Servidores Web

Sesión 2: Eclipse WebTools.
Configuración de Tomcat.

Puntos a tratar

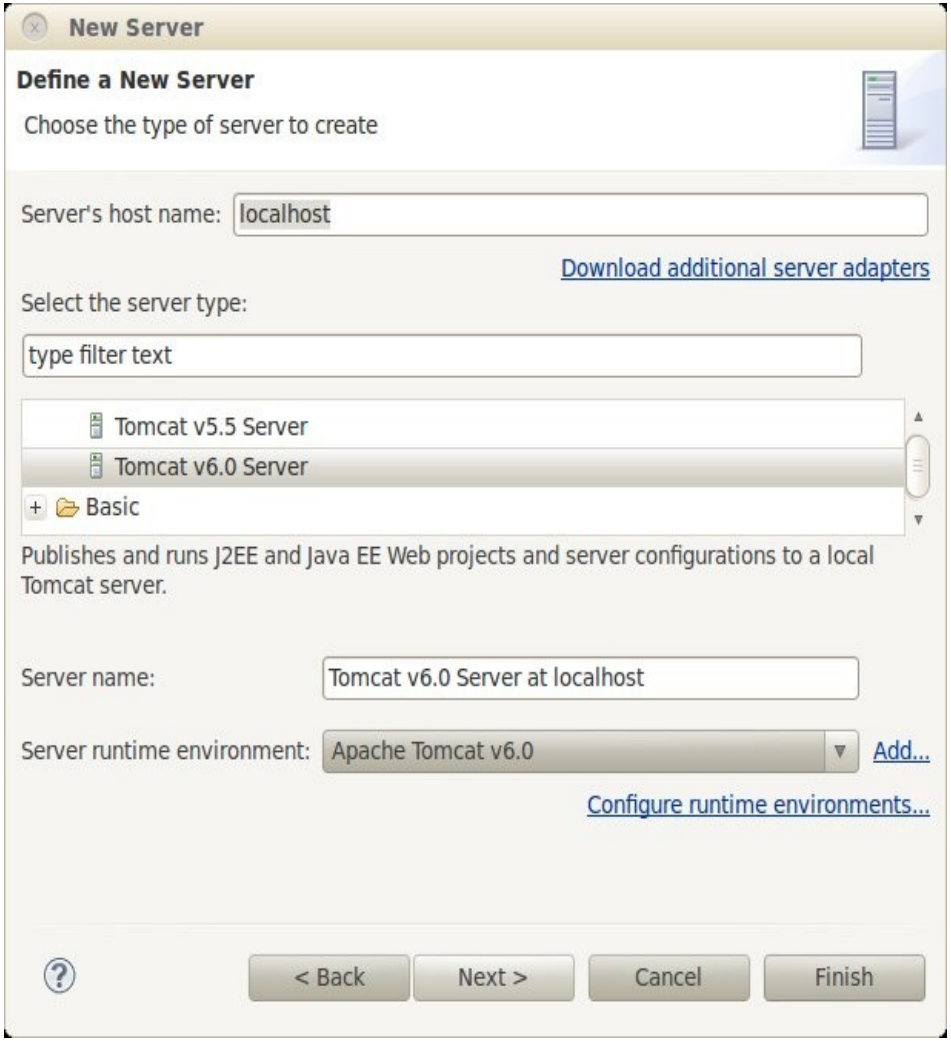
- Eclipse WebTools (WTP)
- Configuración de Tomcat
  - Estructura física y lógica
  - Formas de cambiar la configuración
  - Configurar el host
  - Configuración en Eclipse
- Configuración de aplicaciones web
  - El descriptor de despliegue
  - El contexto de la aplicación en Tomcat

WebTools (WTP)

- Es un plugin de Eclipse que gestiona aplicaciones web como proyectos autointegrados
- Incluido con la versión JavaEE de Eclipse
- Podremos:
  - Gestionar el servidor web en que desplegar
  - Crear y desarrollar la aplicación web
  - Desplegar y probar la aplicación en el servidor

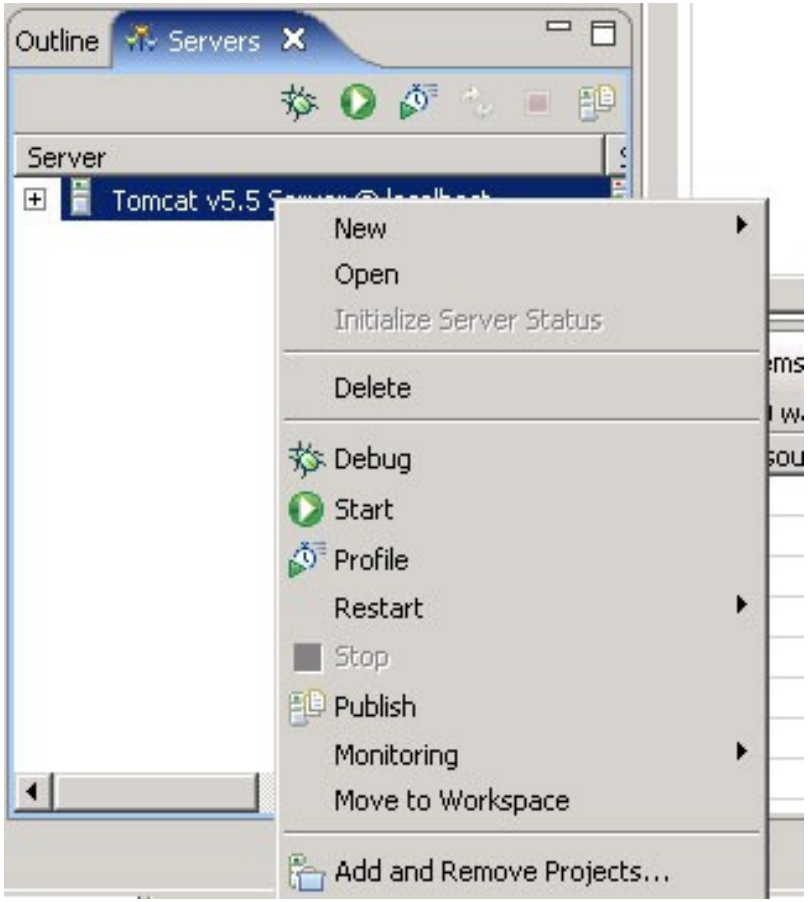
Añadir servidores web

- `File > New > Servers > Server`



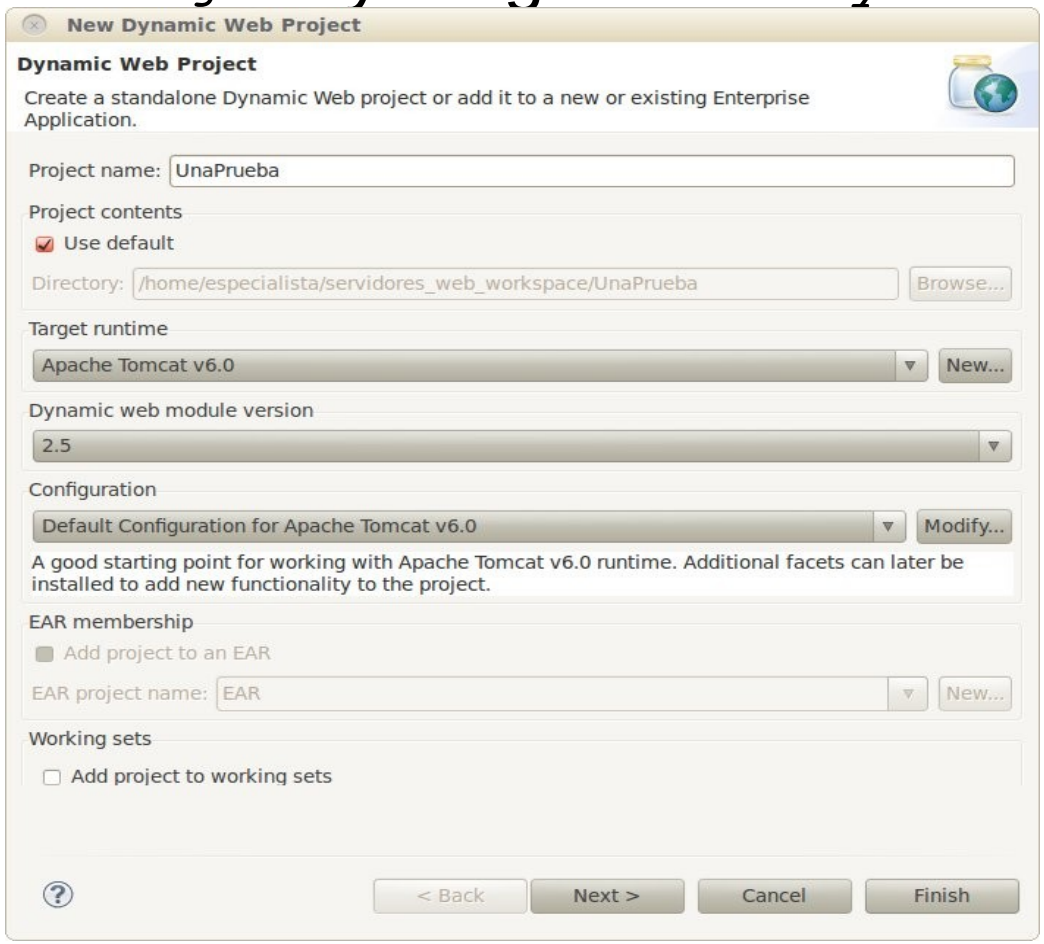
Gestionar los servidores web

- Botón derecho sobre el servidor en la vista *Servers*
- Tenemos opciones para pararlo, reanudarlo, etc



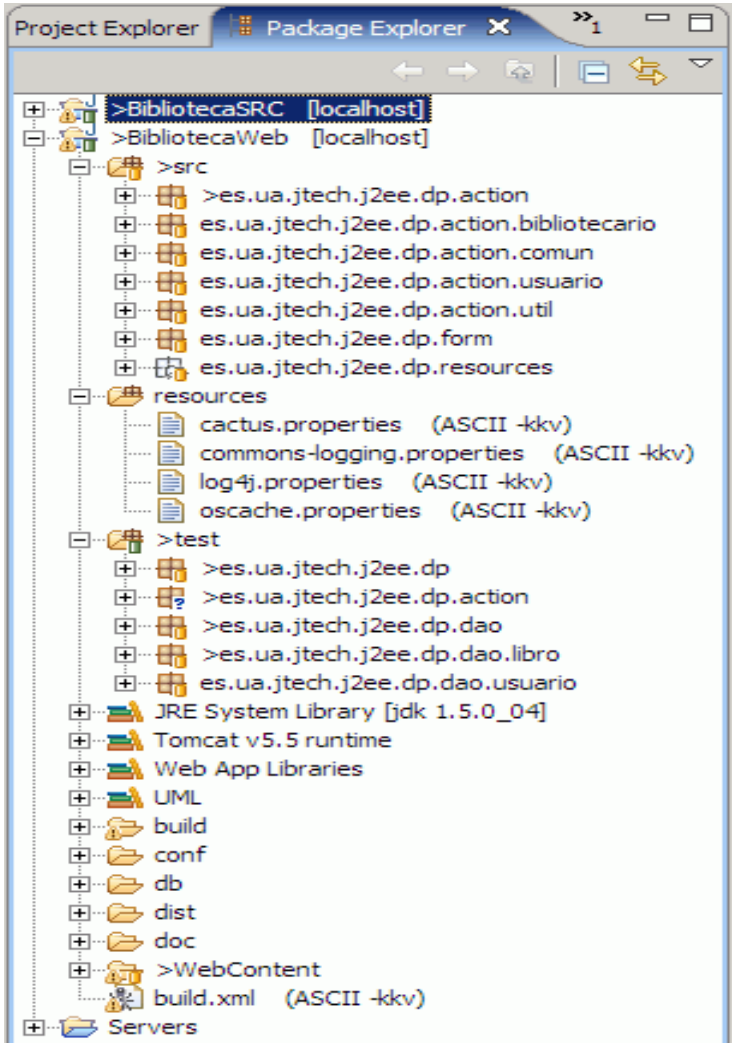
Crear proyecto de aplicación web

- Ir a *File – New – Project* y elegir *Web – Dynamic Web Project*



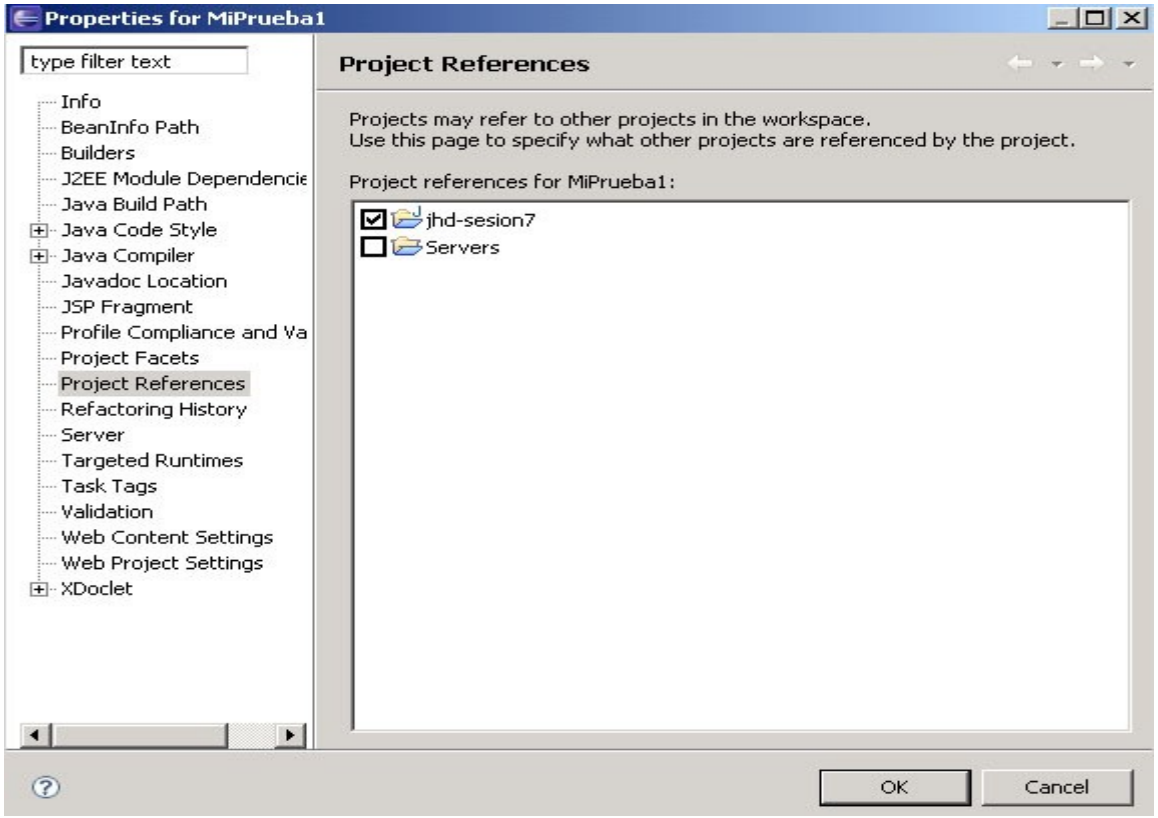
Crear proyecto de aplicación web (2)

- En los siguientes pasos del asistente, elegimos qué carpetas crear y la ruta del contexto que tendrá la aplicación
- Carpetas creadas por defecto:
  - **src**: fuentes
  - **WebContent**: esqueleto aplicación web (con WEB-INF y sus subcarpetas)
  - El resto de carpetas las crearemos nosotros a mano



Interdependencias entre proyectos

- Podemos hacer que nuestro proyecto web dependa de otros proyectos previos



Despliegue de la aplicación

- La aplicación se desplegará sobre el servidor que tengamos asignado en la vista *Servers*.
- Pulsamos botón derecho sobre el proyecto web y elegimos *Run As – Run on Server*
  - En la siguiente pantalla podemos elegir sobre qué servidor de la vista de *Servers* ejecutarlo, si tuviésemos más de uno configurado.
- Repetiremos la operación tras cada cambio que queramos comprobar en la aplicación.

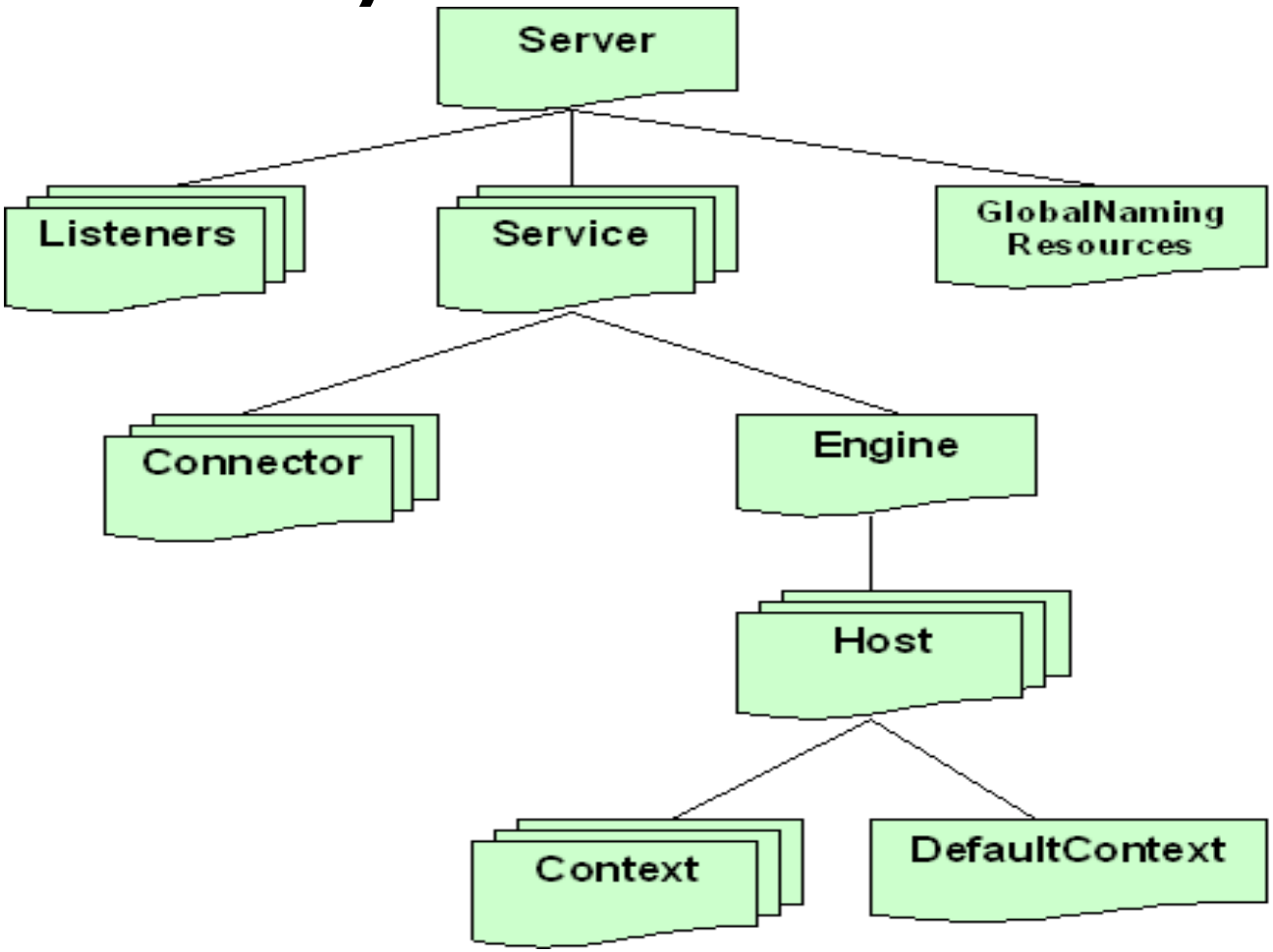
Estructura física de Tomcat

- Directorio *home* de Tomcat
  - bin
  - lib: **clases comunes a serv. y aplic.**
    - **Por ejemplo, drivers de BD**
  - conf: **configuración del servidor**
  - logs: **dir. por defecto de logs de depuración**
  - webapps: **aplic. web**
  - work, temp: **dir. temporales**

Ficheros de configuración

- Dentro de **conf**:
  - server.xml (configuración principal)
  - web.xml (config.global a todas las aplicaciones)
  - tomcat-users.xml (logins y passwords de usuarios)
  - catalina.policy (fichero de políticas de seguridad)

Estructura lógica de Tomcat (*server.xml*)



Componentes principales

- *Server*: el propio Tomcat. Sólo hay una instancia de este elemento
- *Listener*: monitorizan los contenedores web
- *GlobalNamingResources*: mapean variables JNDI
- *Service*: conjunto de conectores que reciben peticiones y un *engine* que las procesa
- *Connector*: acepta ciertas peticiones y las pasa al *engine*
- *Engine*: representa al contenedor web (p. ej. Catalina)
- *Host*: representa al host o host virtual (p.ej. localhost)
- *Context*: representa una aplicación web en un host

Configuración del host

- La etiqueta *Host* de *server.xml* define la configuración general para un host o host virtual (es decir, un subgrupo de aplicaciones dentro del servidor web)

```
<Host name="localhost" debug="0" appbase="webapps"
  unpackWars="true" autoDeploy="true">
```

Atributo	Significado	Valor por defecto
`name`	nombre del host o host virtual	ninguno
`debug`	nivel de mensajes de depuración	0
`appBase`	directorio donde se instalarán las aplicaciones de este host (si es relativa, la ruta se supone con respecto al directorio de Tomcat)	ninguno
`unpackWARs`	si es `true`, las aplicaciones empaquetadas en WAR se desempaquetan antes de ejecutarse	`true`
`autoDeploy`	si es `true`, se utiliza el despliegue automático de aplicaciones	`true`
`liveDeploy`	si es `true`, se utiliza despliegue automático sin necesidad de rearrancar Tomcat	`true`

Componentes reubicables

- Hay algunos componentes que se pueden definir a varios niveles, según el ámbito que queramos que tengan:
  - *Valves*: para filtrar peticiones
  - *Loggers*: para logs de depuración de errores
  - *Realms*: define un conjunto de usuarios con permiso de acceso a un determinado contexto o aplicación web
  - *Managers*: implementan el manejo de sesiones HTTP.
  - *Loaders*: cargador de clases para aplicaciones web

Configuración de la aplicación: web.xml

- Cada aplicación web / contexto debe tener un fichero descriptor de despliegue
  - Fichero `web.xml`
  - Ubicado en la ruta `/WEB-INF/web.xml` del contexto
- Describe y configura la aplicación web
- Fichero XML definido de forma estándar para la configuración de las aplicaciones Java EE

Descriptor básico

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:web="http://java.sun.com/xml/ns/javaee/web-
   app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">
<web-app>
  <display-name>Mi Aplicacion Web</display-name>
  <description>
    Esta es una aplicacion web sencilla a modo de ejemplo
  </description>
  <!-- Resto de elementos -->

</web-app>
```

Elementos del descriptor

- En el descriptor de despliegue se configura:
  - Variable globales del contexto
  - Filtros
  - Servlets y mapeo a URLs
  - Listeners (de cambios en el contexto)
  - Sesiones
  - Página de inicio
  - Restricciones de seguridad
  - Librerías de *tags*
  - Etc ...

Contexto

- Cada Aplicación Web es un contexto
- A cada contexto se le asigna una ruta dentro del servidor
  - Por ejemplo, si asignamos la ruta `aplic` al contexto correspondiente a la siguiente estructura:

```
/pagina.htm
/WEB-INF/web.xml
```

  - Podremos acceder a nuestra página con

```
http://localhost:8080/aplic/pagina.htm
```

Contexto de una aplicación web

- Elemento `Context`

Atributo	Significado	Valor por defecto
`crossContext`	indica si la aplicación web puede comunicarse con otras	`false` (por razones de seguridad)
`debug`	nivel de mensajes de depuración	0
`docBase`	directorio donde está la aplicación, bien especificado de modo absoluto o con respecto al `appBase` del Host	ninguno
`path`	*path* de esta aplicación, que se hace coincidir con la URL de la petición para ver qué aplicación debe servirla.	ninguno
`reloadable`	permite monitorizar los cambios de clases en `/WEB-INF/classes` y `/WEB-INF/lib` para que no sea necesario reiniciar Tomcat	`false` (se suele poner a `true` durante el desarrollo)

Dónde especificar el *Context*

- Dentro del `server.xml`

```
<Context docBase="miAplic" path="/miaplic"
  …/>
```

  - Desaconsejado, hay que tocar la configuración del servidor entero
- Para la aplicación individualmente
  - Dentro de una carpeta **META-INF**, en un archivo llamado **context.xml**
  - Se puede hacer de otras formas, consultar la documentación.

Valves

- Componentes que se insertan en el ciclo de procesamiento de la petición para controlar varios aspectos:
  - Registro de accesos
  - Filtro de hosts o de IPs
  - Volcado de la petición
  - ...
- Clases Java que el usuario podría implementar
- Se pueden poner a distintos niveles: **engine, host o context**
  - O sea, para todas las aplicaciones o para una sola

Ejemplos de *Valves*

- Log de accesos

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

- Filtro de IPs

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

Pooling de conexiones en Tomcat

- Mediante ficheros de configuración podemos dejar definida una batería de conexiones a BD
  - Estas conexiones se abrirán al arrancar el servidor, aunque todavía no se necesiten
- Cada petición de acceso a BD de las aplicaciones irá cogiendo una de las conexiones libres y marcándola como “ocupada”. Cuando la cierre, en realidad simplemente la marcará como “libre”
  - Merece la pena “malgastar” conexiones abiertas con la BD, ya que el proceso de abrirlas/cerrarlas es costoso en tiempo
  - De esta forma aseguramos poder atender un máximo determinado de peticiones concurrentes (el tamaño del *pool*)

Configuración del pooling (1/2)

- Definir un fichero **context.xml** en la carpeta *WebContent/META-INF* de nuestro proyecto

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Context>
    <Resource
        name="PruebaDS"
        type="javax.sql.DataSource"
        auth="Container"
        username="prueba"
        password="prueba"
        driverClassName="com.mysql.jdbc.Driver"
        url="jdbc:mysql://localhost:3306/prueba"
        maxActive="20"
        maxIdle="5"
        maxWait="10000"/>
</Context>
```

Acceso a la BD

- La clase DataSource es la que hace transparente el *pooling*, permitiéndonos usar `getConnection/close` como en JDBC sin *pooling*
- Se usa el API JNDI para acceder al DataSource por su nombre simbólico

```
//Obtener el contexto JNDI
Context initCtx = new InitialContext();
//Obtener el recurso con su nombre lógico (JNDI)
DataSource ds = (DataSource) initCtx.lookup("java:comp/env/PruebaDS");
//A través del DataSource podemos obtener una conexión con la BD
Connection conn = ds.getConnection();
//A partir de aquí trabajaríamos como es habitual en JDBC
```