio debe ser automático o no.

```
            <Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
```

Además contiene un <Valve> para registrar los accesos.

```
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log." suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
```

Investiga y discute en clase qué son los archivos WAR y su relación con los archivos JAR.

Configura un nombre en el DNS para tu servidor y cambia el nombre en <Host>. Prueba que puedes acceder con el nombre a todo.

context.xml

El contexto en Tomcat se puede establecer a mucho niveles y se aplicará el más específico en cada aplicación.

Los descriptores de contexto de administración de cada aplicación se encuentran en $CATALINA_BASE/conf/nombre_motor/nombre_host

En nuestro caso están en /etc/tomcat/Catalina/localhost

Por ejemplo el de host-manager

```
    <Context path="/host-manager"
        docBase="/usr/share/tomcat7-admin/host-manager"
        antiResourceLocking="false" privileged="true" />
```

El *path* indica cómo se accederá a la aplicación en nuestro servidor. El problema para cambiarlo es que habría que modificar todos los archivos xml relacionados para que Tomcat siga encontrando la aplicación.

docBase es el directorio de despliegue de la aplicación.

Los descriptores de contexto específicos de cada aplicación web están en el directorio de cada aplicación en /META_INF

El de la misma aplicación está en /usr/share/tomcat7-admin/host-manager/META-INF

```
<Context antiResourceLocking="false" privileged="true" >
  <!--
    Remove the comment markers from around the Valve below to limit access to
    the host-manager application to clients connecting from localhost
  -->
  <!--
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
  -->
</Context>
```

Entonces ¿para qué sirve context.xml? contiene los parámetros que vayan a ser comunes a todas las aplicaciones. Si no se sobrescriben en algún contexto más concreto se aplicarán éstos.

```
<Context>
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
</Context>
```

En este caso se especifica qué elemento de cada aplicación debe ser vigilado por si cambia. En ese caso se vuelve a desplegar la aplicación automáticamente.

web.xml

Cualquier aplicación web en Java debe tener un descriptor de despliegue. Como sucedía con el contexto, puede establecerse varios y se aplicarán las directivas más concretas.

Cada aplicación tiene su propio descriptor y va en /WEB-INF

Por ejemplo /usr/share/tomcat7-admin/host-manager/WEB-INF

Existe uno común a todas las aplicaciones en /etc/tomcat/web.xml

> Como el año pasado vimos xml es muy fácil entender un descriptor de despliegue. Abre el primero y estudia su contenido.
>
> El segundo archivo es muchísimo más extenso pero no por eso vamos a dejar de echarle un vistazo. ¿Qué tipo de parámetros se incluyen en este archivo? ¿Crees que es adecuada esta división?
>
> En ambos casos, si no entiendes para qué sirve un parámetro, internet puede ayudarte.

En el punto de despliegue de aplicaciones veremos algo más de este tema.

Administrar aplicaciones web

Vamos a administrar las aplicaciones de Tomcat a través del *Web Manager*. En las dos instalaciones que hemos hecho incluimos este paquete. En este caso cuando instalamos los paquetes adicionales.

Podemos acceder al administrador a través del enlace de la página de inicio o en la dirección http://localhost:8080/manager/html

Es en si misma una aplicación (con todas las características que estamos viendo) y de hecho puede administrarse desde la propia aplicación.

El acceso por defecto está deshabilitado, pero ya vimos cómo modificar el archivo tomcat-users.xml para poder entrar. Existen varios roles en caso de que queramos distribuir el trabajo, pero nos asignamos todos los permisos.

Nada más entrar en el manager vemos una lista de las aplicaciones desplegadas en este servidor.

/host-manager	*Ninguno especificado*	Tomcat Host Manager Application	true	1	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos

Vemos que aparece un enlace a la aplicación en sí, si se ha especificado una versión, el nombre que se mostrará para la aplicación, si está en ejecución, el número de sesiones que se han establecido (si la aplicación usa sesiones) y unos botones para arrancar, parar, recargar (la aplicación en caso de modificaciones) o replegar que lo que hace es eliminarla del servidor (no estará desplegada).

El apartado de las sesiones se verá más adelante.

Accede al web manager y prueba a parar los ejemplos y la documentación. ¿Puedes conectarte a ellos?

La parte de desplegar las aplicaciones la veremos en el apartado correspondiente.

Lo siguiente que nos encontramos es

Los diagnósticos sirven para ver si una aplicación está fallando, por ahora solo en el temas de uso de memoria. Si pinchamos se recarga la página y en la parte superior nos indica el resultado de la prueba.

También podemos ver un cuadro resumen con información del servidor. Esta información puede ampliarse al pinchar en “Estado Completo del Servidor” en la parte superior de la página.

Por supuesto esta administración puede ser mucho más profunda a través de los archivos de configuración vistos en este tema.

Estado de Servidor

Gestor			
Listar Aplicaciones	Ayuda HTML de Gestor	Ayuda de Gestor	Estado Completo de Servidor

Información de Servidor							
Versión de Tomcat	Versión JVM	Vendedor JVM	Nombre de SO	Versión de SO	Arquitectura de SO	NombreDeMáquina	Dirección IP
Apache Tomcat/7.0.26	1.7.0_09-b30	Oracle Corporation	Linux	3.2.0-31-generic-pae	i386	sergio-VirtualBox	127.0.1.1

JVM

Free memory: 12.98 MB Total memory: 24.23 MB Max memory: 123.75 MB

"http-bio-8080"

Max threads: 200 Current thread count: 10 Current thread busy: 1
Max processing time: 1154 ms Processing time: 3.876 s Request count: 71 Error count: 8 Bytes received: 0.00 MB Bytes sent: 0.44 MB

Stage	Time	B Sent	B Recv	Client	VHost	Request
R	?	?	?	?	?	?
S	13 ms	0 KB	0 KB	127.0.0.1	localhost	GET /manager/status?org.apache.catalina.filters.CSRF_NONCE=08754385D8DF3136E2F479C673CE02F4 HTTP/1.1

P: Parse and prepare request S: Service F: Finishing R: Ready K: Keepalive

Existen otros dos métodos de administración de Tomcat que dejamos como temas de ampliación por parte del propio alumno en caso de que lo crea conveniente: Ant y administración mediante peticiones HTTP.

La estructura de archivos y directorios de una Aplicación Web

Una aplicación web típicamente va dentro de un directorio (puede ser un WAR como veremos más adelante) y tiene asociada una estructura de directorios común a todas:

- Raíz del directorio de la aplicación y directorios no especificados en la siguiente lista: Todo lo que no esté en los directorios WEB-INF Y META-INF son recursos públicos a los que se puede acceder a partir de la URL adecuada.
- **WEB-INF**: a parte del descriptor de despliegue (web.xml) contiene:
- classes: donde se ubican los archivos .class y otros recursos.
- lib: donde se colocan las librerías .jar específicas de esta aplicación.
- **META-INF**: contiene el archivo de contexto context.xml. También puede hacer aquí un archivo MANIFEST.MF que lista las bibliotecas JAR que deben estar disponibles para el funcionamiento de la aplicación.

Despliegue de aplicaciones en el servidor de aplicaciones

Aunque en este módulo no se incluye cómo desarrollar un Servlet o un JSP, voy a dar unas pequeñas indicaciones. En este caso usaré Netbeans aunque Eclipse incluye las mismas funcionalidades y nos permite integrar desarrollo en muchos lenguajes.

Lo primero será instalar Netbeans desde la página oficial. Vemos que tenemos una opción en la que incluso están incluidos Tomcat y GlassFish. Esta es la recomendada para poder ir haciendo las pruebas. Ten en cuenta que Tomcat no se instala por defecto así que habrá que pulsar en personalizar durante la instalación (como indica la imagen) y seleccionarlo. Incluso aunque lo hagamos de prueba (por supuesto en un caso real la máquina de desarrollo y el servidor nunca deben ser el mismo) es preferible instalar Netbeans en la máquina anfitrión o en otra virtual ya que estamos activando otro servidor Tomcat.

Si ya tenemos Netbeans instalado (y un servidor de aplicaciones) y no queremos la versión de arriba, seguimos estas instrucciones para añadir y configurar el framework de red.

El tipo de proyecto que se debe crear es del grupo Java Web. Por defecto ya nos crea un JSP de tipo “Hola Mundo”. Un Servlet es una clase de java preparada para funcionar en la web. En este enlace explican brevemente cómo programar Servlets. Además recuerda que hemos instalado ejemplos en los que puedes consultar el código para ver qué hacen y cómo lo hacen.

Después de ejecutar o limpiar y contruir (Clean & Build) podremos ir a la carpeta del proyecto y en un subdirectorio “dist” habrá un archivo WAR que nos servirá para el despliegue.

En el site hay un **ejemplo 0310_JSPUno.war** que uso para estos ejemplos y puedes usar para tus prácticas. Cuando lo descargues recuerda que un WAR es un archivo empaquetado y que se puede descomprimir para obtener los archivos y carpetas originales.

Aunque no es el objetivo del curso vamos a ver el ejemplo del site y algunos de los instalados previamente. Los iremos estudiando **uno a uno** según diga el profesor y comentando en clase. Solo veremos los más sencillos. Se deja como ampliación para el alumno que lo desee ver el resto.

Despliegue manual

Exiten dos formas de llevar a cabo el despliegue manual de una aplicación en nuestro servidor de aplicaciones, pero son bastante similares. La primera es usando la carpeta con todos los elementos del proyecto y la segunda es con el archivo WAR.

Para **desplegar la carpeta** solo hay que copiarla en el directorio webapps de la ubicación $CATALINA_BASE/webapps que en nuestro caso es /var/lib/tomcat7/webapps

En el ejemplo he descomprimido los contenidos del WAR en una carpeta “Prueba” que es la que copio a webapps.

Recuerda que en nuestro fichero server.xml aparecía

```
<Host name="localhost"  appBase="webapps"  unpackWARs="true" autoDeploy="true">
```

Si la opción autoDeploy estuviera a false habría que reiniciar el servidor, pero en nuestro caso no hace falta así que podemos ir al navegador web y acceder a nuestra aplicación en la dirección http://localhost:8080/Prueba/

Desplegar el archivo WAR es igual pero ponemos el archivo directamente en la carpeta webapps. Ten en cuenta que ahora la ruta de la aplicación la determina el nombre del archivo WAR así que será http://localhost:8080/0310_JSPUno/

Estableciendo nuestra aplicación como la principal para el servidor

La solución anterior está bien para aplicaciones secundarias, pero ahora queremos que sea la principal en lugar de la página de inicio de Tomcat. Esto también se puede hacer de varias formas aunque lo primero sería eliminar el contenido de la carpeta ROOT de webapps. Si vas a hacerlo, ten en cuenta que en nuestro caso es mejor cambiar el nombre o mover el contenido a otro sitio por si luego queremos recuperar el estado original.

Luego las opciones serían:

1. Copiar el contenido de la carpeta del proyecto (no el proyecto en sí) a ROOT.
2. Cambiar el nombre al archivo WAR por ROOT.war y colocarlo en webapps para que lo despliegue Tomcat automáticamente en la carpeta ROOT
3. Crear un descriptor de contexto de la aplicación en la carpeta $CATALINA_BASE/conf/Catalina/localhost. Recuerda que esta ruta puede variar dependiendo de tu configuración, motor y host. Este descriptor se llamará ROOT.xml y en él habrá que escribir

```
<?xml version="1.0" encoding="UTF-8"?>
<Context docBase="/usr/share/Prueba" />
```

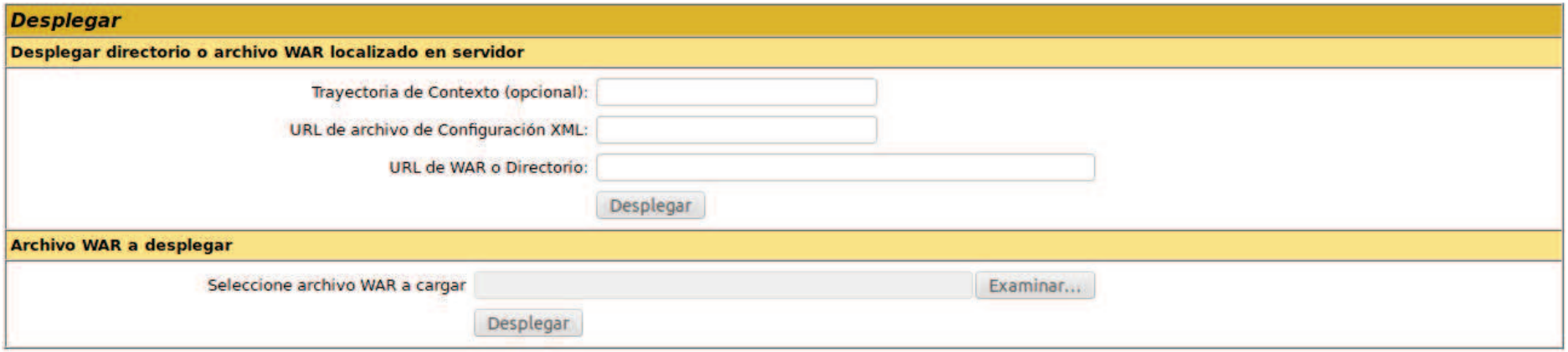
Suponiendo que hubiéramos ubicado nuestra aplicación en el directorio /usr/share/Prueba

Ten en cuenta que no hay que especificar path porque queremos que sea el que se cargue al acceder directamente a la dirección del servidor.

Si usamos esta opción reiniciamos Tomcat para que vuelva a cargar los archivos de configuración.

Despliegue con Tomcat Web Manager

Recuerda que cuando vimos el Web Manager nos saltamos una parte que permitía desplegar aplicaciones.



La primera opción es para desplegar una aplicación que está en otro servidor.

Solo vamos a ver la opción de abajo. Esta opción es equivalente colocar un WAR en webapps. Al pinchar el botón examinar tendremos que ubicar el WAR deseado y pulsamos sobre desplegar para que active la aplicación. Nos aparecerá en la lista de aplicaciones y podremos gestionarla como a cualquier otra. Si pulsamos en “Replegar” se eliminará.

Autenticación de usuarios

Los métodos para autenticar usuarios en Tomcat son los mismos que en Apache: Basic, Digest, Formularios HTML y Certificados Digitales. Los dos primeros ya vimos que tenían un problema si se aplicaban sobre conexiones no seguras. En el caso de formularios se establece una página (por ejemplo un JSP) que sirva de elemento donde se realizará la autenticación. En los ejemplos que hemos instalado se utiliza así que **después de ver este apartado** estarás preparado para entender cómo funciona.

Por ejemplo el descriptor de despliegue del web manager utiliza autenticación Basic y se le asigna un nombre al dominio.

```
<!-- Define the Login Configuration for this Application -->
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Tomcat Manager Application</realm-name>
</login-config>
```

En Tomcat cada usuario tiene asociado uno o más roles. Son estos roles los que se comprueban para conectarse y no el usuario en si. En el mismo archivo podemos ver la configuración de autenticación completa con todos los roles definidos. He eliminado gran parte del código dejando solo un ejemplo de cada tipo.

En el siguiente ejemplo es importante destacar que estamos declarando:

- <Security-constraint> una restricción de seguridad en la que se especifica un recurso o colección de recursos. En este caso todo lo que haya en el directorio *html/* y se le asigna un nombre. Además lleva una restricción de autorización donde se establecen uno o más roles que podrán acceder.
- <login-config> donde se especifica qué método se utilizará para llevar a cabo el control de acceso. Es el trozo que vimos antes.

- <security-role> por último se especifican los roles que se nombran en la seguridad de esta aplicación. Date cuenta de que deben estar definidos con el mismo nombre en el archivo, base de datos, etc que utilicemos para la autenticación.

```
<!-- Define a Security Constraint on this Application -->
<!-- NOTE:  None of these roles are present in the default users file -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>HTML Manager interface (for humans)</web-resource-name>
    <url-pattern>/html/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
     <role-name>manager-gui</role-name>
  </auth-constraint>
</security-constraint>

<!-- Define the Login Configuration for this Application -->
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>Tomcat Manager Application</realm-name>
</login-config>

<!-- Security roles referenced by this web application -->
<security-role>
  <description>
    The role that is required to access the HTML Manager pages
  </description>
  <role-name>manager-gui</role-name>
</security-role>
```

Sin embargo si nos vamos a ver la estructura de carpetas de la aplicación “manager” vemos que no hay un subdirectorio *html/* por ningún lado. Para entender esto hay que tener en cuenta otras dos partes del archivo. Nuevamente me centro en un ejemplo porque hay varios como este.

```
    <servlet-mapping>
      <servlet-name>HTMLManager</servlet-name>
      <url-pattern>/html/*</url-pattern>
    </servlet-mapping>
```

Aquí ya vemos el mismo patrón que estamos buscando. Un mapeado ya suena a una asociación. Se asocia un nombre con un patrón URL. Si nos conectamos al manager con el navegador web veremos que se añade este *html/* a las direcciones. Si te das cuenta, al pinchar en el enlace “Estado del Servidor” se cambia html por status. También encontramos un mapeo a este añadido a las URL y si nos damos cuenta tiene su propia configuración de seguridad y roles asociados.

```
    <servlet-mapping>
      <servlet-name>Status</servlet-name>
      <url-pattern>/status/*</url-pattern>
    </servlet-mapping>
```

¿Qué roles de seguridad tienen acceso a los recursos de status?

Pero todavía nos falta ver qué es ese nombre que se asocia a un patrón URL. Si miramos más arriba en el archivo

```
    <servlet>
      <servlet-name>HTMLManager</servlet-name>
      <servlet-class>org.apache.catalina.manager.HTMLManagerServlet</servlet-class>
      <init-param>
```

```
        <param-name>debug</param-name>
        <param-value>2</param-value>
      </init-param>
      <!-- Uncomment this to show proxy sessions from the Backup manager in the
           sessions list for an application
      <init-param>
        <param-name>showProxySessions</param-name>
        <param-value>true</param-value>
      </init-param>
      -->
      <multipart-config>
        <!-- 50MB max -->
        <max-file-size>52428800</max-file-size>
        <max-request-size>52428800</max-request-size>
        <file-size-threshold>0</file-size-threshold>
      </multipart-config>
    </servlet>
```

Donde lo más importante por ahora son las primeras líneas que asocian un nombre a una clase de Java. Podemos ver otros dos factores (sin entrar mucho más en detalle porque no es el objetivo de este punto) que nos ayudarán a comprender esto. En */usr/share/tomcat7-admin/manager/index.jsp* podemos ver lo siguiente:

```
<% response.sendRedirect(response.encodeRedirectURL(request.getContextPath() + "/html")); %>
```

Que incluso con los conocimientos que tenemos vemos que redirecciona las peticiones al mismo path pero con /html al final. La última duda es ¿dónde está el servlet al que se asocia el nombre HTMLManager? No aparece por la estructura de carpetas de la aplicación. Debemos recordar que hay un directorio de bibliotecas compartidas: */usr/share/tomcat7/lib* y en él hay un archivo que se llama catalina.jar (como parte del nombre completo de la clase). Realmente es un enlace simbólico al archivo *../../java/tomcat-catalina-7.0.26.jar* o lo que es lo mismo */usr/share/../../java/tomcat-catalina-7.0.26.jar*

Si examinamos este archivo encontraremos el dichoso servlet.

Todo este funcionamiento se basa en el elemento <Realm> del que ya hemos hablado. Recuerda que podía ir dentro de los elementos <Engine>, <Host>, <Context> o <Cluster> y como en otros casos esto indicará a qué se aplica. Posteriormente veremos como configurar diferentes *Realms*, pero por ahora nos vale con saber que en el caso por defecto se basa en el archivo *tomcat-users.xml*

Ejemplos de autenticación

Vamos a ver algunos ejemplos de autenticación de los métodos mencionados. El caso de certificados digitales evidentemente no tendrá sentido hasta que veamos la integración con SSL. Ya conocemos los problemas de seguridad de HTTP.

Ya hemos visto como desplegar aplicaciones hechas con Netbeans (con Eclipse es muy similar), por lo que voy a realizar y probar los ejemplos directamente en Netbeans. Para modificar el archivo tomcat-users.xml hay que buscarlo primero:

1. Selecciona la pestaña “Services”
2. En “Servers” hac clic con el botón derecho sobre “Properties”.
3. Comprueba que es Tomcat y no GlassFish es que está seleccionado.
4. Copia la ruta de CATALINA_BASE
5. Cierra la ventana.
6. Haz clic en los menús en File > Open File y vete a la ruta de CATALINA_BASE y al directorio “conf” que hay dentro. Selecciona el archivo tomcat-users.xml
7. Ahora puedes modificar el archivo que se usará. Recuerda que esto tienes que volver a hacerlo en el archivo tomcat-users.xml en el servidor donde despliegues las aplicaciones posteriormente. Este servidor es solo para pruebas.
8. En mi caso voy a dejarlo como se ve a continuación.

```
    <user username="ide" password="AyO2ybUt" roles="manager-script,admin"/>
      <role rolename="pruebas"/>
```

```
      <role rolename="otro"/>
      <user username="sergio" password="sergio" roles="pruebas"/>
      <user username="cuesta" password="cuesta" roles="pruebas,otro"/>
      <user username="vicente" password="vicente" roles="pruebas"/>
    </tomcat-users>
```

Ten en cuenta que si el servidor ya está iniciado debes reiniciarlo para que cargue la configuración. Para esto haz clic con el botón derecho sobre el servidor Tomcat y pulsa “Restart” o la opción que necesites en cada caso.

BASIC

Primero vamos a crear el servlet, se hace como cualquier otra clase de Java pero seleccionando Servlet como tipo y luego tienes que marcar la casilla que añade información al web.xml

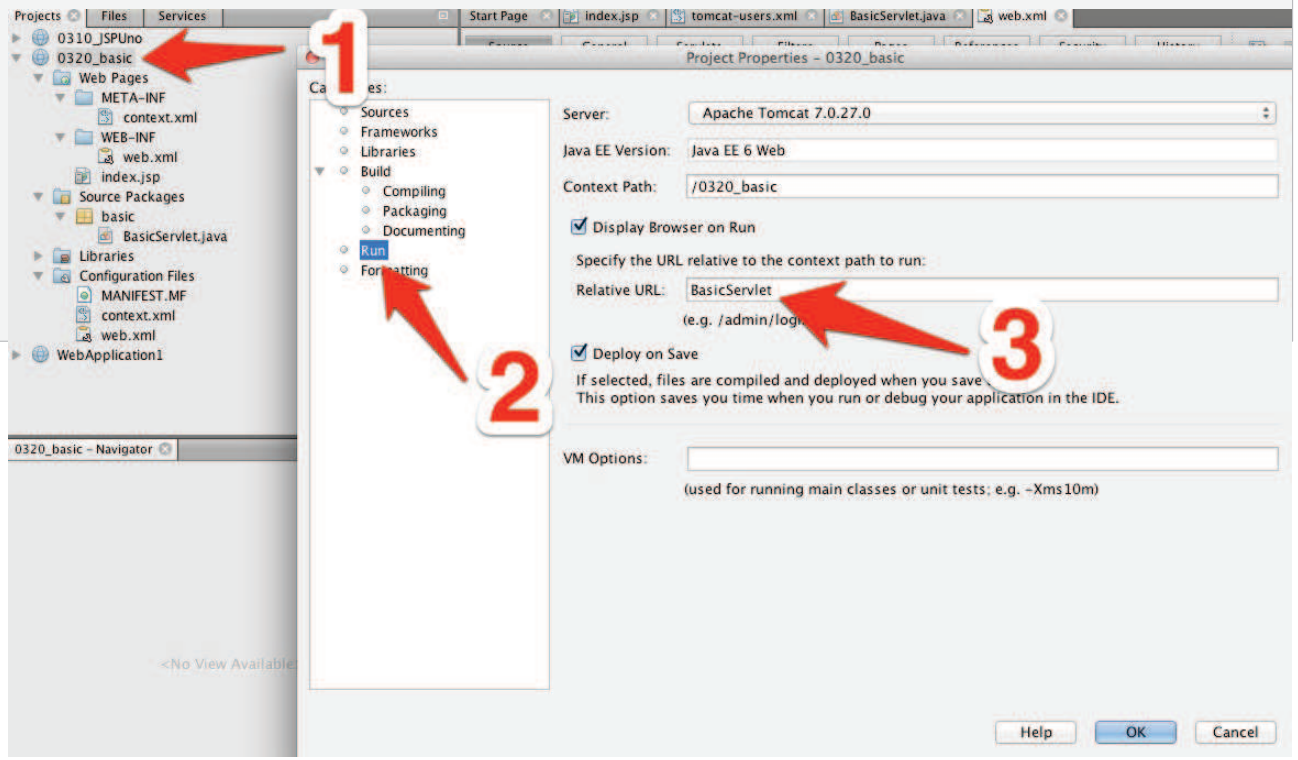
```
    <?xml version="1.0" encoding="UTF-8"?>
    <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                              xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
        <servlet>
            <servlet-name>BasicServlet</servlet-name>
            <servlet-class>basic.BasicServlet</servlet-class>
        </servlet>
        <servlet-mapping>
            <servlet-name>BasicServlet</servlet-name>
            <url-pattern>/BasicServlet</url-pattern>
        </servlet-mapping>
        <session-config>
            <session-timeout>
                30
```

```
            </session-timeout>
        </session-config>
    </web-app>
```

Debemos modificar esto para que se corresponda con lo que pone a continuación:

```
    <?xml version="1.0" encoding="UTF-8"?>
    <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                                xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
        <servlet>
            <servlet-name>BasicServlet</servlet-name>
            <servlet-class>basic.BasicServlet</servlet-class>
        </servlet>
        <servlet-mapping>
            <servlet-name>BasicServlet</servlet-name>
            <url-pattern>/BasicServlet</url-pattern>
        </servlet-mapping>

        <!-- esto es lo que añadimos para la seguridad básica -->
        <!-- Primero una restricción de seguridad -->
        <security-constraint>
                <web-resource-collection>
                            <web-resource-name>El * significa que pedimos sutenticación para toda la
aplicación</web-resource-name>
                        <url-pattern>/*</url-pattern>
                        <http-method>GET</http-method>
                        <http-method>POST</http-method>
                </web-resource-collection>
```

```
                <auth-constraint>
                        <role-name>pruebas</role-name>
                </auth-constraint>

                <user-data-constraint>
                        <!-- hay tres tipos CONFIDENTIAL, INTEGRAL y NONE -->
                        <transport-guarantee>NONE</transport-guarantee>
                </user-data-constraint>
        </security-constraint>

        <login-config>
                <auth-method>BASIC</auth-method>
        </login-config>
        <!-- hasta aquí -->

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



Con esto (ten en cuenta que nos ha creado un servlet) ya tendríamos la autenticación básica funcionando, pero podemos modificar el servlet para que nos proporcione cierta información.

<transport-guarantee> tiene que ser NONE, pero cuando habilitemos SSL deberíamos cambiarlo a CONDIFENTIAL para que nos rediriga las peticiones HTTP a HTTPS y se transmita la información de esa manera.

Si queremos ejecutar este servlet por defecto en Netbeans debemos hacer:

1. Clic con el botón derecho sobre el proyecto y seleccionar “Properties”
2. Seleccionar “Run”
3. Escribir el nombre del Servlet (sin extensión) en “Relative URL”

Ten en cuenta que si lo desplegamos en nuestro servidor Tomcat de producción la URL sería la correspondiente a la aplicación más el nombre del servlet.

p.ej: http://localhost:8080/nom-aplicación/nom-servlet

Puedes ver todo esto y el Servlet modificado que proporciona información sobre la conexión en el **ejemplo 0320_basic.war** del site.

Aunque no es el objetivo del módulo, este tutorial de Netbeans puede serte de utilidad.

DIGEST

La única diferencia entre el uso de BASIC y DIGEST consiste en cambiar esa palabra en web.xml. La única diferencia en el funcionamiento es que la contraseña se codifica, pero no encripta por lo que si alguien la intercepta es prácticamente igual de vulnerable.

> Modifica el ejemplo 0320_basic.war para que use DIGEST y comprueba la diferencia en el resultado.

FORM

Este método se basa en formularios HTML.

Lo primero que cambia es en el archivo web.xml el elemento <login-config>

```
        <login-config>
            <auth-method>FORM</auth-method>
            <!-- CUIDADO, los archivos siguientes deben ir en un subdirectorio para que funcione-->
            <form-login-config>
                <form-login-page>/protegido/login.jsp</form-login-page>
                <form-error-page>/protegido/login-failed.html</form-error-page>
            </form-login-config>
        </login-config>
```

Los dos archivos HTML indican en qué página se hará el log-in y a cuál nos redirigirá si falla el intento.

Todo esto también se puede configurar en Netbeans pinchando en la pestaña “Security” cuando estamos viendo el archivo web.xml

Este método se basa en el uso de tres elementos:

1. j_security_check: que procesa el formulario.
2. j_username: como elemento donde se enviará el nombre de usuario.
3. j_password: para enviar la contraseña.

Puedes estudiar el **ejemplo 0330_form** del site para ver todos estos elementos.

Para probar debes intentar acceder al elemento FormServlet, a través de la ruta si lo tienes en un Tomcat o configurando Netbeans si lo estás probando con el servidor integrado.

El ejemplo usa sesiones además por lo que puedes ver un uso totalmente básico de ello.

Dominios de seguridad para la autenticación

Un dominio (Realm) es algún tipo de asociación entre usuarios, contraseñas y los roles asociados. Los roles son algo similar a los grupos de Linux y en Tomcat el acceso no se proporciona a nivel de usuario si no de rol. Un usuario puede tener asociado un número ilimitado de roles.

Existen seis tipos básicos en Tomcat aunque hay otros extendidos y cualquiera puede programar uno que extienda de estos:

1. **JDBCRealm**: La información de usuarios, contraseñas y roles se almacena en una base de datos relacional y se accede mediante un driver JDBC.
2. **DataSourceRealm**: La información también se guarda en una base de datos relacional, pero se accede a ella mediante una fuente con nombre de tipo JNDI JDBC DataSource. JNDI (Java Naming and Directory Interface) es un servicio de Java que permite buscar objetos y datos por nombre.
3. **JNDIRealm**: La información se almacena en un servidor LDAP y se accede con JNDI.
4. **UserDatabaseRealm**: La información se almacena en lo que se conoce como una fuente JNDI de Base de Datos de Usuarios. Lo más habitual es que se apoye en un archivo xml. Por ejemplo de *conf/tomcat-users.xml* aunque no es esta la opción que se usa por defecto sino la siguiente. La información puede actualizarse dinámicamente, pero no se recomienda para grandes sistemas.
5. **MemoryRealm**: La información está almacenada en un archivo XML que se carga en memoria al iniciar Tomcat. Si se modifica el archivo la información no se usa hasta que se reinicie Tomcat. Es el caso que venimos usando con conf/tomcat-users.xml. Por supuesto no se recomienda en producción.
6. **JAASRealm**: Se utiliza el framework Java Authentication & Authorization Service (JAAS). Es el más sofisticado de todos.

Nosotros solo veremos algunos. Ya he comentado que hay otros que extienden de estos. Toda esta información y mucha más se puede consultar en la página de la documentación de Tomcat.

Solo puede haber un Realm funcionando a la vez, y por lo tanto si pruebas los ejemplos anteriores tras configurar Realms diferentes verás que usa para conectarse el Realm activo.

MemoryRealm

Es el caso que hemos estado usando por defecto al conectarnos por ejemplo al manager de aplicaciones web. Ya hemos visto que la información se almacena en un fichero XML y que por defecto es *tomcat-users.xml*

Al iniciar el servidor se carga la información de usuarios del archivo en memoria y no se vuelve a actualizar hasta que se reinicie el servidor. Por este motivo y por la baja seguridad que supone tener los usuarios y contraseñas en un fichero de texto plano.

Es el caso que vimos como ejemplo en el punto anterior.

Esto se concreta en los puntos:

1. Gestionar los usuarios en el archivo XML correspondiente.
2. Crear el dominio. Para ello en el archivo de contexto que determinemos (por ejemplo en el que se encuentre en el directorio META-INF específico de cada aplicación) escribiremos lo siguiente:

```
<Context>
    <Realm className="org.apache.catalina.realm.MemoryRealm" />
</Context>
```

   He dejado las etiquetas del contexto para evidenciar que va dentro.

3. Crear las restricciones de seguridad del recurso en el descriptor de despliegue correspondiente. Esto se hace con el elemento <security-constraint>
4. Determinar el tipo de acceso al recurso. <login-config>
5. Establecer uno o más roles que puedan acceder al elemento. <security-role>

UserDatabaseRealm

Este tipo es una pequeña modificación del anterior que nos permite simular que estamos realizando la autenticación con una base de datos. La única diferencia con el anterior es que se realiza a través de un recurso JNDI.

IMPORTANTE: Realmente "mentí" en el punto anterior porque el caso que se utiliza en el servidor de Tomcat según lo hemos instalado es éste. Pero al ser una extensión del anterior he preferido separar la explicación en dos partes.

Para verlo tenemos que ir al archivo server.xml

```
      <!-- Global JNDI resources
           Documentation at /docs/jndi-resources-howto.html
      -->
      <GlobalNamingResources>
        <!-- Editable user database that can also be used by
             UserDatabaseRealm to authenticate users
        -->
        <Resource name="UserDatabase" auth="Container"
                  type="org.apache.catalina.UserDatabase"
                  description="User database that can be updated and saved"
                  factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
                  pathname="conf/tomcat-users.xml" />
      </GlobalNamingResources>
```

Si nos damos cuenta se define un recurso global de tipo *UserDatabase* que se asocia al archivo *tomcat-users.xml*

Además un poco más abajo

```
        <!-- Use the LockOutRealm to prevent attempts to guess user passwords
             via a brute-force attack -->
```

```
<Realm className="org.apache.catalina.realm.LockOutRealm">
  <!-- This Realm uses the UserDatabase configured in the global JNDI
       resources under the key "UserDatabase".  Any edits
       that are performed against this UserDatabase are immediately
       available for use by the Realm.  -->
  <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
         resourceName="UserDatabase"/>
</Realm>
```

Vemos que se define el *Realm* (mira el interno primero) que es justo del tipo que estamos comentando en este punto. El *Realm* exterior *LockOutRealm* parece interesante según lo que explican los comentarios ¿no? Es un contenedor para asegurar el *Realm* que utilicemos y se puede ver en la página de la documentación de Tomcat.

Por lo tanto a los puntos de MemoryRealm habría que añadir (el primero se añade y el segundo es una modificación):

1. Configurar el recurso JDNI
2. Crear el <Realm> para que sea de tipo UserDatabaseRealm en lugar de MemoryRealm

Utilizando MemoryRealm, establece un tipo de control de acceso básico a una aplicación que despliegues en tu servidor puedes usar el ejemplo del site. Establece un rol cuyo identificador sea tu apellido y al menos un usuario con tu nombre. Prueba el acceso.

Realmente solo puede haber un <Realm> actuando en una aplicación en un momento dado. Esta afirmación implica (debes elegir una de las dos respuestas después de probar):

1. Se utiliza el Realm más concreto como con las directivas de Apache.

2 .Hay que eliminar el Realm de server.xml porque si no…

Elimina el control de acceso anterior y realiza lo mismo con UserDatabaseRealm ¿Puedes establecer el control de acceso solo para alguna parte de la documentación? ¿y para un archivo en particular? Al ver la estructura de carpetas de una Aplicación Web decíamos que todo lo que no se ubicara en WEB-INF o en META-INF era público. ¿Afecta a esto?

Crea otro rol en el archivo de usuarios pero no le des acceso a la aplicación de documentación. Comprueba que no tiene acceso pero el otro rol sí.

Recuerda que debes reiniciar el servidor (Tomcat) tras cada cambio.

JDBCRealm

Este caso necesita una base de datos para funcionar por lo que empezaremos instalando MySQL. Los usuarios ahora se ubicarán en una base de datos con tablas de usuarios, roles y la relación entre ellos. Los nombres de estos elementos se pueden elegir y configurar pero hay unos prácticamente estándar que serán los que usemos.

Tomcat se conecta a bases de datos mediante una tecnología llamada *Java Database Connectivity* (JDBC). JDBC funciona como una capa de abstracción entre las bases de datos y los elementos que la vayan a usar. De esta forma Tomcat se comunica con la API de JDBC en lugar de con la base de datos directamente. Así se consigue no depender de las peculiaridades de cada base de datos. JDBC se comunica con la base de datos a través de un driver que convierte las órdenes directamente al formato específico de cada base de datos.

Ya tenemos instalado MySQL y JDBC que se incluye en Java por lo que el siguiente paso será instalar el driver. Para ello podemos descargar el driver desde su web y colocarlo en $CATALINA_HOME/lib que en nuestro caso está en */usr/share/tomcat7/lib* lo que lo hará disponible para todas las aplicaciones de Tomcat o colocarlo en el directorio WEB-INF/lib de una aplicación específica para que esté disponible en ella.

Como alternativa podemos instalarlo para que esté disponible para cualquier aplicación que use la JVM de Java:

```
sudo apt-get install libmysql-java
```

y establecemos el CLASSPATH

```
CLASSPATH=".:/usr/share/java/mysql.jar"
```

Este último método da problemas porque el uso de classpath ya no está recomendado.

Nosotros usaremos el método de colocarlo en las librerías comunes de Tomcat.

Tras lo que habrá que reiniciar para que se cargue.

Ahora, tendremos que crear la configuración correcta en MySQL para la gestión de usuarios y roles. Para ello usaremos el siguiente script de ejemplo

```
DROP DATABASE IF EXISTS tomcat_realm;
CREATE DATABASE tomcat_realm;
USE tomcat_realm;
CREATE TABLE tomcat_users (
    user_name varchar(20) NOT NULL PRIMARY KEY,
```

```
          password varchar(32) NOT NULL
     );
     CREATE TABLE tomcat_roles (
          role_name varchar(20) NOT NULL PRIMARY KEY
     );
     CREATE TABLE tomcat_users_roles (
          user_name varchar(20) NOT NULL,
          role_name varchar(20) NOT NULL,
          PRIMARY KEY (user_name, role_name),
          CONSTRAINT tomcat_users_roles_foreign_key_1 FOREIGN KEY (user_name) REFERENCES tomcat_users
(user_name),
          CONSTRAINT tomcat_users_roles_foreign_key_2 FOREIGN KEY (role_name) REFERENCES tomcat_roles
(role_name)
     );
     INSERT INTO tomcat_users (user_name, password) VALUES ('sergio', 'sergio');
     INSERT INTO tomcat_users (user_name, password) VALUES ('maria', 'maria');
     INSERT INTO tomcat_roles (role_name) VALUES ('pruebas');
     INSERT INTO tomcat_roles (role_name) VALUES ('otro');
     INSERT INTO tomcat_users_roles (user_name, role_name) VALUES ('sergio', 'pruebas');
     INSERT INTO tomcat_users_roles (user_name, role_name) VALUES ('sergio', 'otro');
     INSERT INTO tomcat_users_roles (user_name, role_name) VALUES ('maria', 'pruebas');
     COMMIT;
```

La tabla que lista los roles no es necesaria y las claves ajenas tampoco, pero las hemos creado por mantener la corrección.

En mi caso voy a crear este script en un directorio para tenerlo accesible en todo momento. Llamo al script *tomcatusers.sql*

```
     mkdir /usr/db-scripts
     cd /usr/db-scripts/
```

```
gedit tomcatusers.sql
```

Accedemos a MySQL

```
mysql -u root -p
```

dentro de MySQL ejecutamos el script

```
mysql> source /usr/db-scripts/tomcatusers.sql
```

y comprobamos que todo haya ido bien.

```
mysql> select * from tomcat_users;
mysql> select * from tomcat_users_roles;
quit
```

Ahora creamos un usuario para que se conecte el Realm que creemos en Tomcat. Creo un script que se llame *usuarioRealm.sql*

```
USE mysql;
CREATE USER 'accesoRealm'@'localhost' IDENTIFIED BY 'pwdRealm';
GRANT SELECT ON tomcat_realm.* TO accesoRealm@localhost;
```

Y lo ejecuto

```
mysql> source /usr/db-scripts/usuarioRealm.sql
```

En el archivo server.xml modifico la sección del Realm comentando el que hay e introduciendo el de la base de datos.

```
        <!-- Use the LockOutRealm to prevent attempts to guess user passwords
             via a brute-force attack -->
        <Realm className="org.apache.catalina.realm.LockOutRealm">
          <!-- This Realm uses the UserDatabase configured in the global JNDI
               resources under the key "UserDatabase".  Any edits
               that are performed against this UserDatabase are immediately
               available for use by the Realm.  -->
      <!--
          <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
                 resourceName="UserDatabase"/>
      -->
        <Realm  className="org.apache.catalina.realm.JDBCRealm"
             driverName="com.mysql.jdbc.Driver"
                 connectionURL="jdbc:mysql://localhost:3306/tomcat_realm"
            connectionName="accesoRealm" connectionPassword="pwdRealm"
             userTable="tomcat_users" userNameCol="user_name" userCredCol="password"
                 userRoleTable="tomcat_users_roles" roleNameCol="role_name" />
        </Realm>
```

> Estudia los parámetros de configuración del Realm y discútelos en clase.

Tras esto reiniciamos Tomcat y si no recibimos ningún error hemos configurado el Realm bien.

```
/etc/init.d/tomcat7 restart
```

Ten en cuenta que el error puede estar en los ficheros de log de Tomcat. Aunque luego veremos más sobre esto, tienes que mirar en el directorio */var/lib/tomcat7/logs* y buscar el archivo con la fecha de hoy de los que tengan un nombre como *catalina.2013-01-26.log* ten en cuenta que puedes borrar el contenido del log para encontrar más fácilmente los nuevos errores.

El error más habitual es que no encuentre el driver JDBC

```
ene 26, 2013 8:47:38 AM org.apache.catalina.realm.JDBCRealm authenticate SEVERE: Excepción al realizar
la autentificación java.sql.SQLException: com.mysql.jdbc.Driver     at
org.apache.catalina.realm.JDBCRealm.open(JDBCRealm.java:701)   at
org.apache.catalina.realm.JDBCRealm.authenticate(JDBCRealm.java:352)      at
org.apache.catalina.realm.CombinedRealm.authenticate(CombinedRealm.java:146)   at
org.apache.catalina.realm.LockOutRealm.authenticate(LockOutRealm.java:180)     at
org.apache.catalina.authenticator.FormAuthenticator.authenticate(FormAuthenticator.java:295)   at
org.apache.catalina.authenticator.AuthenticatorBase.invoke(AuthenticatorBase.java:450)    at
org.apache.catalina.core.StandardHostValve.invoke(StandardHostValve.java:168)  at
org.apache.catalina.valves.ErrorReportValve.invoke(ErrorReportValve.java:98)   at
org.apache.catalina.valves.AccessLogValve.invoke(AccessLogValve.java:927)      at
org.apache.catalina.core.StandardEngineValve.invoke(StandardEngineValve.java:118)    at
org.apache.catalina.connector.CoyoteAdapter.service(CoyoteAdapter.java:407)    at
org.apache.coyote.http11.AbstractHttp11Processor.process(AbstractHttp11Processor.java:987)     at
org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process(AbstractProtocol.java:579)      at
org.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run(JIoEndpoint.java:309)     at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1110)      at
```

```
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:603)     at
java.lang.Thread.run(Thread.java:722)
Caused by: java.lang.ClassNotFoundException: com.mysql.jdbc.Driver   at
java.net.URLClassLoader$1.run(URLClassLoader.java:366)    at
java.net.URLClassLoader$1.run(URLClassLoader.java:355)    at
java.security.AccessController.doPrivileged(Native Method)     at
java.net.URLClassLoader.findClass(URLClassLoader.java:354)     at
java.lang.ClassLoader.loadClass(ClassLoader.java:423)     at
java.lang.ClassLoader.loadClass(ClassLoader.java:356)     at  java.lang.Class.forName0(Native  Method)
    at java.lang.Class.forName(Class.java:186)      at
org.apache.catalina.realm.JDBCRealm.open(JDBCRealm.java:697)   ... 16 more
```

Que se produce porque no has colocado bien el archivo JAR con el driver o no está funcionando bien el CLASSPATH.

Prueba a acceder al web-manager. No puedes porque en la base de datos no hemos asignado el rol correcto al usuario. Para ello he creado otro script *manager.sql*

```
USE tomcat_realm;
INSERT INTO tomcat_roles (role_name) VALUES ('manager-gui');
INSERT INTO tomcat_users_roles (user_name, role_name) VALUES ('sergio', 'manager-gui');
COMMIT;
```

Prueba a desplegar aplicaciones con control de acceso en el servidor (te pueden servir algunos de los ejemplos del site) y prueba a conectarte con UserDatabaseRealm. Después configura JDBCRealm y vuelve a probar.

Administración de sesiones. Sesiones persistentes

Tomcat utiliza un Persistent Session Manager para realizar copias de las sesiones de los usuarios en el disco. Esta gestión de sesiones es básica para el funcionamiento de las aplicaciones. Permite realizar varias mejoras sobre las sesiones siendo las principales, el establecimiento de un tiempo máximo de vida para las sesiones inactivas y guardando las sesiones en disco cuando se apaga Tomcat, lo que permite que las sesiones se mantengan durante varias ejecuciones del servidor.

Hay definida un Intefaz org.apache.catalina.Manager para gestionar las sesiones. Se configura mediante el elemento <Manager> en alguno de los archivos de contexto (el global context.xml o el mismo en el directorio META-INF de alguna aplicación). Tomcat incorpora dos implementaciones de esta interfaz:

- org.apache.catalina.session.StandardManager: que es la implementación que se usa por defecto si no se especifica otra.
- org.apache.catalina.session.PersistenManager: que permite guardar las sesiones en una base de datos aparte de en un disco.

No vamos a entrar en muchos detalles sobre estas clases pero se pueden consultar en la documentación de Tomcat.

Si vamos al archivo de contexto

```
gedit /etc/tomcat7/context.xml
```

podemos ver que por defecto viene activado

```
<!-- Uncomment this to disable session persistence across Tomcat restarts -->
<!--
<Manager pathname="" />
```

```
        -->
```

Vemos que si lo descomentamos NO tendríamos sesiones persistentes aunque reiniciáramos Tomcat.

Podemos conectarnos al web manager de Tomcat http://localhost:8080/manager/html y probar a acceder al http://localhost:8080/0330_form/FormServlet

Podemos ver que si abrimos y cerramos el navegador varias veces se reinicia la sesión pero si tenemos abierto el navegador se mantiene para cuando volvamos. Cada vez que cerramos el navegador se suma una sesión al número de sesiones de la lista.

Si hacemos clic sobre el número de sesiones podremos ir a la pantalla de gestión

Donde tenemos varias opciones:

- Pinchar en el nombre de una columna para que ser ordenen las sesiones por ese criterio.
- Refrescar la lista. Mientras estamos gestionando esto pueden estar cambiando las sesiones por muchos motivos, por ejemplo un usuario se conecta o desconecta.
- Seleccionar una o más sesiones para invalidarlas posteriormente con el botón correspondiente. Esta terminará la sesión y el usuario tendrá que volver a crearla (por ejemplo autenticándose en la aplicación).
- Pinchar en el identificador de la sesión para ver sus detalles.

Details for Session A207B5B451DBC3DF3FB7794089432E64

Session Id	A207B5B451DBC3DF3FB7794089432E64
Guessed Locale	
Guessed User	maria
Creation Time	2013-01-26 10:14:02
Last Accessed Time	2013-01-26 10:14:05
Session Max Inactive Interval	00:30:00
Used Time	00:00:03
Inactive Time	00:13:32
TTL	00:16:27

Refresh

0 ATTRIBUTES

Remove Attribute	Attribute name	Attribute value

Return to session list

Podemos comprobar que aunque reiniciemos Tomcat o incluso la máquina virtual las sesiones se mantienen. Si miramos el descriptor de despliegue de una aplicación podemos ver que es posible configurar el tiempo que se mantendrán las sesiones inactivas. Si tuviéramos aplicaciones que requieren mucha seguridad conviene bajarlo para que cuando el usuario se ausente se termine la conexión.

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

Si miramos la lista de aplicaciones del manager vemos que podemos cerrar las sesiones que lleven inactivas más de un tiempo que pongamos. Para ello se escribe el tiempo y se pulsa el botón. Por supuesto solo tiene sentido en caso de que el tiempo que escribamos sea menor que el establecido por defecto para la aplicación.

Por defecto se utiliza el *StandardManager* que guarda las sesiones en un archivo *SESSIONS.ser* que se ubica en el directorio *work* de cada aplicación. Estos archivos se crean al detener Tomcat y se destruyen cuando se reinicia por lo que si quieres verlos tienes que parar Tomcat. Ten en cuenta que si Tomcat no se apaga correctamente (por ejemplo por un fallo en la corriente) no se guardan las sesiones.

Para configurar más aspectos de las sesiones usaríamos el elemento <Manager> de los archivos de contexto

CUIDADO: en el siguiente ejemplo, la clase indicada en realidad es la interfaz que tienen que implementar las clases para poder usarse en el elemento manager. Debería ser una clase concreta, por ejemplo una de las dos que pone al principio del punto. Ten en cuenta que el paquete de la clase tampoco es el mismo para la interfaz y las clases concretas.

```
<?xml version="1.0" encoding="UTF-8"?>
 <Context antiJARLocking="true" path="/0330_form">
    <Manager
         className = "org.apache.catalina.Manager"
         distributable = "true"
         maxActiveSessions = "25"
         maxInactiveInterval = "1200"
         sessionIdLength = "16"
    />
 </Context>
```

Estudia los atributos comunes a todas las implementaciones de Manager en la documentación. Coméntalos en clase.

¿Qué hace el atributo path de *StandardManager*? ¿Qué valor le darías para deshabilitar las sesiones persistentes?

Archivos de registro de acceso y filtro de solicitudes

Como cualquier servidor, en Tomcat es muy importante registrar algunos tipos de actividad. En el directorio */etc/tomcat7* podemos ver dos archivos que hacen referencia a estos registros. El primero es *logging.properties* y el segundo *server.xml*.

Ambos hacen referencia al directorio $CATALINA_BASE/logs que en nuestra instalación se encuentra como enlace simbólico en /var(lib/tomcat7/logs

Es una buena práctica revisar y borrar los archivos de registro periódicamente ya que pueden crecer mucho y acabar afectando al rendimiento del servidor.

En ese directorio podemos ver cuatro tipos de archivos de registro:

1. catalina.fecha.log: Estos archivos guardan la actividad del motor durante un día determinado.
2. catalina.out: es un compendio de los anteriores.
3. localhost.fecha.log: guardan la actividad del sitio.
4. localhost_access_log.fecha.txt: son registros de acceso a las aplicaciones del servidor durante un día. Son los que trataremos aquí.

Este último tipo se aparece configurado en el archivo server.xml dentro del elemento <Host>

```
            <!-- Access log processes all example.
                 Documentation at: /docs/config/valve.html
                 Note: The pattern used is equivalent to using pattern="common" -->
            <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                   prefix="localhost_access_log." suffix=".txt"
                   pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
```

y como podemos ver nos refiere a una página de la documentación que hemos instalado pero que también podemos ver en la página de Apache.

Tenemos que darnos cuenta de que realmente va asociado a un elemento <Valve> y muy relacionado con él hay otro que se llama <Filter>

Estas dos tecnologías sirven para interceptar las peticiones y respuestas de HTTP y pre procesarlas para realizar algún tipo de acción antes de que sigan su camino. Una gran ventaja es que no dependen de cada aplicación sino que pueden implementarse para todo el sitio. Una gran diferencia entre ellas es que <Valve> es un desarrollo asociado a Tomcat mientras que <Filter> pertenece a la API de Servlets.

Como ya vimos se pueden colocar en <Engine>, <Host> o <Context> afectando al entorno concreto según corresponda.

Válvulas

Se puede consultar su uso en la página correspondiente de la configuración, pero se puede apreciar que es muy extensa.

Uno puede programar su propia válvula o filtro partiendo de la interfaz Valve o de alguna de las clases del paquete valves. *ValveBase* está espacialmente indicada para ello. De hecho muchas de las otras clases del paquete son *final* por lo que no podremos heredar de ellas. Esto queda totalmente fuera del contenido del curso por lo que nos centraremos en ver algunas de las válvulas incluidas con Tomcat.

Vamos a empezar volviendo a la que está habilitada en server.xml

```
            <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                   prefix="localhost_access_log." suffix=".txt"
                   pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />
```

Podemos ver que usa *AccessLogValve* que como su nombre indica registra el acceso. Como está incluida en el elemento <Host> registra todos los accesos a localhost.

Consulta la documentación para hasta entender que hace esta configuración de la válvula.

Estudia el resto de válvulas (las que no acaban en Filter) solo para poder tener una idea de qué hacen. No es necesario que consultes los atributos ni la configuración.

Tras ver las válvulas incluidas, la utilidad de ellas es evidente. Una de ellas es especialmente curiosa y útil en determinados entornos. Si por ejemplo estamos en una intranet de una empresa donde hay varias aplicaciones web disponibles para los empleados, SingleSignOn nos permite que solo tengan que autenticarse en la primera que se conecten y se mantendrán esas credenciales a través de todas las aplicaciones.

Prueba la válvula SingleSignOn en tu sitio.

Crea un usuario con un rol que tenga acceso a unas aplicaciones pero no a otras. Conéctate con dicho usuario a una aplicación para la que sí tenga acceso. Luego ve a una en la que no tenga acceso. ¿Qué sucede?

Filtros

Los filtros son una interfaz de los Servlets. Por lo tanto no son exclusivos de Tomcat. Tienen un comportamiento muy similar a las válvulas pero se configuran en el descriptor de despliegue de cada aplicación. Aunque podemos implementar nuestros propios filtros que implementen la interfaz, Tomcat incluye algunos que pueden consultarse en la página de la documentación correspondiente.

Remote Address Filter, Remote Host Filter y Remote IP Filter/Valve pueden usarse tanto como válvulas como filtros. Comprueba que la documentación en ambos apartados es prácticamente igual. Echa un vistazo para entender qué hace cada uno.

Si miras (por encima) la documentación de los demás filtros verás que tienen usos mucho más concretos que las válvulas, muchos de ellos relacionados con ataques o situaciones muy específicas.

Configurar el servidor de aplicaciones para cooperar con servidores web

Aunque Tomcat puede gestionar peticiones HTTP y servir archivos HTML esto lo hace mejor Apache (u otro servidor web). Además Apache incluye muchas más opciones de configuración y dispone de muchísimos módulos para realizar diferentes tareas. Otros motivos pueden ser el rendimiento, la distribución en varias máquinas, etc.

En este punto vamos a aprender a configurar un servidor web que reciba las peticiones y se las reenvíe a Tomcat si es necesario. El contenido estático (o derivado de otros lenguajes como PHP si tenemos Apache configurado para ello) lo servirá Apache y las peticiones relacionadas con JSPs o Servlets se las redirigirá a Tomcat.

Como ya hemos comentado la conexión con el exterior se realiza a través de conectores. Los dos principales son el HTTP y el AJP; ambos soportan SSL.

El conector HTTP viene activado por defecto y es el que permite a Tomcat procesar peticiones HTTP y funcionar como un servidor web independiente. Si consultamos el archivo *server.xml* veremos.

```
        <Connector port="8080" protocol="HTTP/1.1"
                   connectionTimeout="20000"
                   URIEncoding="UTF-8"
                   redirectPort="8443" />
```

Podemos ver que las opciones de configuración son muchas más en la ayuda de configuración de Tomcat. No vamos a entrar en detalles pero es importante ver que hay un apartado donde indica cómo usarlo con SSL. De hecho, aunque no esté activo porque aparece comentado, se puede observar que viene preparado para usarlo.

```
        <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
                   maxThreads="150" scheme="https" secure="true"
                   clientAuth="false" sslProtocol="TLS" />
```

Sin embargo ahora queremos configurar Tomcat para funcionar como apoyo a un servidor web (en nuestro caso Apache). Para ello necesitamos el conector *Apache JServ Protocol* (AJP). Esto es desde el lado de Tomcat, pero ¿qué sucede con Apache? En su lado podemos usar dos módulos para realizar esta conexión: *mod_jk* o *mod_proxy*. Incluso de este último existen dos versiones: *mod_proxy_ajp* y *mod_proxy_http*. Aunque no es fácil y ni siquiera hay unanimidad sobre estas cuestiones, vamos a intentar aclarar un poco cual usar:

- **mod_jk**: Es el más maduro y probado de todos. Realmente es el más potente y se desarrolla por la comunidad Tomcat y no como módulo de Apache. Por ello no viene incluido en la distribución y Apache y hay que añadirlo a mano. En general si no se necesita alguna de sus funcionalidades específicas no parece recomendable afrontar el resto de inconvenientes.
- **mod_proxy_ajp**: se distribuye con Apache desde las versiones 2.2 y usa el protocolo AJP que tiene mejor rendimiento que HTTP comunicando servidores.
- **mod_proxy_http**: Es similar pero usando HTTP en lugar de AJP. Es una manera fácil y rápida de configurarlo y ponerlo en funcionamiento pero que en general tiene varios inconvenientes que desaconsejan su uso en entornos de producción si no es necesario.

De cualquier forma el primer paso que debemos dar en instalar Apache y Tomcat. Nosotros ya tenemos el segundo por lo que añadiremos Apache. Aunque nosotros lo vamos a configurar en la misma máquina, puede estar en dos diferentes y de hecho esto mejorará el rendimiento reduciendo la carga. Ten en cuenta que vamos a realizar dos configuraciones diferentes por lo que es buena idea clonar la máquina virtual en este punto.

mod_jk

Este módulo no viene con la distribución estándar de apache, por lo que habrá que descargarlo aunque deberíamos comprobar que no esté ya descargado en */etc/apache2/mods-available*

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

Lo que produce una salida que es interesante leer porque nos indica que ya se activa el módulo al instalarlo (sin necesidad que activarlos manualmente) y de que tenemos que reiniciar apache para que los cambios tengan efecto.

```
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Paquetes sugeridos:
  tomcat6 libapache-mod-jk-doc
Se instalarán los siguientes paquetes NUEVOS:
  libapache2-mod-jk
0 actualizados, 1 se instalarán, 0 para eliminar y 282 no actualizados.
Necesito descargar 144 kB de archivos.
Se utilizarán 532 kB de espacio de disco adicional después de esta operación.
Des:1 http://es.archive.ubuntu.com/ubuntu/ precise/universe libapache2-mod-jk i386 1:1.2.32-1 [144
kB]
Descargados 144 kB en 0seg. (179 kB/s)
Seleccionando paquete libapache2-mod-jk previamente no seleccionado
(Leyendo la base de datos ... 174672 ficheros o directorios instalados actualmente.)
Desempaquetando libapache2-mod-jk (de .../libapache2-mod-jk_1%3a1.2.32-1_i386.deb) ...
Configurando libapache2-mod-jk (1:1.2.32-1) ...
Enabling module jk.
To activate the new configuration, you need to run:
  service apache2 restart
```

Podemos reiniciar Apache de cualquier forma.

```
/etc/init.d/apache2 restart
```

Deberíamos comprobar que esté activado consultando mods_enabled o

```
apachectl –M
```

Si miramos la configuración del módulo

```
gedit /etc/apache2/mods-available/jk.conf
```

Podemos observar un montón de parámetros de configuración, pero nos basta por ahora con fijarnos en la configuración de trabajadores.

```
    # We need a workers file exactly once
    # and in the global server
    JkWorkersFile /etc/libapache2-mod-jk/workers.properties
```

Un trabajador (*worker*) en Tomcat es una instancia que procesa las peticiones. En nuestra configuración nos bastará con uno solo pero múltiples trabajadores pueden configurarse si es necesario repartir la carga.

La directiva que hemos visto antes indica en qué archivo se configurarán los *workers*. Podemos editar el archivo para configurar estos trabajadores:

```
sudo gedit /etc/libapache2-mod-jk/workers.properties
```

Vemos que viene configurado para Tomcat 6 pero cambiarlo es sencillo

```
#
# workers.tomcat_home should point to the location where you
# installed tomcat. This is where you have your conf, webapps and lib
# directories.
#
workers.tomcat_home=/usr/share/tomcat7
```

Lo siguiente a configurar es la lista de trabajadores que estarán disponibles. Solo vamos a tener uno pero se pueden añadir más separando los nombres con comas (,)

```
#
#------ worker list ------------------------------------------
#-------------------------------------------------------------------------
#
#
# The workers that your plugins should create and work with
#
worker.list=ajp13_worker
```

Por ejemplo si tuviéramos dos

```
worker.list = workerprueba1, workerprueba2
```

Por último hay que configurar cada worker

```
#
#------ ajp13_worker WORKER DEFINITION ------------------------------
#-----------------------------------------------------------------------
#

#
# Defining a worker named ajp13_worker and of type ajp13
# Note that the name and the type do not have to match.
#
worker.ajp13_worker.port=8009
worker.ajp13_worker.host=localhost
worker.ajp13_worker.type=ajp13
```

Esto se podría repetir para diferentes workers. Mucho más sobre estas configuraciones se puede consultar en la ayuda.

Como hemos cambiado la configuración del módulo hay que reiniciar Apache.

```
/etc/init.d/apache2 restart
```

Cuando hemos visto el archivo *server.xml* de Tomcat observamos que el conector AJP viene configurado pero comentado. Si lo descomentamos está preparado para funcionar.

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

Reinicia Tomcat para que los cambios surtan efecto

```
/etc/init.d/tomcat7 restart
```

Todavía nos falta indicar a Apache qué peticiones debe redirigir a Tomcat. Esto se puede hacer en varios puntos dependiendo de cómo tengamos configurado el servidor web, pero en nuestro caso lo vamos a realizar en el sitio web por defecto.

```
gedit /etc/apache2/sites-available/default
```

Donde montamos los directorios e indicamos el trabajador que atenderá las peticiones

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost

    DocumentRoot /var/www

    JkMount /0330_form/ ajp13_worker
    JkMount /0330_form/* ajp13_worker
```

Tras reiniciar Apache

```
/etc/init.d/apache2 restart
```

podemos acceder a la aplicación a través de Apache, observa que no indicamos el puerto: http://localhost/0330_form/

Configura Apache y Tomcat para que colaboren y que Apache redirija las peticiones de los ejemplos de Tomcat.

mod_proxy

No se debe usar este módulo con el anterior, por lo que empezaré con otra máquina virtual. Este módulo es la solución para usar AJP que viene integrada con Apache desde las versiones 2.2. Como se puede observar en la introducción de la documentación del módulo existen varios derivados del módulo que permiten añadir funcionalidad. Estos derivados requieren que el módulo principal esté presente para poder funcionar.

Lo primero que hay que hacer es activar el módulo de proxy y el que habilita el protocolo HTTP.

```
sudo a2enmod proxy
sudo a2enmod proxy_http
```

Editamos la configuración del módulo

```
gedit /etc/apache2/mods-available/proxy.conf
```

Ten en cuenta que para nuestro objetivo no es necesario descomentar la primera línea como explican en los comentarios que están justo encima. La directiva *ProxyPreserveHost* indica que se le debe pasar el nombre del host que viene en la petición al servidor de aplicaciones en lugar de otro que indiquemos nosotros. Esto es importante para el virtual hosting en el servidor de aplicaciones.

```
# If you only want to use apache2 as a reverse proxy/gateway in
# front of some web application server, you DON'T need
# 'ProxyRequests On'.

ProxyRequests Off
ProxyPreserveHost On
<Proxy *>
        AddDefaultCharset off
        Order deny,allow
            Allow from all
</Proxy>
```

Posteriormente editamos el archivo del host virtual que estemos configurando

```
gedit /etc/apache2/sites-available/default
```

Y configuramos las redirecciones. Ten en cuenta que el archivo está cortado y que las tres primeras líneas ya estaban así.

```
<VirtualHost *:80>
     ServerAdmin webmaster@localhost

     DocumentRoot /var/www

     ProxyPass /0330_form http://localhost:8080/0330_form
     ProxyPassReverse /0330_form http://localhost:8080/0330_form

     <Location /0330_form >
          Order allow,deny
```

```
            Allow from all
        </Location>
```

y reiniciamos Apache antes de ponernos a configurar Tomcat.

```
    /etc/init.d/apache2 restart
```

Solo tenemos que realizar un pequeño ajuste en el conector de server.xml

```
    gedit /etc/tomcat7/server.xml
```

añadimos una línea con el puerto del que recibiremos las redirecciones.

```
        <Connector port="8080" protocol="HTTP/1.1"
                   connectionTimeout="20000"
                   URIEncoding="UTF-8"
                   redirectPort="8443"
                  proxyPort ="80" />
```

y reiniciamos Tomcat

```
    /etc/init.d/tomcat7 restart
```

Ahora, al igual que en el caso anterior podemos acceder a la aplicación a través del servidor web http://localhost/0330_form

mod_proxy_ajp

Pero, ¿y el protocolo AJP?

Se configura prácticamente igual. Además pueden coexistir HTTP (y HTTPS) y AJP. En este apartado solo voy a destacar los puntos donde la configuración es diferente por lo que si quieres configurar solo AJP deberías leer antes el punto anterior.

Lo primero sería activar el módulo

```
sudo a2enmod proxy_ajp
```

Posteriormente editamos el archivo del host virtual que estemos configurando

```
gedit /etc/apache2/sites-available/default
```

Y configuramos las redirecciones. Mantengo la configuración del caso anterior para que se vea que son compatibles.

```
<VirtualHost *:80>
     ServerAdmin webmaster@localhost

     DocumentRoot /var/www

     ProxyPass /0330_form http://localhost:8080/0330_form
     ProxyPassReverse /0330_form http://localhost:8080/0330_form

     <Location /0330_form >
          Order allow,deny
          Allow from all
```

```
        </Location>

        ProxyPass /examples/jsp ajp://localhost:8009/examples/jsp
        ProxyPassReverse /examples/jsp ajp://localhost:8009/examples/jsp

        <Location /examples/jsp >
             Order allow,deny
             Allow from all
        </Location>
```

y reiniciamos Apache antes de ponernos a configurar Tomcat.

```
    /etc/init.d/apache2 restart
```

Solo tenemos que realizar un pequeño ajuste en el conector de server.xml

```
    gedit /etc/tomcat7/server.xml
```

descomentamos el conector AJP

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

y reiniciamos Tomcat

```
/etc/init.d/tomcat7 restart
```

Ahora, al igual que en el caso anterior podemos acceder a la aplicación a través del servidor web http://localhost/0330_form (con HTTP) y a http://localhost/examples/jsp/ (con AJP)

> Configura Apache y Tomcat para que todo que colaboren mostrando la documentación de Tomcat en http://localhost/tom-doc/
>
> Pruébalo con HTTP y luego con AJP.

Seguridad en el servidor de aplicaciones. Configurar el servidor de aplicaciones con soporte SSL/TSL.

Ya vimos que HTTPS no era más que HTTP con una capa intermedia. Los mismos principios se aplican en Tomcat. En este caso también usaremos certificados digitales.

Es importante destacar que aunque en Tomcat se puede usar un conector que usa OpenSSL nosotros vamos a ver los que se basan en Java SSL. Desde JDK 1.4 se incluse JSSE en las distribuciones, por lo que tenemos a nuestra disposición todo lo necesario. Nosotros solo vamos a generar un certificado autofirmado para Tomcat, pero en esta página se pueden ver los pasos para crear uno y pedir que nos firme el certificado una CA.

Vamos a crear un almacén de certificados. La opción –genkey especifica que se cree un par clave pública/privada que se almacenan en un certificado autofirmado. “almacen” es el nombre del archivo que se va a crear.

```
sudo keytool -genkey -alias tomcat -keyalg RSA -keystore /var/lib/tomcat7/almacen
```

Lo que nos va pidiendo los datos necesarios.

```
Introduzca la contraseña del almacén de claves:
Volver a escribir la contraseña nueva:
¿Cuáles son su nombre y su apellido?
  [Unknown]:  Sergio Cuesta
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:  Consejería de Educación
```

```
     ¿Cuál es el nombre de su organización?
       [Unknown]:  Comunidad de Madrid
     ¿Cuál es el nombre de su ciudad o localidad?
       [Unknown]:  Madrid
     ¿Cuál es el nombre de su estado o provincia?
       [Unknown]:  Madrid
     ¿Cuál es el código de país de dos letras de la unidad?
       [Unknown]:  ES
     ¿Es correcto CN=Sergio Cuesta, OU=Consejería de Educación, O=Comunidad de Madrid, L=Madrid,
ST=Madrid, C=ES?
       [no]:  si

     Introduzca la contraseña de clave para <tomcat>
         (INTRO si es la misma contraseña que la del almacén de claves):
```

Durante esta configuración nos pide que definamos las contraseñas tanto del almacén como del archivo de claves. En mi caso ambas son “sergio".

Lo siguiente sería habilitar el conector HTTPS en server.xml

```
         <!-- Define a SSL HTTP/1.1 Connector on port 8443
              This connector uses the JSSE configuration, when using APR, the
              connector should be using the OpenSSL style configuration
              described in the APR documentation -->
         <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
                    maxThreads="150" scheme="https" secure="true"
                    clientAuth="false" sslProtocol="TLS"
                   keystoreFile="/var/lib/tomcat7/almacen" keystorePass="sergio"
                   keyAlias="tomcat" keyPass="sergio" />
```

Puedes ver que he añadido las dos últimas líneas para indicar la configuración de seguridad. Tras reiniciar Tomcat podemos conectarnos al servidor de aplicaciones con conexiones seguras https://localhost:8443/

El problema es que aunque ahora puedo conectarme a las aplicaciones mediante HTTPS sigo teniendo la opción de hacerlo mediante HTTP http://localhost:8080/0330_form/

Para obligar a que a una aplicación haya que conectarse mediante HTTPS hay que modificar su descriptor de despliegue. Para hacer la prueba modifico el descriptor de despliegue de 0320_basic

```
gedit /var/lib/tomcat7/webapps/0320_basic/WEB-INF/web.xml
```

Hay que cambiar el tipo de transporte que se obliga de NONE a CONFIDENTIAL

```
            <user-data-constraint>
                <transport-guarantee>CONFIDENTIAL</transport-guarantee>
            </user-data-constraint>
```

Podemos ver que ahora si probamos a conectarnos a http://localhost:8080/0320_basic/ nos redirige automáticamente a https://localhost:8443/0320_basic/

Ya sabemos configurar Apache y Tomcat con SSL pero **¿qué sucede con la conexión entre ellos?**

Es muy raro que esta conexión se realice a través de una red en la que no confiemos. Incluso en el caso de que se encuentren en máquinas diferentes suele usarse algún tipo de conexión privada entre ellos para que los mensajes que se envían no se distribuyan por redes más o menos públicas. Si necesitáramos configurarlo de todas formas deberíamos prestar atención a las directivas que comienzan por SSLProxy del módulo mod_ssl de Apache.

De todas formas para asegurar la conexión Apache – Tomcat (con HTTPS) tendríamos que usar un conector HTTPS, lo que hago un poco más arriba y cambiar las cadenas de redirección en la configuración del host virtual para que utilicen HTTPS y el puerto 8443.

Prácticas finales

En este punto y con los conocimientos de hosts virtuales que aprendimos en Apache, estamos preparados para aprender a crear hosts virtuales en Tomcat por nosotros mismos. Por ello debes aprender a configurar un host virtual.

Te ayudarán mucho, la aplicación de host manager que instalamos con los paquetes adicionales y repasar todo este tema para ver dónde se configuraba un elemento <Host> y qué implicaciones tenía. Ten en cuenta que al intentarte conectar al host manager, si no puedes te dice el rol que tienes que tener para poder conectarte. También hay un pequeño tutorial en la documentación de Tomcat. También es importante recordar que el host manager no configura el elemento <Host> de server.xml, pero internet está lleno de tutoriales para realizar esto.

Tu objetivo final debe ser poder configurar un nuevo host virtual en Tomcat a mano y ser capaz de desplegar una de las aplicaciones de ejemplo (mejor con autenticación) sobre él. Las siguientes líneas te pueden dar una idea de cómo hacerlo.

```
<Host name="www.sergio.es" appBase="webapps_sergio.es"
       unpackWARs="true" autoDeploy="true" >
   <Alias>sergio.es</Alias>
   <Valve className="org.apache.catalina.valves.AccessLogValve"
          directory="logs"
          prefix="sergio.es_access_log." suffix=".log"
          pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b"
          resolveHosts="false" />
```

```
</Host>
```

Práctica a entregar

El último ejercicio que vamos a realizar implica usar casi todo lo que hemos visto en Apache y Tomcat. Debes configurar un sistema que gestiona peticiones HTTPS exclusivamente (no HTTP) por lo que tanto Tomcat como Apache deben usar SSL. El sitio debe tener tu nombre como nombre de dominio y Tendrá archivos HTML que servirá Apache y parte con Servlets y JSPs que servirá Tomcat a través de Apache. El control de acceso se realizará con un JDBCRealm. Puedes usar los ejemplos del site para la parte de aplicaciones en el servidor. Si decides desarrollar las tuyas propias se tendrá en cuenta para la nota, pero no es necesario para que la práctica esté bien. Habrá enlaces en la parte HTML a los Servlets/JSPs. Un usuario introducirá únicamente sus credenciales una vez para todas las aplicaciones.

Pon tu nombre en comentarios tanto al principio como al final de cada archivo de configuración o de contenido que edites.

Evita que se pueda acceder a las aplicaciones de Tomcat sin pasar por Apache.