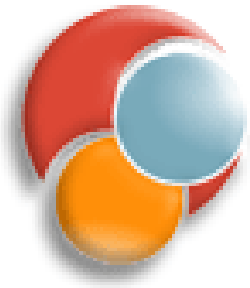


Servidores Web

Sesión 1: Protocolo HTTP y aplicaciones web

Puntos a tratar

- El protocolo HTTP
- Introducción a Tomcat
- Introducción a las aplicaciones web
  - Estructura estándar
  - Despliegue con Tomcat Manager

Protocolos

- Internet se construye sobre TCP/IP
  - TCP: Protocolo de transporte
  - IP: Identificación de las máquinas de la red
  - Comunicación mediante *sockets*
  - Cada aplicación utiliza un determinado puerto por defecto
- Protocolos de aplicación sobre TCP/IP
  - HTTP: Acceso a la web (puerto 80)
  - FTP: Transferencia de ficheros (puerto 21)
  - Telnet: Terminales remotos (puerto 23)
  - SMTP: Envío de correo (puerto 25)
  - POP3: Recepción de correo (puerto 110)
  - etc

Protocolo HTTP

- Comunicación entre cliente y servidor
  - El cliente solicita un documento del servidor
  - El servidor sirve el documento al cliente
- Mecanismo petición/respuesta
  - Se solicita el documento enviando un mensaje de petición HTTP al servidor
  - El servidor devuelve el documento requerido al cliente dentro de un mensaje HTTP de respuesta

    Si el documento no puede ser servido, devolverá un mensaje HTTP de respuesta indicando el error producido
  - Sin estado

    Cada petición es independiente para el servidor

Protocolo HTTP: mensaje de petición

- Lo envía el cliente al servidor HTTP
  - Solicita un recurso
- Se compone de:
  - Comando HTTP

    Compuesto por: `Comando URI Protocolo`

    Por ejemplo: `GET /index.htm HTTP/1.1`
  - Cabeceras

    Información sobre la petición

    La sección de cabeceras finaliza con una línea en blanco (`\r\n\r\n`)
  - Contenido adicional

    Bloque de datos de cualquier tipo

Protocolo HTTP: comandos

- Comandos `GET` y `POST`
  - Se utilizan para solicitar un documento al servidor
  - `GET` proporciona los parámetros en la URI

```
GET /servlet/envia?msg=hola&usr=miguel HTTP/1.1
```

  - `POST` proporciona los parámetros en el bloque de contenido
- Otros comandos:

  `OPTIONS`: Consulta opciones del servidor

  `HEAD`: Solicita información sobre el recurso (no su contenido)

  `PUT`: Guarda un fichero en el servidor

  `DELETE`: Borra un fichero del servidor

  `TRACE`: Muestra el camino seguido por la petición

Protocolo HTTP: comando GET

- Se realiza esta petición cuando pulsamos sobre un enlace en una página web
  - Si queremos proporcionar parámetros tendremos que incluirlos en la misma URL

```
<a href=“pag.jsp?id=123&nombre=pepe">Pulsa Aqui</a>
```

- También se realiza cuando utilizamos formularios con método GET

```
<form action="pag.jsp“ method=“GET”>
  <input type="text" name="id" value="123">
  <input type="text" name="nombre" value="pepe">
  <input type="submit" value="Enviar">
</form>
```

  - Los datos introducidos en el formulario se envían en la URI

```
GET /pag.jsp?id=123&nombre=pepe HTTP/1.1
<cabeceras>
```

Protocolo HTTP: comando POST

- Se realiza cuando utilizamos un formulario con método POST

```
<form action="pag.jsp" method="POST">
  <input type="text" name="id" value="123">
  <input type="text" name="nombre" value="pepe">
  <input type="submit" value="Enviar">
</form>
```

- Los parámetros se envían en el bloque de contenido

```
POST /pag.jsp HTTP/1.1
<cabeceras>

id=123&nombre=pepe
```

Protocolo HTTP: cabeceras de petición

- Envían información sobre
  - El agente de usuario (navegador)
  - La petición realizada
- Algunas cabeceras estándar son:

`Accept-Language`	Idiomas aceptados
`Host`	Host y puerto indicado en la URL (requerido)
`If-Modified-Since`	Sólo se desea el documento si ha sido modificado tras esta fecha
`User-Agent`	Tipo de cliente que realiza la petición

- Por ejemplo, según el idioma especificado en la petición, algunos servidores podrán devolver el documento en dicho idioma

Protocolo HTTP: mensaje de respuesta

- El servidor nos responderá con un mensaje HTTP de respuesta
- Este mensaje se compone de:
  - Código de estado:
    Indica si se ha procesado correctamente o si ha habido un error
    Ejemplo: `HTTP/1.1 200 OK`
  - Cabeceras
    Información sobre el recurso y sobre el servidor
    Se definen de la misma forma que las de la petición
  - Contenido
    En el bloque de contenido se incluye el recurso devuelto, si se ha devuelto alguno

Protocolo HTTP: códigos de estado

- Indica el resultado de la petición
- Encontramos varios grupos de códigos:
  - `1XX`: Códigos de información
  - `2XX`: Códigos de aceptación
    - `200 OK`: Se ha servido correctamente
    - `204 No content`: No hay contenido nuevo
  - `3XX`: Redirecciones, el documento ha sido movido
  - `4XX`: Errores en la petición
    - `400 Bad request`: El mensaje de petición tiene sintaxis errónea
    - `401 Unauthorized`: El usuario no tiene permiso
  - `5XX`: Errores en el servidor
    - `500 Internal Server Error`: Error interno del servidor

Protocolo HTTP: cabeceras de respuesta

- El servidor también puede enviar cabeceras en la respuesta con información sobre
  - El documento devuelto
  - Las características del servidor
- Algunas cabeceras estándar de la respuesta son:

`Content-Length`	Longitud del contenido (en bytes)
`Content-Type`	Tipo MIME del contenido
`Last-Modified`	Fecha de modificación del documento

- Podemos establecer estas cabeceras también desde la cabecera del código HTML de nuestro documento:

```
<META HTTP-EQUIV="Cabecera" CONTENT="Valor">
```

Cookies: definición

- El protocolo HTTP no tiene estado
- Podemos implementar estado sobre HTTP utilizando cookies
  - No es propio del protocolo HTTP
  - Está soportado por la mayoría de los navegadores
- Las cookies se componen de

```
nombre=valor
```

- Se almacenan en el cliente
- Se envían en cada petición al servidor
  - Identifican al cliente en cada petición

Cookies: envío y recepción

- El servidor envía una cookie al cliente con la cabecera `set-cookie`

```
Set-Cookie: CLAVE1=VALOR1;...;CLAVEN=VALORN [OPCIONES]
```

- Donde `OPCIONES` es

```
expires=FECHA;path=PATH;domain=DOMINIO;secure
```

- El cliente almacena la cookie de forma local
- En sucesivas peticiones al servidor se envía la cookie en la cabecera `cookie`

```
Cookie: CLAVE1=VALOR1;CLAVE2=VALOR2;...;CLAVEN=VALORN
```

El servidor web Tomcat

- Servidor web construido sobre la plataforma Java
  - Necesitamos tener instalado JDK para utilizarlo
- Soporta parte de la especificación de Java EE para desarrollar aplic. web (servlets y JSPs)
- Instalamos el servidor web descomprimiéndolo en el directorio escogido
  - En Windows contamos con un instalador
- Establecemos variables de entorno

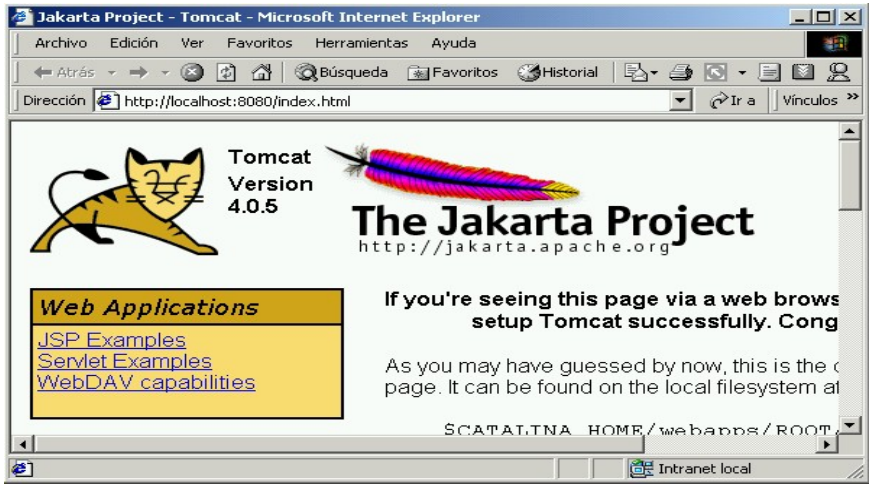
  `JAVA_HOME`: Directorio de JDK

  `CATALINA_HOME`: Directorio donde hemos instalado Tomcat

Ejecución de Tomcat

- En Linux contamos con los comandos:
  `${CATALINA_HOME}/bin/startup.sh`: Activar el servidor
  `${CATALINA_HOME}/bin/shutdown.sh`: Detener el servidor
- En Windows tenemos un *Monitor* en el menú *Inicio – Programas*, que permite iniciar y parar el servidor con el ratón
- Una vez en marcha podemos acceder a su página de bienvenida:

```
http://localhost:8080/
```



Estructura física de Tomcat

- Tomcat
  - bin

    catalina.sh

    startup(=catalina start), shutdown(=catalina stop)
  - common (classes, libs): **clases comunes a serv. y aplic.**
  - conf: **configuración del servidor**
  - logs: **dir. por defecto de logs de depuración**
  - server: **clases de Tomcat**
  - shared: **clases compartidas por las aplic. web**
  - 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)

- En Tomcat 5.x y anteriores había una aplicación web de configuración y administración del servidor

Aplicaciones web: conceptos generales

- Una aplicación web es una aplicación a la que se accede mediante HTTP
  - Utilizando un navegador web
- A la hora de desarrollar una aplicación web suelen utilizarse diferentes tecnologías
- En el lado del SERVIDOR:
  - Debe ser capaz de recoger la petición del cliente y enviarle la respuesta adecuada
  - Puede valerse de herramientas externas para procesar la petición y generar la respuesta de forma dinámica

    servlets, JSP, PHP, ASP, etc.
- En el lado del CLIENTE:
  - Al cliente se le ofrece una respuesta visible en forma de página web
  - Podemos utilizar elementos estáticos (HTML) o bien valernos de herramientas que den cierto dinamismo también a lo que se envía al cliente

    Javascript, Applets, Flash, etc.

Aplicación web Java EE

- Las aplicaciones web Java EE se componen de
  - Recursos estáticos
    HTML, imágenes, etc.
  - Documentos dinámicos
    Páginas JSP
  - Clases Java
    Servlets, *beans* y otros objetos Java
    Deben ser compiladas
  - Configuración de la aplicación
    Descriptor de despliegue (fichero XML)

Directorios en una aplicación web Java EE

- Estructura de directorios:

`/`	Recursos estáticos y JSP Parte pública accesible desde la web
`/WEB-INF`	Configuración y clases Java No accesible desde la web
`/WEB-INF/web.xml`	Fichero descriptor de despliegue Configuración de la aplicación
`/WEB-INF/classes`	Clases Java de nuestra aplicación Ficheros `.class` (en estructura de paquetes)
`/WEB-INF/lib`	Librerías que utiliza la aplicación Ficheros JAR

Contexto

- Cada Aplicación Web es un contexto
  - Se compone de la estructura de directorios anterior
- 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
```

Ficheros WAR

- Podemos empaquetar las Aplicaciones Web en ficheros WAR (Archivos de Aplicación Web)
- Se utiliza la misma herramienta JAR para crearlos (sólo utilizamos una extensión distinta)
  - Contendrá la estructura de directorios completa del contexto
- Es un estándar de los servidores de aplicaciones Java EE
- Se utiliza para distribuir aplicaciones web
  - Podremos copiar el fichero WAR directamente al servidor web para poner en marcha la aplicación

Ejemplo WAR

- Dada la siguiente estructura de carpetas:

```
/home/especialista/web/ejemplo/
                      index.html
                      WEB-INF/
                                        web.xml
                                        classes/
                                                ClaseServlet.class
```

- Entrar en el directorio */home/especialista* y teclear

```
jar cMvf ejemplo.war *
```

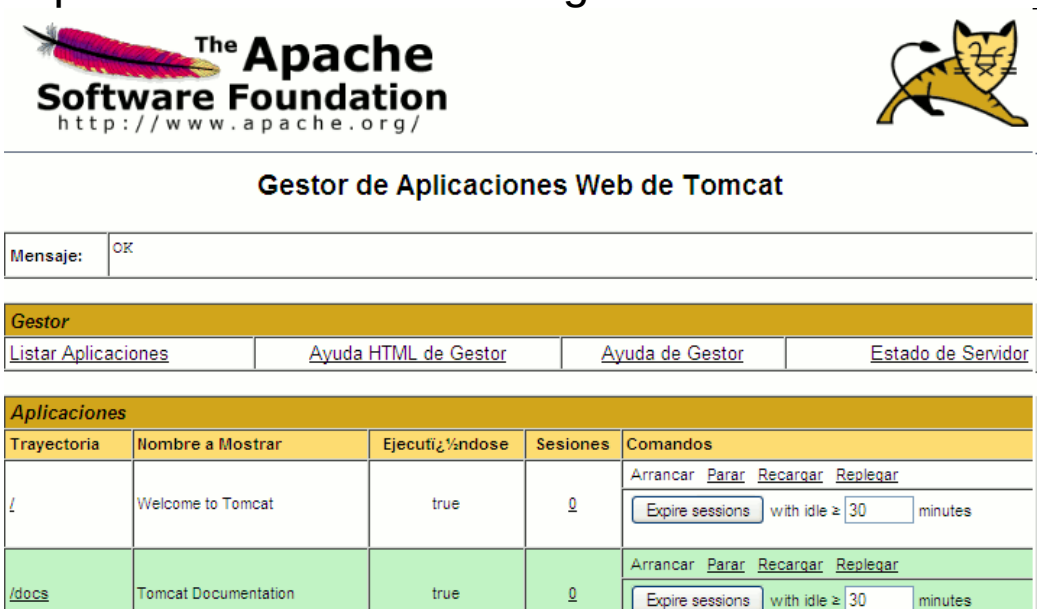
La aplicación “manager” de Tomcat

- Nos permite gestionar las aplicaciones
  - Listar aplicaciones desplegadas
  - Desplegar/replegar una aplicación
  - Rearrancar/parar una aplicación
- Para poder usarlo es necesario tener el rol “manager”
  - Si no existe ningún usuario con dicho rol, podemos crearlo en el tomcat-users.xml

```
<role rolename="manager"/>
<user username="admin" password="JavaEE" roles="manager"/>
```

Formas de ejecutar el manager

- Interfaz Web:
  - http://localhost:8080/manager/html



- El *manager* también acepta peticiones HTTP, que podemos hacer con cualquier programa capaz de generarlas, no solo un navegador, por ejemplo *curl*