

Práctica 03

Programación Web Dinámica con Interpretación en el Servidor

Fundamentos de Aplicaciones y Servicios Telemáticos

2º Curso Grado en Ingeniería de Tecnologías de Telecomunicación

Departamento de Ingeniería Telemática (DIT)

Universidad de Sevilla

Ignacio Campos Rivera

Francisco José Fernández Jiménez

José Ángel Gómez Argudo

Francisco Javier Muñoz Calle

Juan Antonio Ternero Muñiz

@2018

1	Objetivos y alcance	1
1.1	Introducción.....	1
1.2	Objetivo de la práctica.....	2
1.3	Documentación de apoyo	2
1.4	Preparación del entorno de trabajo	2
1.5	Descripción del entorno de trabajo.....	3
2	CGI.....	4
2.1	Introducción.....	4
2.2	Directorios	5
2.2.1	Prueba de programas CGI	9
2.3	Variables CGI.....	10
2.4	Formularios: método GET.....	12
2.5	Formularios: método POST.....	13
2.6	Formularios: varios campos en conjunto.....	18
2.7	Formularios: varios campos por separado	19
2.8	Ejemplo mostrar-entorno.....	22
3	J2EE	23
3.1	Introducción.....	23
3.2	Arquitectura de Java EE	24
3.3	Contenedor Web: Tomcat	24
3.4	Descripción del entorno de trabajo.....	25
4	Servlets	26
4.1	Introducción.....	26
4.2	Ejemplo	26
4.3	Descriptor de despliegue	28
4.4	Estructura de directorios.....	29
4.5	Compilación del Servlet	30

4.6	Ciclo de vida.....	30
4.7	Anotaciones.....	32
4.8	Atributos y ámbitos	38
4.9	Parámetros de la petición y parámetros de inicialización.....	38
4.9.1	Parámetros de la petición.....	39
4.9.2	Parámetros de inicialización del servlet	39
4.9.3	Parámetros de inicialización de la aplicación.....	39
4.10	Clase auxiliar: ServletContextListener.....	40
4.11	Clase auxiliar: Filter	44
4.12	Inclusión y reenvío: RequestDispatcher.....	46
5	JSP.....	47
5.1	Introducción de JSP.....	47
5.1.1	Motivación de JSP.....	47
5.1.2	Aspectos generales de JSP.....	47
5.1.3	Ciclo de vida de una página JSP.....	48
5.1.4	Métodos sobrescribibles en JSP.....	51
5.1.5	Elementos de las páginas JSP.....	51
5.2	Comprobación del código	53
5.3	Directivas	55
5.3.1	Directiva page.....	55
5.3.2	Directiva include	57
5.3.3	Directiva taglib.....	58
5.4	Elementos de script	58
5.4.1	Declaraciones	58
5.4.2	Expresiones	60
5.4.3	Scriptlets.....	62
5.4.4	Comentarios.....	63
5.5	Ejercicios propuestos.....	64
5.5.1	Tabla bicolor.....	64
5.5.2	Tablero de ajedrez	65
5.5.3	Tabla potencias.....	66

5.6	Etiquetas de Acción JSP (Action Tags o Standard Actions)	66
5.6.1	Etiqueta de Acción Include	67
5.6.2	Etiqueta de Acción Forward	68
5.7	Objetos implícitos	69
5.7.1	Atributos en los objetos implícitos	69
5.7.2	request	71
5.7.3	response	73
5.7.4	out	75
5.7.5	session	76
5.7.6	application	76
5.7.7	config	78
5.7.8	pageContext	78
5.7.9	page	80
5.7.10	exception	80
5.8	JavaBeans	81
5.8.1	Introducción	81
5.8.2	Asignación condicional en la creación	86
5.8.3	Asignación de parámetros de un formulario a las propiedades de un javabean	87
5.8.4	Ejemplos	88
5.9	Cookies	89
5.9.1	La estructura de una cookie	90
5.9.2	Definición de cookies	91
5.9.3	Lectura de cookies	93
5.9.4	Eliminación de cookies	94
5.9.5	Ejemplos	95
5.10	Sesiones	97
5.10.1	Introducción	97
5.10.2	El objeto session	97
5.10.3	Seguimiento de sesión	97
5.10.4	Eliminación de sesión	99
5.10.5	Ejemplos	100
5.11	Expression Language (EL)	101
5.11.1	Objetos implícitos de EL	101

5.11.2	Sintaxis	102
5.11.3	El objeto pageContext	104
5.11.4	Operadores JSP EL.....	104
5.11.5	Palabras reservadas de JSP EL	105
5.11.6	JSP EL: puntos importantes.....	106
5.11.7	Ejemplos.....	106
6	Ejemplos de aplicaciones	108
6.1	formQuiz	108
6.2	formQuiz y Ajax.....	112
6.3	Aplicación “Comentarios”	113
7	Anexo no evaluable.....	113
7.1	Ejecución de comandos externos	113

1 Objetivos y alcance

1.1 Introducción

Los primeros servidores web permitían visualizar exclusivamente información estática. Esto presentó pronto una limitación, sobre todo desde el momento en el que la actividad publicitaria y comercial comenzó a concentrarse también en Internet. La primera solución técnica aportada fue la posibilidad de que el servidor web ejecutase programas residentes en la propia máquina del servidor. Esta tecnología, conocida como *Common Gateway Interface* (**CGI**) permitía lanzar programas escritos principalmente en **C** o **Perl**.

Si bien la tecnología CGI resolvía el problema de la presentación exclusiva de información estática, al mismo tiempo presentaba dos limitaciones importantes: el problema de seguridad que podía representar el hecho de que mediante una petición se pudiesen ejecutar programas indeseados en el servidor y la carga del mismo (si una página que lanzaba un programa era llamada desde 100 clientes simultáneamente, el servidor ejecutaba 100 procesos, uno por cada cliente que solicitaba dicha página).

Para resolver estos problemas, se buscó desarrollar una tecnología que permitiera ejecutar, en un único proceso del servidor, todos los pedidos de ejecución de código sin importar la cantidad de clientes que se conectaban concurrentemente. Así surgieron los denominados **servlets**, basados en la tecnología Java de Sun Microsystems. Éstos permitían ejecutar código en un único proceso externo que gestionaba todas las llamadas realizadas por el servidor web, impidiendo al mismo tiempo que el servidor web pueda ejecutar programas del sistema operativo.

No obstante, aunque de este modo se limitaron los problemas de prestación y seguridad de la tecnología CGI, no se resolvió el problema representado por un desarrollo demasiado costoso en términos de tiempo. Asimismo, se hizo necesario que dos figuras profesionales distintas trabajasen en un único proyecto: el programador (que conoce el lenguaje de programación utilizado del lado del servidor) y el diseñador web (que conoce la parte gráfica y el lenguaje HTML).

Para resolver estas limitaciones, fueron desarrollados lenguajes cuyo código puede ser incluido dentro de archivos HTML. El código puede ser interpretado como, por ejemplo, las páginas **ASP** o **PHP**, o precompilado, como en las páginas **JSP** o **ASP.NET**.

Lenguajes de script de servidor son los lenguajes que se ejecutan en el lado del servidor, como PHP, JSP, ASP, etc. Estos lenguajes se utilizan para generar páginas dinámicas, facilitando, por ejemplo, el acceso a bases de datos.

Actualmente hay una tendencia creciente a ejecutar aplicaciones escritas en ECMAScript en el servidor, lo que se conoce como SSJS (Server-Side JavaScript), y un ejemplo de entorno de ejecución es node.js.

1.2 Objetivo de la práctica

Esta práctica tiene como objetivo que el alumno se familiarice con algún lenguaje de programación web del lado del servidor, en particular, JSP (versión 2.1).

En una práctica posterior, emplearemos los conocimientos adquiridos en esta práctica para ampliar la funcionalidad de nuestras aplicaciones, mediante el acceso a Bases de Datos (BBDD).

1.3 Documentación de apoyo

CGI:

- <http://httpd.apache.org/docs/2.2/howto/cgi.html>
- http://oreilly.com/openbook/cgi/ch01_01.html
- <http://www.ietf.org/rfc/rfc3875>

Servlets y JSP:

- <http://www.oracle.com/technetwork/java/javaee/jsp/index.html>
- <http://www.oracle.com/technetwork/java/javaee/documentation/index.html>
- <http://docs.oracle.com/javaee/>
- “Java servlet & JSP cookbook”. Bruce W. Perry. Editorial: O'Reilly, 2004. ISBN: 0-596-00572-5.
- “Core Servlets and JavaServer Pages Volume 1: Core Technologies, 2nd Edition”. Marty Hall, Larry Brown. Editorial: Prentice Hall, 2003. ISBN: 0-13-009229-0.
- Tomcat: <http://tomcat.apache.org/tomcat-8.0-doc/index.html>

1.4 Preparación del entorno de trabajo

Para acceder a las URLs mediante el protocolo HTTP (en el navegador debe aparecer `http://` y no `file://`), es necesario que el servidor web se encuentre en ejecución (Tomcat hace de servidor web, arrancado desde Eclipse, recuerde cómo hacerlo en las prácticas anteriores).

La interpretación de **Servlets** y páginas **JSP**, también la hace **Tomcat**, que es contenedor de servlets.

IMPORTANTE: Para la realización de la práctica, utilice el usuario **dit**.

DESCARGA DE LOS FICHEROS DE ESTA PRÁCTICA

Nota: recuerde que previamente tiene que recuperar el directorio `/home/dit/workspace` tal como se explicó en la práctica P01.

```
cd /home/dit
rm -r workspace
tar xfvz ./workspace.tar.gz
```

Para los ficheros de esta práctica, acceda a la página Web de la Asignatura en Enseñanza Virtual y vaya a la misma carpeta en la que se encuentra la memoria de esta práctica. En dicha carpeta encontrará el fichero `FAST_t03-ficheros.tar.gz`. Descárguelo a su máquina local, dentro del directorio `/home/dit/`.

Con el usuario "dit", ejecute los siguientes comandos para descomprimir el archivo:

```
cd /home/dit/  
tar xfvz FAST_t03-ficheros.tar.gz  
  
export WEB=/home/dit/workspace/AppWeb/WebContent
```

Mediante el último comando, hemos creado una variable de entorno que nos facilitará el trabajo durante la práctica: desde cualquier posición en la estructura de directorios, podemos ejecutar `cd $WEB` para ir al directorio del proyecto de Eclipse.

El fichero `tar.gz` contiene los subdirectorios y ficheros relacionados con la práctica y que se desplegarán en:

```
/home/dit/workspace/AppWeb/WebContent
```

1.5 Descripción del entorno de trabajo

Durante el desarrollo de la práctica es necesario que utilice el IDE (entorno de desarrollo integrado) Eclipse. Además, al arrancarlo debe escoger el directorio de trabajo (ya descargado en prácticas anteriores):

```
/home/dit/workspace
```

El directorio donde se han de ubicar los ficheros para la publicación de páginas webs a través de Eclipse se encuentra en:

```
/home/dit/workspace/AppWeb/WebContent
```

Desde Eclipse se puede acceder al contenido de estos desde el explorador de proyectos (Project Explorer) dentro de AppWeb:

```
AppWeb/WebContent/
```

Eclipse copia los ficheros editados en ese directorio en el correspondiente de Tomcat (que es quien responde al navegador):

```
/home/dit/tomcat/webapps/AppWeb/
```

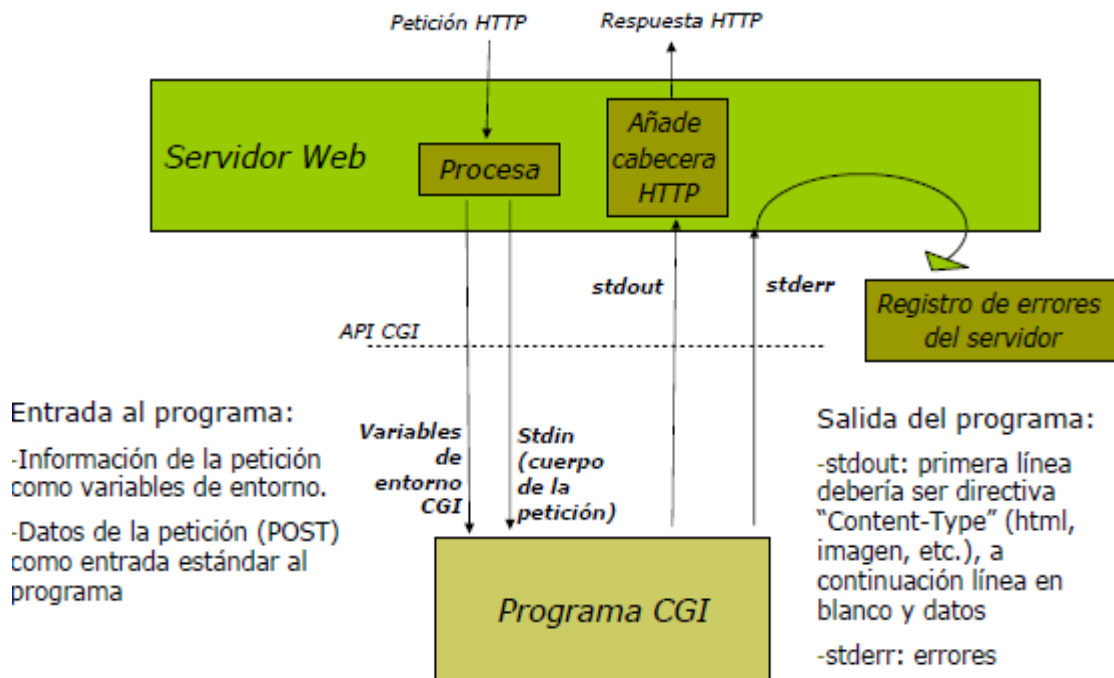
Para acceder al servidor web de Tomcat, la URL a utilizar es:

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


2 CGI

2.1 Introducción

La tecnología CGI permite ejecutar código en la parte del servidor (a diferencia de ECMAScript que se ejecuta en el cliente) de tal forma que cuando llega una petición al servidor web, los datos que vienen en la petición son pasados al programa que se ejecuta (como entrada, variables de entorno, desde qué host se está conectando el usuario, o la entrada que el usuario ha proporcionado utilizando un formulario HTML, etc). Entonces, el programa procesa esos datos y la respuesta por parte del servidor es generada con el resultado de la ejecución del programa y enviada al cliente (navegador del usuario). Los programas pueden ser ejecutables (resultado de compilar un lenguaje de alto nivel) o incluso scripts interpretados (como shell script). En la introducción ya se indicaron los inconvenientes de esta tecnología, y actualmente se usa cada vez menos.



La mayor parte de la petición de un programa CGI es la misma que la petición de cualquier otra página web. La diferencia es que cuando el servidor reconoce que la dirección pedida es un programa CGI, el servidor no devuelve los contenidos del fichero, sino que intenta ejecutar el programa.

La petición del cliente también pasa los formatos de datos que puede aceptar (www/source, text/html, and image/gif, etc.), se identifica con el User-Agent o agente de usuario y envía la información de usuario correspondiente. Toda esta información está disponible desde el programa CGI, junto con información adicional del servidor.

La forma en que los programas CGI consiguen la información de entrada depende del servidor y el sistema operativo del mismo. En un sistema UNIX, como es el caso de las prácticas, los programas CGI consiguen su entrada desde la entrada estándar (STDIN) y desde las variables de entorno UNIX. Estas

variables almacenan esta información: la cadena de búsqueda de entrada (en el caso de un formulario), el formato de la entrada, la longitud de la entrada (en bytes), el host remoto, el usuario que envía la entrada y otra información del cliente. También almacenan el nombre del servidor, el protocolo de comunicación y el software que se ejecuta en el servidor.

Una vez que el programa CGI comienza a ejecutarse, puede crear y enviar su salida a una nueva página web, o proporcionar la URL a una existente. En UNIX, los programas envían su salida a la salida estándar (STDOUT) como un flujo de datos. Este flujo de datos está formado por dos partes: la primera parte es una cabecera HTTP completa o parcial que, como mínimo, describe el formato de los datos devueltos (HTML, texto plano, imagen GIF, etc.). Posteriormente, una línea en blanco que significa el fin de la cabecera. La segunda parte es el cuerpo, que contiene los datos que conforman el tipo de datos reflejado en la cabecera.

Existen dos diferencias principales entre la programación “tradicional” y la programación CGI:

- a) La primera, que toda la salida de un programa CGI debe ser precedida de una cabecera de tipo MIME. Esta es una cabecera que le dice al navegador Web qué tipo de contenido está recibiendo, y suele tener una forma como esta:

```
Content-type: text/html
```

Después de esta cabecera debe haber una línea en blanco (ya observará en los fuentes en C la existencia de esa línea en blanco, que es el resultado de `\n\n`)

- b) La segunda, que la salida ha de ser código HTML, o algún tipo de formato que el navegador sea capaz de manejar (y si no sabe manejarlo da la opción de descargarlo): una imagen, un fichero comprimido, etc.

Aparte de estas dos cosas, escribir un programa CGI es similar a la sintaxis de cualquier otro lenguaje conocido, empleado en el propio CGI.

2.2 Directorios

La mayoría de los servidores esperan que los programas y scripts CGI residan en un directorio especial.

En la configuración de Eclipse-Tomcat, se empleará el siguiente directorio para la publicación de páginas web relacionadas con CGI (páginas HTML que invocan a CGI):

```
/home/dit/workspace/AppWeb/WebContent/ejemplosCGI/
```

Para acceder a esas páginas se utilizará la URL:

```
http://localhost:8080/AppWeb/ejemplosCGI
```

Los ficheros fuente se ubican en un directorio oculto a los clientes. Nótese que no está en WebContent (aunque podría estar en WebContent/WEB-INF). En este caso se ubican dentro del directorio:

```
/home/dit/workspace/AppWeb/src_cgi/
```

Los ficheros ejecutables se ubican en el directorio:

```
/home/dit/workspace/AppWeb/WebContent/WEB-INF/cgi/
```

Los ficheros ejecutables pueden tener cualquier nombre, y pueden ser tanto el resultado de compilar y enlazar ficheros fuente como ficheros shell script, ficheros Python, etc.

El directorio de los ejecutables es el anterior, porque el fichero descriptor de despliegue WEB-INF/web.xml (se verá más adelante) contiene:

```
<param-name>cgiPathPrefix</param-name>
<param-value>WEB-INF/cgi</param-value>
```

Para acceder a los ejecutables se usará la URL:

```
http://localhost:8080/AppWeb/cgi-bin
```

Esto es así porque el fichero WEB-INF/web.xml contiene:

```
<servlet-mapping>
  <servlet-name>cgi</servlet-name>
  <url-pattern>/cgi-bin/*</url-pattern>
</servlet-mapping>
```

Eclipse (cuando lo detecta, refrescando con F5, y si no hay que reiniciar Tomcat) lo copia al directorio correspondiente de Tomcat (pero sin permiso de ejecución):

```
/home/dit/tomcat/webapps/AppWeb/WEB-INF/cgi
```

y en ese directorio hay que volver a darle permiso de ejecución con la orden (por ejemplo):

```
chmod +x *
```

Nota: para facilitar la compilación y modificación de permisos hay disponible un fichero makefile:

```
/home/dit/workspace/AppWeb/makefile
```

Este fichero lo puede usar mediante el comando make desde un terminal o desde Eclipse con la perspectiva C/C++.

Ejemplo de fichero fuente:

```
                                /home/dit/workspace/AppWeb/src_cgi/ejemplo-cgi-1.c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    printf("Content-Type:text/html\n\n");
    printf("<!DOCTYPE html>\n");
    printf("<html>\n<head>\n<meta charset='UTF-8'>\n");
    printf("<title>CGI en c</title>\n");
    printf("</head>\n");
```

```
printf("<body>\n");
printf("<h1>Página Web generada por CGI y programa en c</h1>\n");
printf("<p>Puerto:      %s</p>\n", getenv("SERVER_PORT"));
printf("<p>Datos GET: %s</p>\n", getenv("QUERY_STRING"));
printf("</body>\n");
printf("</html>\n");

return 0;
}
```

- a) Mediante un navegador acceda, para ver su funcionamiento, a:

<http://localhost:8080/AppWeb/ejemplosCGI/index.html>

Y dentro de éste, acceda al primer ejemplo:

`http://localhost:8080/AppWeb/ejemplosCGI/ejemplo-cgi-1.html`

Fíjese en la URL del ejecutable del programa en C:

`/AppWeb/cgi-bin/ejemplo-cgi-1.cgi`

Acceda a esa URL y compruebe que aparece Puerto: 8080 y que Datos GET está vacío.

Acceda ahora añadiendo datos, por ejemplo:

`http://localhost:8080/AppWeb/cgi-bin/ejemplo-cgi-1.cgi?par1=A&par2=B`

Compruebe que Datos GET contiene los datos de la URL

- b) Compruebe el directorio donde está el fichero fuente en Eclipse:

`/home/dit/workspace/AppWeb/src_cgi/ejemplo-cgi-1.c`

Abra al fichero fuente desde Eclipse y estudie su funcionamiento. ¿Qué argumento hay que pasar a `getenv` para obtener los datos de GET?. Ejecute en un terminal `man getenv`.

Compruebe el directorio donde está el ejecutable en Eclipse

`/home/dit/workspace/AppWeb/WebContent/WEB-INF/cgi/ejemplo-cgi-1.cgi`

y el directorio donde está el ejecutable en Tomcat

`/home/dit/tomcat/webapps/AppWeb/WEB-INF/cgi/ejemplo-cgi-1.cgi`

Quite el permiso de ejecución en el ejecutable de Tomcat desde un terminal:

```
dit@localhost:~/tomcat/webapps/AppWeb/WEB-INF/cgi$  chmod  -x
./ejemplo-cgi-1.cgi
```

Compruebe que si ahora intenta acceder desde el navegador mediante HTTP se obtiene un error de permiso denegado y que aparece el directorio de Tomcat donde está el ejecutable. Vuelva a poner el permiso de ejecución y compruebe que ahora sí se puede acceder desde el navegador mediante http.

```
dit@localhost:~/tomcat/webapps/AppWeb/WEB-INF/cgi$  chmod  +x
./ejemplo-cgi-1.cgi
```

Ejecute el programa desde un terminal y observe que la salida que aparece en pantalla es lo que el servidor envía al navegador en la respuesta http.

```
dit@localhost:~/tomcat/webapps/AppWeb/WEB-INF/cgi$  ./ejemplo-
cgi-1.cgi
```

- c) Edite el fichero fuente desde Eclipse para que se inserte después de la cabecera h1 una cabecera h2 con su nombre. Después tiene que compilarlo con gcc (el fuente está en el directorio sources) y el ejecutable debe tener el nombre adecuado (use la opción -o) en el directorio de Eclipse:

```
dit@localhost:~/workspace/AppWeb/WebContent/WEB-INF/cgi$ gcc -g
-w -Wall -o ejemplo-cgi-1.cgi sources/ejemplo-cgi-1.c
```

Para que Eclipse detecte el cambio del ejecutable, debe refrescarlo (F5).

Si ahora intenta acceder desde el navegador obtendrá error de permiso denegado, ya que Eclipse habrá copiado el ejecutable al directorio de Tomcat, pero no tiene permiso de ejecución (si no es así reinicie Tomcat). Para que funcione debe dar permiso de ejecución en Tomcat:

```
dit@localhost:~/tomcat/webapps/AppWeb/WEB-INF/cgi$ chmod +x
./ejemplo-cgi-1.cgi
```

- d) Acceda desde el navegador mediante http y compruebe que aparece su nombre.
e) Aunque no domine el lenguaje Python, acceda a:

```
http://localhost:8080/AppWeb/ejemplosCGI/ejemplo-cgi-1.html
```

y dentro de éste al ejemplo de Python. Observe que en este caso el script (que es un fichero ejecutable) está en:

```
/home/dit/workspace/AppWeb/WebContent/WEB-INF/cgi/ejemplo-cgi-
1.py
```

Abra el script desde Eclipse y observe que su funcionamiento es parecido al programa en C (enviar a la salida lo que tiene que devolver el servidor en la respuesta http). Si modifica el script (por ejemplo añadiendo una cabecera h2 con su nombre) deberá refrescar Eclipse (para que copie a Tomcat) y dar permiso de ejecución en Tomcat (para que no de error de permiso denegado).

2.2.1 Prueba de programas CGI

Si quiere probar el comportamiento de los programas CGI, puede crear las variables de entorno que necesite con la orden export:

```
dit@localhost:~/tomcat/webapps/AppWeb/WEB-INF/cgi$ export QUERY_STRING="par1=A&par2=B"
dit@localhost:~/tomcat/webapps/AppWeb/WEB-INF/cgi$ export SERVER_PORT="2525"
dit@localhost:~/tomcat/webapps/AppWeb/WEB-INF/cgi$ ./ejemplo-cgi-1.cgi
Content-Type:text/html
```

```
<!DOCTYPE html>
<html>
<head>
<meta charset='UTF-8'>
<title>CGI en c</title>
</head>
<body>
<h1>Página Web generada por CGI y programa en c</h1>
<p>Puerto: 2525</p>
<p>Datos GET: par1=A&par2=B</p>
</body>
```

```
</html>
```

Si además necesita leer de la entrada estándar, puede redirigirla desde un fichero (con <).

2.3 Variables CGI

Las variables CGI son definidas por el servidor y “exportadas” al programa CGI. Pueden clasificarse en dos grupos según el tipo de información que contienen:

- a) Datos del servidor web (comunes a todas las peticiones http): SERVER_NAME (host/IP servidor), SERVER_SOFTWARE (implementación servidor), GATEWAY_INTERFACE (versión CGI), ...
- b) Datos relativos al HTTP Request recibido: son de 3 tipos:
 - Meta-variables con información de la petición (generadas por el servidor): SERVER_PROTOCOL (HTTP 1.1, ...), SERVER_PORT (80, 443, ...), REQUEST_METHOD (GET, POST, ...), REMOTE_ADDR (IP del navegador), ...
 - Cabeceras existentes en el mensaje HTTP Request recibido (prefijo “HTTP_Cabecera”): HTTP_USER_AGENT, HTTP_COOKIE, HTTP_ACCEPT_LANGUAGE, ...
 - Datos de usuario (por ejemplo al enviar formularios): QUERY_STRING (GET), CONTENT_TYPE y CONTENT_LENGTH (POST).

TAREAS

- a) Aunque tenga pocos conocimientos de shell script, acceda a:

`http://localhost:8080/AppWeb/ejemplosCGI/ejemplo-cgi-1.html`

y dentro de éste al ejemplo de shell script. Observe que en este caso el script (que es un fichero ejecutable) está en:

`/home/dit/workspace/AppWeb/WebContent/WEB-INF/cgi/ejemplo-cgi-1.sh`

- b) Abra el script desde Eclipse y observe su funcionamiento.
- c) Modifique el fichero fuente de ejemplo-cgi-1.c para que muestre la misma información que el shell script. Para ello recuerde que después de editar y grabar tiene que:
- compilar
 - refrescar Eclipse
 - dar permiso de ejecución en Tomcat (o utilizar el comando `make`)
- d) Modifique el fichero fuente en C para que contenga algunas de las variables que faltan de esta lista (ver más adelante ejemplo mostrar-entorno):
- SERVER_PROTOCOL
 - REMOTE_HOST
 - REMOTE_ADDR
 - REQUEST_URI
 - SERVER_SOFTWARE
 - HTTP_REFERER
 - CONTENT_TYPE
 - REQUEST_METHOD
 - SCRIPT_NAME
 - SERVER_NAME
 - PATH_INFO
 - AUTH_TYPE
 - GATEWAY_INTERFACE
 - SERVER_PORT
 - SCRIPT_FILENAME
 - HTTP_ACCEPT_LANGUAGE
 - REMOTE_IDENT
 - REMOTE_USER
 - HTTP_HOST
 - CONTENT_LENGTH
 - QUERY_STRING
 - HTTP_ACCEPT
 - HTTP_ACCEPT_ENCODING
 - HTTP_COOKIE

2.4 Formularios: método GET

Ahora vamos a ver algunos ejemplos relacionados con formularios HTML, y su interacción con programas CGI. Ya hemos visto en el apartado anterior el paso de parámetros mediante el método GET (recuerde que GET y POST son los métodos disponibles para el paso de información en un formulario HTML desde el lado cliente al servidor).

Este es el contenido de la página `ejemplo-cgi-2.html`:

```
/home/dit/workspace/AppWeb/WebContent/ejemplosCGI/ejemplo-cgi-2.html
<!DOCTYPE html>
<html>
  <head><title>Formulario HTML y CGI</title></head>
  <body>
    <form action="/AppWeb/cgi-bin/ejemplo-cgi-2.out" method="get">
      <div><label>Multiplicando 1: <input name="m" size="5"/></label></div>
      <div><label>Multiplicando 2: <input name="n" size="5"/></label></div>
      <div><input type="submit" value="Multiplica"/></div>
    </form>
  </body>
</html>
```

TAREAS

Abra ahora `ejemplo-cgi-2.c` con el editor y analice su contenido (se encuentra en el subdirectorio `"AppWeb/src_cgi"`, donde se están ubicando los ficheros de código fuente):

```
/home/dit/workspace/AppWeb/src_cgi/ejemplo-cgi-2.c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
  char *datos;
  long m,n;

  printf("Content-Type:text/html\n\n");
  printf("<!DOCTYPE html>\n");
  printf("<html>\n<head>\n");
  printf("<title>Resultado de la multiplicacion</title>\n");
  printf("</head>\n");

  printf("<body>\n");
  printf("<h1>Resultado de la multiplicacion:</h1>\n");
  datos = getenv("QUERY_STRING");

  if(datos == NULL)
    printf("<p>Error pasando datos desde el formulario al prog. CGI</p>");

  else if(sscanf(datos, "m=%ld&n=%ld", &m, &n)!=2)
    printf("<p>Error: los datos deben ser numericos.</p>");

  else
    printf("<h3>El producto de %ld y %ld es %ld.</h3>", m, n, m*n);
```

```
printf("</body>\n");
printf("</html>\n");

return 0;
}
```

Para probar el ejemplo, realice las siguientes acciones:

TAREAS

- Acceda a la URL siguiente, introduzca dos números y compruebe el resultado
`http://localhost:8080/AppWeb/ejemplosCGI/ejemplo-cgi-2.html`
- Abra el fichero con el código fuente y estudie su funcionamiento. Observe la función usada para extraer los parámetros de GET:
`if(sscanf(datos, "m=%ld&n=%ld", &m, &n)!=2)`
Observe la cadena de formato: `"m=%ld&n=%ld"` ¿Qué carácter se utiliza en la cadena de formato para separar un parámetro de otro?
- Modifique el formulario html y el fichero fuente para que ahora multiplique 3 números. Añada también un párrafo al final con el contenido de la cadena `datos`. Recuerde que después hay que compilar, refrescar y dar permiso. ¿Las letras en la cadena de formato deben coincidir con los valores del parámetro `name` de los `input` del formulario html o con los nombres de las variables del programa en C? Haga pruebas para asegurarse.

2.5 Formularios: método POST

Veamos ahora un ejemplo de uso de CGI y el método POST. Cuando se envía un formulario con el método POST, los datos se envían en el cuerpo del mensaje, y estos le llegan al programa a través de la entrada estándar.

Observe el contenido de la siguiente página:

```
/home/dit/workspace/AppWeb/WebContent/ejemplosCGI/ejemplo-cgi-3.html
<!DOCTYPE html>
<html>
<head>
  <title>Formulario HTML y CGI (POST)</title>
  <meta charset="UTF-8">
</head>
<body>
  <form action="/AppWeb/cgi-bin/ejemplo-cgi-3.out" method="post">
    <div>Introduzca un texto (máx. 80 caracteres):<br />
      <input name="data" size="60" maxlength="80" />
      <input type="submit" value="Enviar" />
    </div>
  </form>
</body>
```

```
</html>
```

Y examine también el contenido del siguiente código escrito en lenguaje C:

/home/dit/workspace/AppWeb/src_cgi/ejemplo-cgi-3.c

```
#include <stdio.h>
#include <stdlib.h>

//Tamano maximo del mensaje introducido en el formulario HTML:
#define MAXLEN 80

//EXTRA: valor del campo "data" del formulario mas "="
#define EXTRA 5

// Datos de la peticion
#define MAXINPUT MAXLEN+EXTRA

//Fichero de salida
#define DATAFILE "/home/dit/datos3.txt"

//Funcion para convertir caracteres no alfanumericos dentro de codigo HTML.
//Codigos HTML (hexadecimal): http://www.asci.cl/htmlcodes.htm
void descodifica(char *cini, char *cfin, char *dest);

int main(void)
{
    char *lenstr;
    char input[MAXINPUT+1];
    char data[MAXLEN+1];
    long len;
    FILE *fout;

    printf("Content-Type:text/html\n\n");
    printf("<!DOCTYPE html>\n");
    printf("<html>\n<head>\n");
    printf("<title>Almacen de datos</title>\n");
    printf("</head>\n<h1>Almacen de datos</h1>\n");
    printf("<body>\n");

    //Obtenemos la longitud de la peticion POST:
    lenstr = getenv("CONTENT_LENGTH");

    if(lenstr == NULL || sscanf(lenstr,"%ld",&len)!=1 || len > MAXINPUT)
        printf("<p>Error en la invocacion: revise el formulario.</p>");

    else
    {
        fgets(input, len+1, stdin);

        //Obtenemos el mensaje descodificado a partir de "data"...
        descodifica(input+EXTRA, input+len, data);

        fout = fopen(DATAFILE, "w");

        if(fout == NULL)
            printf("<p>No ha sido posible almacenar los datos.</p>");
    }
}
```

```

else
{
    //Almacenamos el mensaje en el fichero de salida:
    fprintf(fout, "*****\nDatos codificados\n*****\n");
    fprintf(fout, "%s\n", input);

    fprintf(fout, "*****\nDatos
descodificados\n*****\n");
    fprintf(fout, "%s\n", data);

    printf("<p>Datos almacenados.</p>");
    fclose(fout);
}
}

printf("</body>\n");
printf("</html>\n");

return 0;
}

//Funcion para convertir caracteres no alfanumericos dentro de codigo HTML.
void descodifica(char *cini, char *cfin, char *dest)
{
    unsigned int cod;

    for(; cini < cfin; cini++, dest++)
    {
        if(*cini == '+')
            *dest = ' ';

        else if(*cini == '%') {

            if(sscanf(cini+1, "%2x", &cod) != 1)
                cod = '?';

            *dest = cod;
            cini +=2;
        }
        else
            *dest = *cini;
    }
    *dest = '\0';
}


```

El programa recibe información del número de caracteres del mensaje POST, a través de la variable de entorno “CONTENT_LENGTH”. La función “descodifica” transforma los códigos HTML en texto plano. Puede consultar <http://www.w3.org/TR/REC-html40/interact/forms.html#form-content-type>, donde dice que para el tipo de contenido por defecto (application/x-www-form-urlencoded):

- Los caracteres espacio se sustituyen por el carácter `+`
- Los caracteres no alfanuméricos se reemplazan por `% HH`, es decir, un signo de porcentaje y dos dígitos hexadecimales que representan el código ASCII del carácter

El mensaje introducido en el campo “data” del formulario es almacenado en el fichero /home/dit/datos3.txt a través del programa CGI ejemplo-cgi-3.out.

Si ejecuta el ejemplo con el siguiente mensaje:



Obtendrá el siguiente contenido en el fichero /home/dit/datos3.txt:

```
dit@localhost:~$ cat datos3.txt
*****
Datos codificados
*****
data=Esto+es+una+prueba%21%21+%2B%23

*****
Datos descodificados
*****
Esto es una prueba!! +#
```

Para probar el ejemplo, acceda al formulario, introduzca la cadena

Esto es una prueba!! +#

y desde un terminal muestre el contenido del fichero /home/dit/datos3.txt

TAREAS

- a) Abra el fichero ejemplo-cgi-3.c y estudie el código y la función descodifica.
- b) ¿De qué tipo es la variable lenstr? ¿Qué valor se le asigna?
- c) La sentencia `sscanf(lenstr, "%ld", &len) != 1` ¿qué hace?
- d) Cuando se ejecuta la sentencia `fgets(input, len+1, stdin)`; ¿de dónde se leen los valores para asignárselos a input? Recuerde que se está procesando una petición POST y que el segundo argumento de fgets debe ser uno más de los caracteres a leer ya que fgets añade el '\0'.
- e) Cuando se llama a descodifica con `descodifica(input+EXTRA, input+len, data)`; se va a analizar la cadena input desde input+EXTRA hasta input+len, y el resultado se va a almacenar en data ¿por qué se empieza por input+EXTRA, teniendo en cuenta que EXTRA vale 5?
- f) Dentro del bucle de la función descodifica cuando en cini (cadena input):
 - se encuentra el carácter '+', ¿en qué se convierte en dest (cadena data)?
 - se encuentra el carácter '%', ¿en qué se convierte en dest (cadena data)? ¿por qué se usa "%2x" como cadena de formato en sscanf? ¿por qué se empieza en cini+1?

```
if(sscanf(cini+1, "%2x", &cod) != 1)
    cod = '?';

    *dest = cod;
    cini +=2;
```

- no se encuentra ni el carácter '+' ni el '%' ¿qué hace?
- g) Al final de la función descodifica ¿qué hace esta instrucción?


```
*dest = '\0';
```
 - h) Añada dos sentencias printf para que en el navegador aparezcan dos párrafos con el contenido de input y data. Recuerde que después hay que compilar, refrescar y dar permiso. Utilice las siguientes cadenas de formato:


```
"<p>Datos codificados: %s</p>"
"<p>Datos descodificados: %s</p>"
```
 - i) Pruebe a introducir en el formulario los siguientes caracteres:


```
!"#$%&'()
```

 y compruebe que se codifican con el carácter % seguido del valor en hexadecimal correspondiente a su código ascii. Puede consultar los códigos ascii escribiendo en un terminal `man ascii` o por ejemplo en:


```
http://www.ascii-code.com/
```

 ¿cómo se ha codificado el carácter %? ¿cuál es su código ascii en decimal y hexadecimal?

2.6 Formularios: varios campos en conjunto

En este caso, se va a estudiar un formulario HTML, que mediante el método POST envía datos al servidor: nombre, apellidos y edad del alumno. Dichos datos son enviados a un programa CGI escrito en C, el cual guarda los datos a un fichero, llamado `datos_alumno.txt`, dentro de `/home/dit/`. El script CGI devolverá al navegador del cliente “*Datos escritos correctamente*” o “*Error de escritura de datos*”.

`/home/dit/workspace/AppWeb/WebContent/ejemplosCGI/ejemplo-cgi-4.html`

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <title>Ejemplo 4: formulario HTML y CGI en C</title>
</head>
<body>
  <form action="/AppWeb/cgi-bin/ejemplo-cgi-4.out" method="POST">
    <div><label>Nombre:   <input name="nombre"   size="20"></label></div>
    <div><label>Apellidos:<input name="apellidos" size="30"></label></div>
    <div><label>Edad:    <input name="edad"      size="2"></label></div>
    <div><input type="submit" value="Enviar"></div>
  </form>
</body>
</html>
```

`/home/dit/workspace/AppWeb/src_cgi/ejemplo-cgi-4.c`

```
#include <stdio.h>
#include <stdlib.h>
#define MAXLEN 80
#define DATAFILE "/home/dit/datos_alumno.txt"

int main(void)
{
  char *lenstr;
  char input[MAXLEN+1];
  long len;

  FILE *fout;
  int error = 0;

  printf("Content-Type:text/html\n\n");
  printf("<html>\n<head>\n");
  printf("<title>Ejercicio 2: Formulario CGI y Leng. C</title>\n");
  printf("</head>\n<body>\n");

  lenstr = getenv("CONTENT_LENGTH");

  if(lenstr == NULL || sscanf(lenstr,"%ld",&len)!=1 || len > MAXLEN)
    printf("<p>Error en formulario.</p>");

  else {
    if ((fout = fopen(DATAFILE, "w")) == NULL)
    {
      printf("<p>Error de escritura de datos.</p>\n");
      error=1;
    }
  }
```

```

    else {

        fgets(input, len+1, stdin);
        fputs(input, fout);

        fclose(fout);
        printf("<p>Datos escritos correctamente.</p>\n");
        error=0;
    }
}

printf("</body>\n");
printf("</html>\n");

return error;
}

```

TAREAS

- Estudie el código de los ficheros ejemplo-cgi-4.html y ejemplo-cgi-4.c.
- Añada una sentencia printf para que en el navegador aparezca un párrafo con el contenido de input. Recuerde que después hay que compilar, refrescar y dar permiso. Utilice la siguiente cadena de formato:
"<p>Datos codificados: %s</p>"
- Introduzca un valor en cada campo y observe en la codificación la separación de los campos.
- Introduzca dos apellidos separados por espacios en el campo apellidos y compruebe cómo se codifica la separación de los apellidos.

2.7 Formularios: varios campos por separado

En este caso, se va a estudiar un formulario HTML, que mediante el método POST envía el contenido de dos campos de 80 caracteres máximo (campos "data1" y "data2"):

```

/home/dit/workspace/AppWeb/WebContent/ejemplosCGI/ejemplo-cgi-5.html
<!DOCTYPE html>
<html>
<head>
<title>Formulario HTML y CGI (POST)</title>
<meta charset="UTF-8" />
</head>
<body>
<form action="/AppWeb/cgi-bin/ejemplo-cgi-5.out" method="POST">
  <div>Introduzca un texto (máx. 80 caracteres):<br />
  <input name="data1" size="60" maxlength="80" />
</div>
  <div>Introduzca un texto (máx. 80 caracteres):<br />
  <input name="data2" size="60" maxlength="80" />
</div>
  <input type="submit" value="Enviar">
</form>
</body>
</html>

```


Los datos son enviados a un programa CGI escrito en C, el cual separa los dos campos y guarda los datos en un fichero /home/dit/datos5.txt. El script CGI devuelve al navegador del cliente **“Datos escritos correctamente”** o **“Error de escritura de datos”**.

/home/dit/workspace/AppWeb/src_cgi/ejemplo-cgi-5.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//Tamano maximo del mensaje introducido en el formulario HTML:
#define MAXLEN 80

//EXTRA: valor del campo "dataX" del formulario mas "="
#define EXTRA 6

// Datos del formulario (uno de los campos)
#define MAXINPUT MAXLEN+EXTRA
// Datos del formulario (dos campos + separador)
#define TOTALINPUT (MAXLEN+EXTRA)*2+1

//Fichero de salida
#define DATAFILE "/home/dit/datos5.txt"

//Funcion para convertir caracteres no alfanumericos dentro de codigo HTML.
//Codigos HTML (hexadecimal): http://www.ascii.cl/htmlcodes.htm
void descodifica(char *cini, char *cfin, char *dest);

int main(void)
{
    char *lenstr;
    char input[TOTALINPUT+1];
    char input_orig[TOTALINPUT+1];
    char data1[MAXLEN+1];
    char data2[MAXLEN+1];
    long len;
    FILE *fout;
    char *p;

    printf("Content-Type:text/html\n\n");
    printf("<html>\n<head>\n");
    printf("<title>Almacen de datos</title>\n");
    printf("</head>\n<h1>Almacen de datos</h1>\n");
    printf("<body>\n");

    //Obtenemos la longitud de la peticion POST:
    lenstr = getenv("CONTENT_LENGTH");

    if(lenstr == NULL || sscanf(lenstr,"%ld",&len)!=1 || len > TOTALINPUT)
        printf("<p>Error en la invocacion: revise el formulario.</p>");

    else
    {
        fgets(input, len+1, stdin);
        strcpy(input_orig, input);
```

```

//Obtenemos el mensaje descodificado a partir de "data1"...
p = strtok (input, "&");
descodifica(p+EXTRA, p+strlen(p), data1);

//Obtenemos el mensaje descodificado a partir de "data2"...
p = strtok (NULL, "&");
descodifica(p+EXTRA, p+strlen(p), data2);

fout = fopen(DATAFILE, "w");

if(fout == NULL)
printf("<p>No ha sido posible almacenar los datos.</p>");
else
{
    //Almacenamos el mensaje en el fichero de salida:
    fprintf(fout, "*****\nDatos
codificados\n*****\n");
    fprintf(fout, "%s\n", input_orig);

    fprintf(fout, "*****\nDatos
descodificados\n*****\n");
    fprintf(fout, "data1: %s\n", data1);
    fprintf(fout, "data2: %s\n", data2);

    printf("<p>Datos almacenados.</p>");
    fclose(fout);
}

}

printf("</body>\n");
printf("</html>\n");

return 0;
}

//Funcion para convertir caracteres no alfanumericos dentro de codigo HTML.
void descodifica(char *cini, char *cfin, char *dest)
{
    unsigned int cod;

    for(; cini < cfin; cini++, dest++)
    {
        if(*cini == '+')
            *dest = ' ';

        else if(*cini == '%') {

            if(sscanf(cini+1, "%2x", &cod) != 1)
                cod = '?';

            *dest = cod;
            cini +=2;
        }
        else
            *dest = *cini;
    }
    *dest = '\0';
}

```

TAREAS

- a) Estudie el código de los ficheros ejemplo-cgi-5.html y ejemplo-cgi-5.c.
- b) Añada tres sentencias printf para que en el navegador aparezca un párrafo con el contenido de input_orig, data1 y data2. Recuerde que después hay que compilar, refrescar y dar permiso. Utilice como cadena de formato:

```
"<p>Datos originales: %s</p>"
```

```
"<p>data1: %s</p>"
```

```
"<p>data2: %s</p>"
```

- c) ¿Por qué en este programa EXTRA vale 6?
- d) Averigüe cómo se separan los datos de un campo respecto a otro. Consulte man strtok para entender las sentencias:

```
p = strtok (input, "&");
```

```
p = strtok (NULL, "&");
```

- e) Introduzca un valor en cada campo y observe en la codificación la separación de los campos.
- f) Introduzca dos valores separados por espacios en cada campo y compruebe cómo se codifica la separación de los apellidos.
- g) Modifique el formulario y el programa para que ahora sean tres campos.
- h) Modifique el programa para usar bucles en vez de la función strtok.

2.8 Ejemplo mostrar-entorno

En este caso, se va a estudiar un programa que muestra las variables de entorno y la entrada estándar.

/home/dit/workspace/AppWeb/src_cgi/mostrar-entorno.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[], char *env[])
{
    int i;

    printf("Content-Type:text/html\n\n");
    printf("<!DOCTYPE html>\n");
    printf("<html>\n<head>\n");
    printf("<meta charset='UTF-8'>");
    printf("<title>Información de argumentos y entorno</title>\n");
    printf("</head>\n");

    printf("<body>\n");

    printf("<h1>Argumentos:</h1>\n");
    i=0;
    while (i<argc)
    {
        printf("<p>%d: %s</p>\n", i, argv[i]);
        i++;
    }

    printf("<h1>Variables de entorno:</h1>\n");
    i=0;
```

```
while (env[i]!=NULL)
{
    printf("<p>%d: %s</p>\n", i, env[i]);
    i++;
}

printf("<h1>Entrada estándar:</h1>\n<pre>\n");
char *lenstr = getenv("CONTENT_LENGTH");
if (lenstr!=NULL && strlen(lenstr)!=0 )
{
    while ( (i = getchar()) != EOF )
        putchar(i);
}
printf("</pre>\n");
printf("</body>\n");
printf("</html>\n");

return 0;
}
```

El ejecutable de este programa se llama `mostrar-entorno.html`, y está en el directorio correspondiente a los ejecutables.

TAREAS

- Estudie el código.
- Cree un fichero html con un formulario para que envíe datos y se reciban en el programa a través de variable de entorno.
- Cree un fichero html con un formulario para que envíe datos y se reciban a través de la entrada estándar.
- Modifique el programa para que muestre también el número de caracteres recibidos.

3 J2EE

3.1 Introducción

Las distintas plataformas de desarrollo en Java son las siguientes:

- Java SE (Standard Edition), para desarrollo de aplicaciones convencionales (visto en el primer curso del grado).
- Java EE** (Enterprise Edition), para desarrollo de aplicaciones empresariales basadas en Web.
- Java ME (Micro Edition), para desarrollo de aplicaciones en dispositivos móviles.

Haremos foco en Java EE, pues es el indicado para lograr nuestro objetivo de desarrollar aplicaciones web, en este caso, de aplicaciones web dinámicas del lado del servidor.

A la hora de desarrollar aplicaciones Java EE, hay que pensar en términos de escalabilidad y el número de usuarios potenciales, pues la aplicación (y el servidor que la ofrece) debe estar preparada para soportar la demanda de peticiones de los usuarios.

3.2 Arquitectura de Java EE

Puede consultar información adicional en:

- <https://docs.oracle.com/javaee/7/tutorial/index.html>
- <http://docs.oracle.com/javaee/6/tutorial/doc/>
- <http://docs.oracle.com/javaee/5/tutorial/doc/>

Durante esta práctica y la siguiente nos centraremos en el desarrollo de aplicaciones web dinámicas del lado del servidor, basadas en Servlets y JSP con consultas a BBDD mediante JDBC. El acceso a dichas aplicaciones será originado por peticiones desde el navegador web de un usuario.

3.3 Contenedor Web: Tomcat

Como ya hemos visto en prácticas anteriores, un navegador web (Web Browser, en nuestro caso FireFox) ofrece una interfaz gráfica de usuario (GUI) que permite la petición/recepción de recursos (normalmente la visualización de documentos HTML sin/con contenido dinámico en el lado cliente, como ECMAScript) con su contenido visual y auditivo, utilizando el protocolo cliente-servidor HTTP.

El componente del lado del servidor de HTTP es conocido como el servidor web (Web Server, en nuestro caso el servidor web de Tomcat), y su función primordial es recuperar y transmitir recursos al cliente.

Cuando el servidor web recibe peticiones HTTP solicitando una página JSP o un Servlet, el contenedor web (Web Container, en nuestro caso Tomcat) es el encargado de ejecutar el código Java.

Apache Tomcat (también llamado **Jakarta Tomcat** o simplemente **Tomcat**) funciona como un contenedor de Servlets, desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat es una implementación en código abierto de las especificaciones de Sun Microsystems de:

- Java Servlets
- Java Server Pages (JSP)
- Java Expression Language
- Java WebSocket (fuera del alcance de la práctica)

Tomcat incluye un compilador, que **compila páginas JSPs convirtiéndolas en servlets**. El motor de servlets de Tomcat puede funcionar en combinación con otro servidor web, (genérico), aunque también puede funcionar como servidor web por sí mismo. Dado que Tomcat está escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

Hay otros productos más complejos que Tomcat para dar soporte a todas las posibilidades de JEE, son los llamados servidores de aplicaciones (que normalmente incluyen un contenedor de servlets). Un ejemplo es JBoss.

En la máquina virtual de la asignatura, **Tomcat** trabaja también como **servidor web** para servir las páginas de las aplicaciones web Java que desarrollaremos, se arranca desde **Eclipse** y escucha en el **puerto 8080**.

3.4 Descripción del entorno de trabajo

Se va a usar Tomcat como servidor web y contenedor de servlets desde el entorno de Eclipse, y el funcionamiento ya se ha explicado en 1.5

Dentro de los directorios de la aplicación web, el directorio WEB-INF es privado (no es accesible mediante http). En Eclipse el directorio asociado es:

```
/home/dit/workspace/AppWeb/WebContent/WEB-INF
```

En Tomcat el directorio asociado es (es donde Eclipse lo copia):

```
/home/dit/tomcat/webapps/AppWeb/WEB-INF
```

Desde Eclipse se puede acceder al contenido desde el explorador de proyectos (Project Explorer) dentro de AppWeb:

```
AppWeb/WebContent/WEB-INF
```

Si se quiere recargar las aplicaciones de Tomcat (para no tener que reiniciarlo), hemos de acceder a la web de gestión de Tomcat: <http://localhost:8080/manager/>, con el usuario "dit" y clave "dit". Una vez allí (ver figura siguiente), es posible recargar la aplicación correspondiente a "/AppWeb". En el caso de que sea necesario hacerlo, se le pedirá.

Gestor de Aplicaciones Web de Tomcat

Mensaje: OK

Gestor

[Listar Aplicaciones](#) [Ayuda HTML de Gestor](#) [Ayuda de Gestor](#) [Estado de Servidor](#)

Trayectoria	Versión	Nombre a Mostrar	Ejecutándose	Sesiones	Comandos
/	Ninguno especificado	Welcome to Tomcat	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/AppWeb	Ninguno especificado	Ejemplos de FAST	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/docs	Ninguno especificado	Tomcat Documentation	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/examples	Ninguno especificado	Servlet and JSP Examples	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos
/host-manager	Ninguno especificado	Tomcat Host Manager Application	true	0	Arrancar Parar Recargar Replegar Expirar sesiones sin trabajar ≥ 30 minutos

4 Servlets

4.1 Introducción

Los Servlets realizan tareas similares a las de los programas accedidos mediante CGI, pero se ejecutan en un entorno diferente, en concreto dentro de un contenedor de Servlets. En nuestro caso, será ejecutado dentro de Tomcat y su resultado es presentado al usuario mediante el navegador (que recibe la respuesta enviada por el servidor web del propio Tomcat). Su código es Java, que se corresponde con una clase que debe derivar de la clase abstracta `javax.servlet.http.HttpServlet`. Consulte esta clase en:

<https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html>

Estas clases no tienen método `main()`, (ya que el método `main()` está en el contenedor, es decir, en Tomcat) pero sí tienen los métodos `doGet()` o `doPost()`, que son ejecutados cuando se recibe la petición GET o la petición POST. Uno de estos métodos es invocado por el método `service()`, que se ejecuta en un hilo cada vez que llega una petición. En cambio el método `init()` se invoca sólo cuando llega la primera petición o en el despliegue de la aplicación, y el método `destroy()` se invoca sólo en el repliegue.

4.2 Ejemplo

Veamos un código de ejemplo de un servlet.

Dentro de Eclipse los servlets están en:

```
/home/dit/workspace/AppWeb/src/
```

Y son accesibles a través del explorador de proyectos mediante:

```
AppWeb > Java Resources > src
```

En este caso, al estar `ServletFecha.java` en el paquete `fast`:

```
/home/dit/workspace/AppWeb/src/fast/ServletFecha.java
```

```
AppWeb > Java Resources > src > fast > ServletFecha.java
```

AppWeb/src/fast/ServletFecha.java

```
package fast;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class ServletFecha extends HttpServlet {

    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException {
        PrintWriter salida = response.getWriter();
        java.util.Date hoy = new java.util.Date();

        salida.println("<!DOCTYPE html>");
        salida.println(
            "<html><body><div style='text-align:center'>" +
            "<h1>Ejemplo1 de servlet</h1><br/>" +
            "<p> La fecha actual es: " +
            hoy +
            "</p>" +
            "</div>" +
            "</body></html>");
    }
}
```

Como se puede apreciar en el código no existe el método `main()`, y lo que se ha hecho es sobrescribir el método `doGet()`, y también se podría haber sobrescrito el método `doPost()`, `init()` y `destroy()`. El que NO se debe sobrescribir es el método `service()` (si se quiere que se ejecute el mismo código con GET y con POST, `doPost()` puede llamar a `doGet()`).

Los tipos de los parámetros de `doGet()` son interfaces, que puede consultar en:

<https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html>
<https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html>

TAREAS

- a) Mediante un navegador acceda, para ver su funcionamiento, a:

<http://localhost:8080/AppWeb/servlets/ServF>

Use las herramientas de desarrollador para inspeccionar el código HTML (p. ej. con F12). Acceda fichero ServletFecha.java desde Eclipse (tal como se ha indicado anteriormente) y observe que se corresponde con el código que se ha ejecutado, y que la salida es lo que llega al navegador.

- b) Acceda al siguiente fichero desde Eclipse:

AppWeb/WebContent/WEB-INF/web.xml

y busque la relación entre el fragmento de la URL a partir del alias:

`/servlets/ServF`

y el nombre de la clase servlet cuyo código se ejecuta (fichero ServletFecha.java):

`ServletFecha`

- c) ¿El nombre interno que las relaciona aparece en el fichero java, en la URL o en ninguno de los dos?

4.3 Descriptor de despliegue

Como ha podido observar al realizar la tarea anterior, el fichero web.xml, llamado Descriptor de Despliegue (*Deployment Descriptor*, **DD**), informa al contenedor cómo ejecutar los servlets (y también JSPs). El DD tiene diversas funciones, una de las funciones principales es relacionar URLs con servlets: una relación para el nombre de la URL pública conocida por los clientes hacia el nombre interno, y otra relación para el nombre interno hacia el nombre de la clase. Veamos un fragmento:

fragmento de ... AppWeb/WebContent/WEB-INF/web.xml

```
<servlet>
  <servlet-name>ServletEj1</servlet-name>
  <servlet-class>ServletFecha</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>ServletEj1</servlet-name>
  <url-pattern>/servlets/ServF</url-pattern>
</servlet-mapping>
```

En general, se cumple lo siguiente para un DD:

- Puede haber un DD por aplicación web, el nombre del fichero debe ser **web.xml** y estar en el directorio adecuado (WEB-INF).
- Un DD puede declarar varios servlets.

- Un `<servlet-name>` contiene el nombre interno, y relaciona el elemento `<servlet>` con el elemento `<servlet-mapping>`.
- Dentro de `<servlet>`, un `<servlet-class>` es la clase Java del servlet.
- Dentro de `<servlet-mapping>`, un `<url-pattern>` es el nombre del recurso (URL) que el cliente emplea para la petición.

Así, cuando desde el cliente se accede a la URL:

```
http://localhost:8080/AppWeb/servlets/ServF
```

la petición HTTP que se realiza al servidor web de Tomcat es (a través de una conexión TCP al puerto 8080 de la dirección 127.0.0.1):

```
GET /AppWeb/servlets/ServF
```

Tomcat interpreta el alias `/AppWeb`, y lo que queda es:

```
/servlets/ServF
```

Gracias al DD visto anteriormente, la clase del servlet asociado es (relacionado a través del nombre interno `ServletEj1`):

```
ServletFecha
```

TAREAS

- a) Acceda al Descriptor de Despliegue desde Eclipse:

AppWeb/WebContent/WEB-INF/web.xml

- b) Modifíquelo para que la URL a la que haya que acceder desde el navegador para que se ejecute el Servlet `ServletFecha` sea:

<http://localhost:8080/AppWeb/servlets/Corto>

Nota: recuerde grabar la modificación. Si aun así no obedece, pruebe a recargar la aplicación `/AppWeb` a través de <http://localhost:8080/manager/> tal como se explicó en el apartado “Descripción del entorno de trabajo”.

4.4 Estructura de directorios

La estructura de directorios usada en Eclipse para el servlet de ejemplo (aparece el directorio `fast` dentro de `src` porque pertenece al paquete `fast`) es la siguiente:

```
/home/dit/workspace/AppWeb/
```

```
| -src
|   |-fast
|       |-ServletFecha.java
|-WebContent
|   |-WEB-INF/
|       |-web.xml
```

La estructura de directorios usada en tomcat:

```
/home/dit/tomcat/webapps/AppWeb/
|   |-WEB-INF/
|       |-classes
|           |-fast
|               |-ServletFecha.class
|   |-web.xml
```

4.5 Compilación del Servlet

A continuación, vamos a modificar el fichero `ServletFecha.java` y generar un nuevo fichero `ServletFecha.class`.

TAREAS

- Use la estructura de directorios del apartado anterior y compruebe que los ficheros `ServletFecha.java` y `Servlet.class` están sus directorios correspondientes.
- Compruebe que la fecha de modificación del `.class` es posterior al `.java`.
- Edite el fichero `.java` y añada su nombre al texto de la etiqueta `h1`.
- Grabe la modificación y vuelva a acceder al Servlet desde el navegador y compruebe que aparece su nombre (puede que tenga que esperar algunos segundos). Esto funciona ya que Eclipse ha detectado el cambio en el fichero java, lo ha compilado, y ha copiado el fichero class en el directorio adecuado de tomcat (si no se usara Eclipse habría que compilarlo manualmente con `javac`).
- Compruebe que se ha generado un nuevo `.class` y que su fecha de modificación es posterior al `.java`.

4.6 Ciclo de vida

Recordemos (visto en teoría) cómo el contenedor maneja una petición:

- El usuario accede a una URL que apunta a un servlet, en lugar de a una página estática:
- El contenedor ve que la petición es para un servlet, y si es la primera petición desde el despliegue (y no se instanció en el despliegue), se instancia un objeto de la clase del servlet y se ejecuta su método `init()`.
- El contenedor crea dos objetos, petición (request) y respuesta (response):
 - request: objeto que implementa la interfaz `HttpServletRequest` (subclase de `ServletRequest`)
 - contiene un input stream donde se pasa la información del cliente al servlet.

- incluye parámetros, protocolo, identificaciones del cliente y del servidor, etc.
 - response: objeto que implementa la interfaz `HttpServletResponse` (subclase de `ServletResponse`)
 - contiene un output stream donde se debe almacenar la información que se desea enviar al cliente.
- 4) El contenedor encuentra el servlet correcto basándose en la URL de la petición, crea o reserva un hilo para dicha petición, y pasa los objetos de petición y respuesta al método `service()` del servlet para que se ejecute en ese hilo. Dependiendo del tipo de petición, el método `service` llama a su vez al método `doGet()` o a `doPost()`.
- 5) Asumiremos que la petición es de tipo HTTP GET: El método `doGet()` genera la página dinámica y la introduce dentro del objeto de respuesta, puesto que el contenedor aún tiene una referencia al objeto de respuesta.
- 6) El hilo completa su ejecución, el contenedor convierte el objeto de respuesta en una respuesta HTTP, que se envía de vuelta al cliente, y entonces borra los objetos de petición y respuesta.
- 7) Cuando se repliega la aplicación web, el contenedor ejecuta el método `destroy()` de cada uno de los objetos instanciados.

Para comprobar que sólo hay un objeto instanciado del Servlet, y que sólo hay un proceso que es el de tomcat, realice la siguiente tarea:

TAREAS

- a) Averigüe el pid del proceso de tomcat, por ejemplo con:

```
ps ax | grep tomcat
```

- b) Acceda desde el navegador al siguiente Servlet:

<http://localhost:8080/AppWeb/servlets/Largo>

- c) Compruebe que el valor numérico del PID que aparece bajo la fecha coincide con el pid de tomcat. Acceda varias veces y compruebe que no cambia (esta es una de las diferencias con CGI).
- d) Acceda desde otra ventana del navegador al mismo Servlet y compruebe que el PID sigue siendo el mismo. El valor del ID del hilo podrá ir cambiando. Si no cambia, espere entre 5 y 10 segundos antes de volver a acceder (los hilos pueden ser reutilizados por Tomcat).
- e) Busque, usando el fichero `web.xml`, la clase de ese Servlet y compruebe que su nombre coincide con el valor devuelto por `getClass()`.
- f) Consulte en la documentación de java el método `hashCode()` de `Object`:

<http://docs.oracle.com/javase/8/docs/api/java/lang/Object.html>

- g) Compruebe que en los distintos accesos el valor devuelto por `hashCode()` es el mismo, ya que se trata del mismo objeto (sólo se instancia una vez).
- h) Usando el gestor de aplicaciones de Tomcat, recargue la aplicación web /AppWeb como se ha explicado anteriormente (<http://localhost:8080/manager/html>).
- i) Compruebe que el PID sigue siendo el mismo (Tomcat no terminó, simplemente recargó la aplicación).
- j) Compruebe que el valor devuelto por `hashCode()` ahora es distinto al devuelto antes de recargar. Esto es así porque en el repliegue se eliminó el objeto y el nuevo despliegue se ha vuelto a instanciar.

4.7 Anotaciones

Ya se ha visto que con el fichero WEB-INF/web.xml se puede asociar una URL con un servlet.

Una forma más simple de asociar una URL con un servlet es usando anotaciones en el servlet. Las anotaciones también permiten otras acciones, como crear parámetros de inicialización.

Con la anotación

```
@WebServlet(urlPatterns={"/test", "*.test"})
```

se consigue un resultado equivalente a usar `<servlet-mapping>`, con `<url-pattern>` dentro del descriptor de despliegue (WEB-INF/web.xml). Con esa anotación previa al comienzo de la clase del servlet, cuando llegue una petición que sea `/AppWeb/test` o cualquiera que empiece por `/AppWeb` y termine en `“.test”` (por ejemplo `/AppWeb/lo/que/sea.test`), será servida por el servlet que tiene la anotación.

Se puede acceder desde Eclipse al siguiente servlet de ejemplo

AppWeb > Java Resources > src > fast.anotaciones > TestServlet.java

Observe las anotaciones:

AppWeb/src/fast/anotaciones/TestServlet.java

```
package fast.anotaciones;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.annotation.WebServlet;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
//import javax.servlet.annotation.WebInitParam;

/**
 * @author dit
 *
 */
//Lo más sencillo es poner solo:
@WebServlet(urlPatterns={"/test", "*.test"})
//@WebServlet(name = "testServlet", description = "Ejemplo de Servlet con
anotaciones", urlPatterns = {
//    "/test" }, initParams = { @WebInitParam(name = "n1", value =
"v1"), @WebInitParam(name = "n2", value = "v2") })
public class TestServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest req,
        HttpServletResponse resp) throws ServletException, IOException {
        doPost(req, resp);
    }
}
```

```

    }

    @Override
    protected void doPost(HttpServletRequest req,
        HttpServletResponse resp) throws ServletException, IOException {
        PrintWriter salida = resp.getWriter();

        salida.println("<!DOCTYPE html>");
        salida.println("<html><body><div>" +
            "<h1>Test con anotaciones</h1><br/>" +
            "<p>Así no hace falta tener que modificar el fichero
web.xml</p>" +
            "</div>" +
            "</body></html>");
    }
}

```

TAREAS

- a) Acceda desde el navegador a

<http://localhost:8080/AppWeb/test>
<http://localhost:8080/AppWeb/cualquier/cosa.test>

- b) Compruebe que el resultado es el mismo y coincide con la salida de TestServlet
 c) Ponga su nombre en un encabezado h2 (después de h1). Grabe y vuelva a acceder mediante http para ver el resultado.
 d) Compruebe que /test y *.test están asociados a TestServlet accediendo en Eclipse a:

AppWeb > Deployment Descriptor > Servlet Mappings

- e) Acceda a la clase TestServlet haciendo doble click en Eclipse en:

AppWeb > Deployment Descriptor > Servlets > TestServlet

- f) Compruebe que /servlets/ServF está asociado al nombre interno ServletEj1 (información sacada de WEB-INF/web.xml) accediendo en Eclipse a:

AppWeb > Deployment Descriptor > Servlet Mappings

- g) Acceda a la clase correspondiente haciendo doble click en Eclipse en:

AppWeb > Deployment Descriptor > Servlets > ServletEj1

- h) ¿Se ha abierto en el editor el siguiente fichero?

AppWeb/src/fast/ServletFecha.java

Ya que se ha visto cómo funciona un servlet con anotaciones, se va a crear uno con la ayuda de Eclipse. Se va a crear un servlet en el paquete fast.paqprueba, cuyo nombre de clase será ServletPrueba03, el nombre interno será ServletEj3, con dos parámetros de inicialización y asociado a las URLs /servlets/ServP03 y *.prueba.

TAREAS

- a) Para crear un nuevo servlet pulse el botón derecho en el explorador de proyecto:

AppWeb > Botón-derecho > New > Servlet

- b) Rellene los campos paquete Java y nombre Clase. Pulse siguiente.
 c) Rellene los campos para nombre interno, descripción, dos parámetros de inicialización y las URLs (ver capturas a continuación). Pulse fin.
 d) Se ha debido crear el fichero

AppWeb/src/fast/paqPrueba/ServletPrueba03.java

- e) Compruebe que / servlets/ServP03 y *.prueba están asociados al nombre interno ServletEj3 accediendo en Eclipse a:

AppWeb > Deployment Descriptor > Servlet Mappings

- f) Acceda a la clase ServletPrueba03 haciendo doble click en Eclipse en:

AppWeb > Deployment Descriptor > Servlets > ServletEj3

- g) ¿Se ha abierto en el editor el siguiente fichero?

AppWeb/src/fast/paqPrueba/ServletPrueba03.java

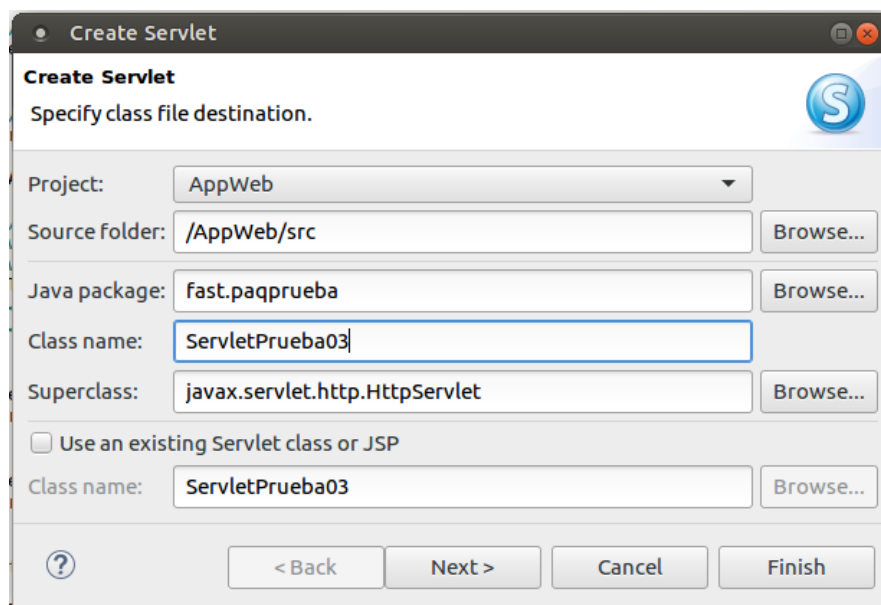
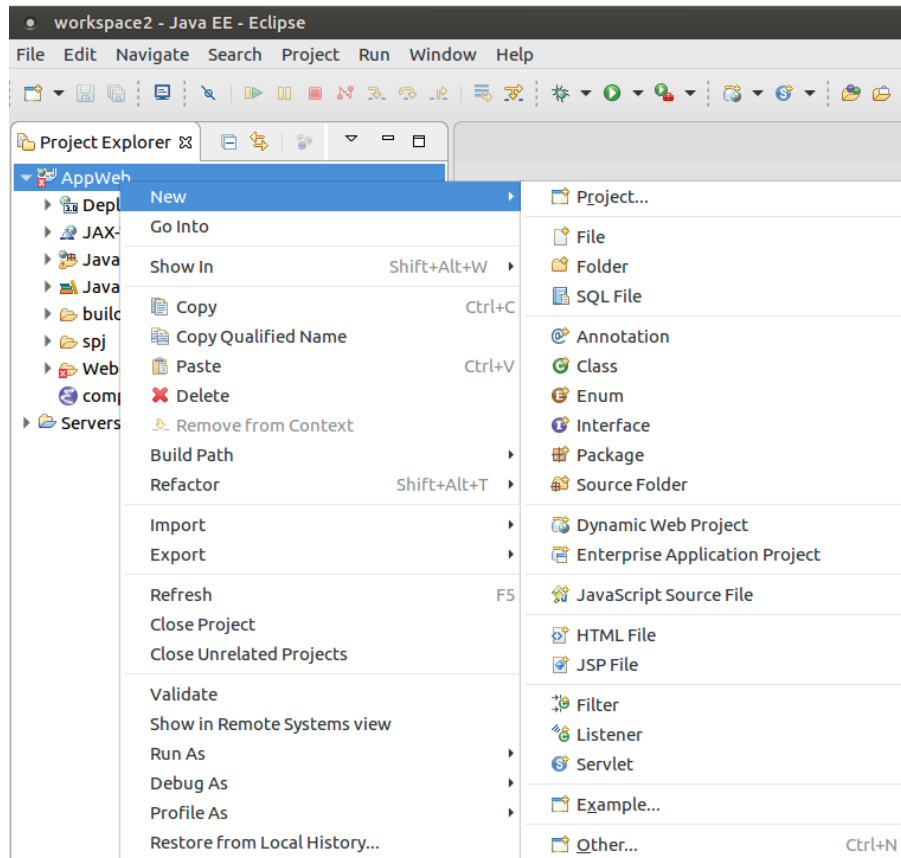
- h) Modifique en el código java el método doGet para que envíe una página html (sea similar al de TestServlet.java, declarando una variable del tipo PrintWriter) pero que muestre el resultado de request.getContextPath(). Compruebe que Eclipse ayuda a terminar de completar los nombres de las variables, los métodos, etc. Si no aparece puede pulsar Ctl+espacio. Puede que tenga que reiniciar tomcat la primera vez después de grabar los cambios.
 i) Compruebe la existencia del fichero java en los directorios de Eclipse y del fichero class en los directorios de tomcat tal como se hizo en el apartado de compilación del servlet
 j) Para acceder con http ¿qué URL debe usar?

http://localhost:8080/AppWeb/ServletEj3
 http://localhost:8080/AppWeb/ServletPrueba03.java
 http://localhost:8080/AppWeb/fast/paqprueba/ServletPrueba03.java
 http://localhost:8080/AppWeb/fast/paqprueba/ServletPrueba03.class
 http://localhost:8080/AppWeb/servlets/ServP03
 http://localhost:8080/AppWeb/algo/distinto.prueba

- k) Modifique el fichero java y añada párrafos con el resultado de distintos métodos de request que devuelvan un String. Ayúdese de Eclipse y de la documentación vista en teoría sobre el tipo de request (es del tipo HttpServletRequest):

<http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html>

- l) Acceda a través de http para comprobar el resultado



Create Servlet
Enter servlet deployment descriptor specific information.

Name:

Description:

Initialization parameters:

Name	Value	Description
data1	valor1	primer parámetro
data2	valor2	segundo parámetro

URL mappings:

/servlets/ServP03
*.prueba

☐ Asynchronous Support

< Back Next > Cancel Finish

AppWeb/src/fast/paqpueba/ServletPrueba03.java

```
package fast.paqpueba;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class ServletPrueba03
 */
@WebServlet(
    name = "ServletEj3",
    description = "Servlet de prueba",
    urlPatterns = {"/servlets/ServP03", "*.prueba"},
    initParams = {
        @WebInitParam(name = "data1", value = "valor1", description =
"primer parámetro"),
        @WebInitParam(name = "data2", value = "valor2", description =
"segundo parámetro")
    })
public class ServletPrueba03 extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
```

```
    * @see HttpServlet#HttpServlet()
    */
    public ServletPrueba03() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request,
     HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException
    {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at:
        ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request,
     HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException
    {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}
```

4.8 Atributos y ámbitos

Los atributos permiten almacenar información para poder recuperarla posteriormente. Aplicado a los Servlets se usan 3 ámbitos o contextos:

- Petición
- Sesión
- Aplicación

Para almacenar información se usa el método `setAttribute` y para recuperarla `getAttribute`. Dependiendo del ámbito, si `request` es el objeto que representa la petición (es el parámetro de `doGet` o de `doPost`), del tipo `HttpServletRequest`, las llamadas a los métodos tendrían los siguientes prototipos:

```
void request.setAttribute(String name, Object value)
Object request.getAttribute(String name)
void request.getSession().setAttribute(String name, Object value)
Object request.getSession().getAttribute(String name)
void request.getServletContext().setAttribute(String name, Object value)
Object request.getServletContext().getAttribute(String name)
```

Consulte estos métodos en la documentación de la interfaz `HttpServletRequest` del paquete `javax.servlet.http`

<http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html>

o en las interfaces de las que hereda.

Nota: Tenga en cuenta que el objeto devuelto por `getServletContext` (que implementa la interfaz `ServletContext`), a pesar de su nombre, hace referencia a la aplicación. Algunos autores sostienen que debería haberse llamado `AppContext` (pero desgraciadamente no fue así). También se puede llamar a `getServletContext()` directamente desde el servlet, ya que es un método que pertenece a la clase `GenericServlet`:

<http://docs.oracle.com/javaee/7/api/javax/servlet/GenericServlet.html>

La razón de usar `request.getServletContext()` es por buscar una relativa uniformidad.

Los ámbitos son estudiados en mayor profundidad en el apartado de JSP. Una vez estudiado JSP vuelva a este apartado y programe servlets con una funcionalidad similar a los JSPs.

4.9 Parámetros de la petición y parámetros de inicialización

No hay que confundir los parámetros que vienen en la petición http (en un mensaje GET o POST) y los de inicialización que se declaran asociados a la configuración de servlet o de la aplicación (estos dos últimos mediante anotaciones o en `web.xml`).

4.9.1 Parámetros de la petición

Para los parámetros de la petición http se utilizan los métodos de la interfaz `HttpServletRequest` (o los de las que hereda) `getParameter`, `getParameterNames`, `getParameterValues` o `getParameterMap`. Si `request` es la petición:

```
String request.getParameter(String name)
```

Consulte los otros métodos en la documentación de la interfaz `HttpServletRequest` (ya vista).

4.9.2 Parámetros de inicialización del servlet

Para los parámetros de inicialización del servlet se utiliza el método `getInitParameter` (y también `getInitParameterNames`) de la clase `GenericServlet` (de la cual deriva `HttpServlet`). Para poder usarlo se accede directamente desde el servlet:

```
String getInitParameter(String name)
```

Consulte los métodos en la documentación:

<http://docs.oracle.com/javaee/7/api/javax/servlet/GenericServlet.html>

Realmente los parámetros están en el objeto de tipo `ServletConfig`, devuelto por `getServletConfig()` y que se corresponde con el parámetro del método `init` del `Servlet`. Consulte los métodos en la documentación de la interfaz

<https://docs.oracle.com/javaee/7/api/javax/servlet/ServletConfig.html>

Nota: Los parámetros de inicialización del servlet se encuentran en la anotación del servlet o en el descriptor de despliegue (`WEB-INF/web.xml`) dentro de la etiqueta `<init-param>`, que a su vez está dentro de la etiqueta `<servlet>` del servlet correspondiente.

4.9.3 Parámetros de inicialización de la aplicación

Para los parámetros de inicialización de la aplicación se utiliza el método `getInitParameter` de la interfaz `ServletContext` (y también `getInitParameterNames`). Para poder usarlo se accede a través del objeto devuelto por `getServletContext` (que implementa la interfaz `ServletContext`), que hace referencia a la aplicación.

```
String getServletContext().getInitParameter(String name)
```

Consulte los métodos en la documentación:

<http://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html>

Nota: Los parámetros de inicialización de la aplicación se encuentran en el descriptor de despliegue (`WEB-INF/web.xml`) dentro de la etiqueta `<context-param>`. Puede consultar información sobre `web.xml` en:

<http://localhost:8080/docs/appdev/deployment.html>

TAREAS

- Cree una página html que contenga un formulario (puede reutilizar alguna existente) y haga que los datos sean enviados con GET al servlet ServletPrueba03.java
- Modifique el servlet para que muestre los parámetros de la petición, de inicialización del servlet y de inicialización de la aplicación.
- Modifique el formulario para que los datos sean enviados con POST, ¿hay que modificar el servlet para obtener los parámetros de la petición?

4.10 Clase auxiliar: ServletContextListener

A través de los eventos en el servidor, se puede estar informado del inicio y finalización (ciclo de vida) de la aplicación, y ejecutar acciones asociadas (en general ServletContext significa aplicación). Para ello se debe usar una clase que implemente la interfaz ServletContextListener. Las interfaces basadas en “Listener” permiten detectar cambios. Existen otras interfaces “Listener” para detectar otros eventos, como cambios en los atributos de aplicación (ServletContextAttributeListener), cambios en la sesión (HttpSessionListener), etc.

Para ejecutar acciones asociadas al inicio y al fin de la aplicación, la interfaz contiene:

```
void contextInitialized()
void contextDestroyed()
```

Consulte los métodos en la documentación:

<https://docs.oracle.com/javaee/7/api/javax/servlet/ServletContextListener.html>

Se usa la anotación WebListener. Consulte la documentación:

<http://docs.oracle.com/javaee/7/api/javax/servlet/annotation/WebListener.html>

Observe el siguiente “Listener”

AppWeb/src/fast/ anotaciones/TestServletContextListener.java

```
package fast.anotaciones;

import java.util.Date;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class TestServletContextListener implements ServletContextListener {

    @Override
    public void contextDestroyed(ServletContextEvent contextEvent) {
        System.out.println("TestServletContextListener: Detectado el cierre de la aplicación web de FAST.");
        System.out.println(
            "TestServletContextListener: Si hay que guardar o cerrar algún recurso, este es el momento.");
    }

    @Override
    public void contextInitialized(ServletContextEvent contextEvent) {
        System.out.println("TestServletContextListener: Inicializando la aplicación web de FAST.");
        System.out.println("TestServletContextListener: " +
            "Los ficheros de la aplicación están en el directorio " +
            contextEvent.getServletContext().getRealPath("/"));
        contextEvent.getServletContext().setAttribute("fechaInicio", new Date());
    }
}
```

```
System.out.println("TestServletContextListener: "+
    "Creo la variable fechaInicio con valor "+
    contextEvent.getServletContext().getAttribute("fechaInicio")
    );
}
}
```

TAREAS

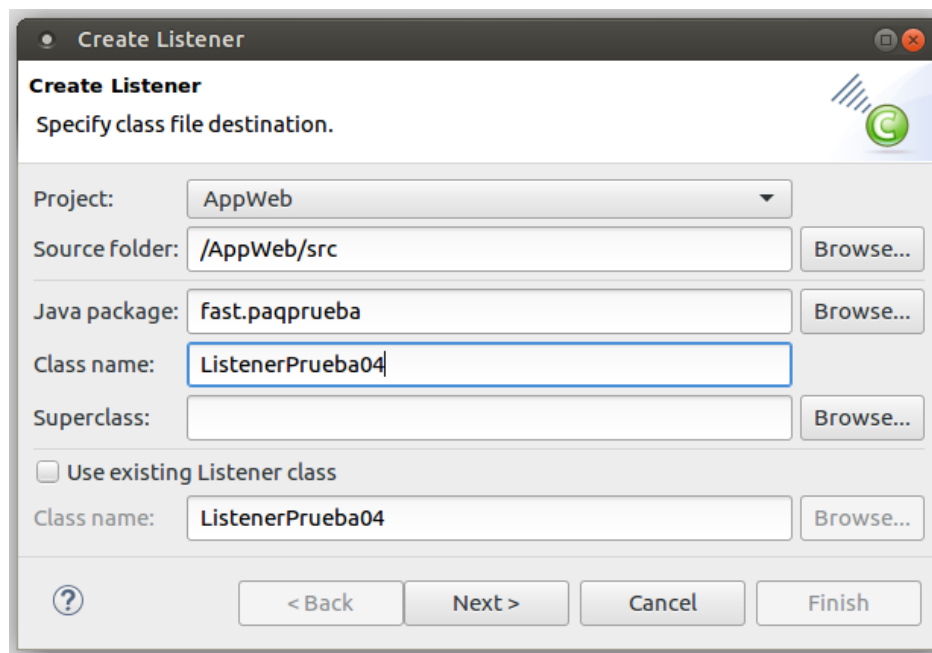
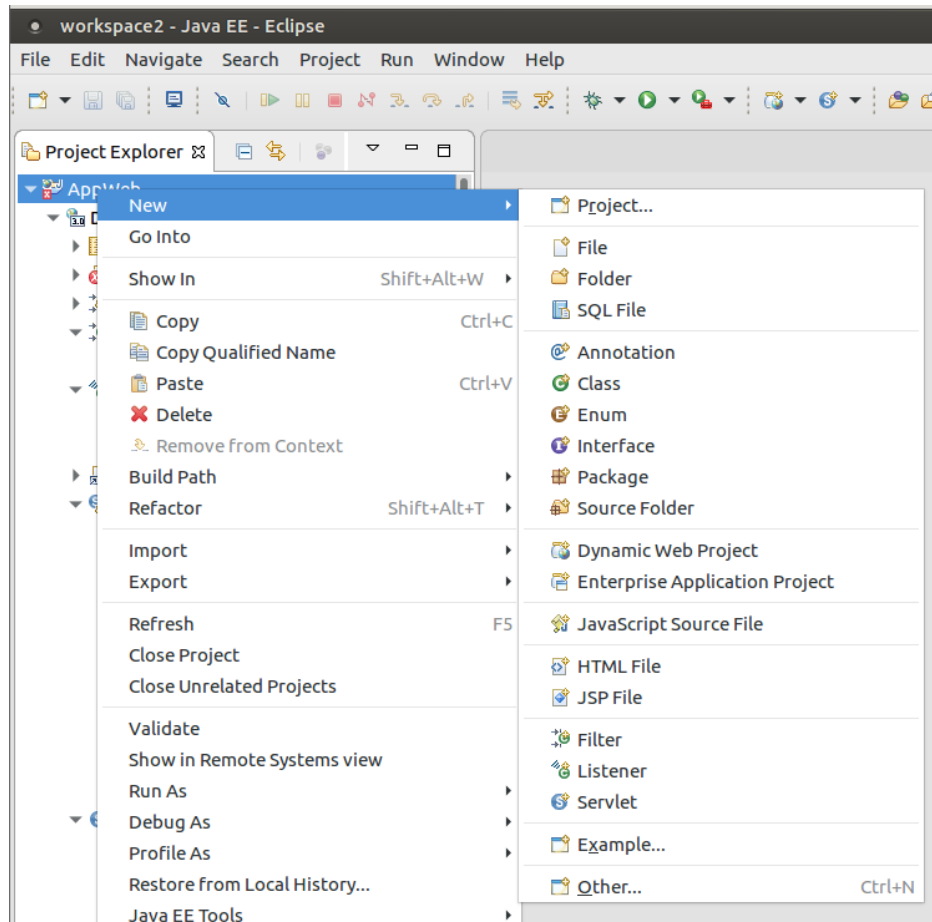
- a) Estudie el código y observe cómo se accede al contexto de la aplicación.
- b) Acceda al fichero desde Eclipse y añada una línea para que imprima por consola su nombre cuando se inicie la aplicación.
- c) Observe que cuando se recompila el fichero, se reinicia la aplicación (observe la salida de consola).
- d) Pruebe a recargar la aplicación /AppWeb desde el manager de tomcat, y observe la salida de consola de Eclipse

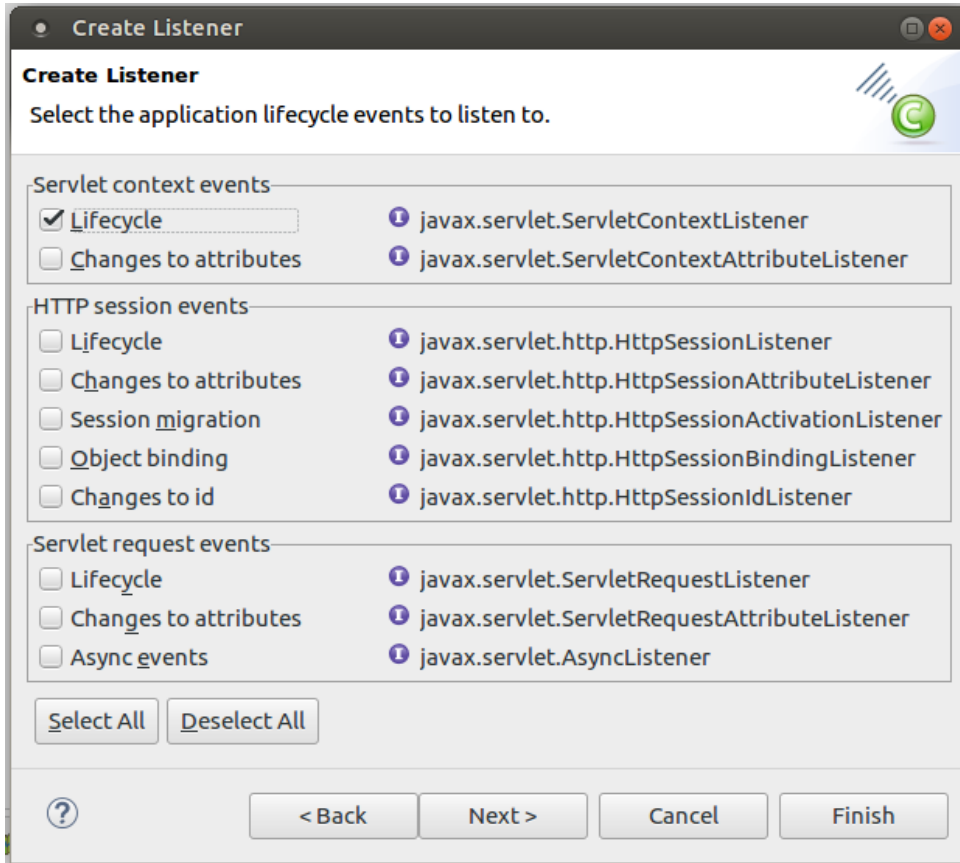
<http://localhost:8080/manager/>

- e) La creación de un “Listener” con anotaciones es similar a la creación de un servlet con anotaciones, ya visto anteriormente.
- f) Para crear un nuevo “Listener” pulse botón derecho en el explorador de proyecto (ver capturas a continuación):

AppWeb > Botón-derecho > New > Listener

- g) Cree un nuevo “Listener” (paquete fast.paqprueba, cuyo nombre de clase será ListenerPrueba04) para que también realice acciones con la inicialización y finalización de la aplicación. En concreto haga que cuando se cierre la aplicación imprima por consola el valor del atributo fechaInicio del contexto de aplicación (el que se crea en el Listener de ejemplo).
- h) Compruebe que cuando se cierra la aplicación se imprime por consola la fecha de la inicialización.





Create Listener

Select the application lifecycle events to listen to.

Servlet context events

<input checked="" type="checkbox"/> Lifecycle	javax.servlet.ServletContextListener
<input type="checkbox"/> Changes to attributes	javax.servlet.ServletContextAttributeListener

HTTP session events

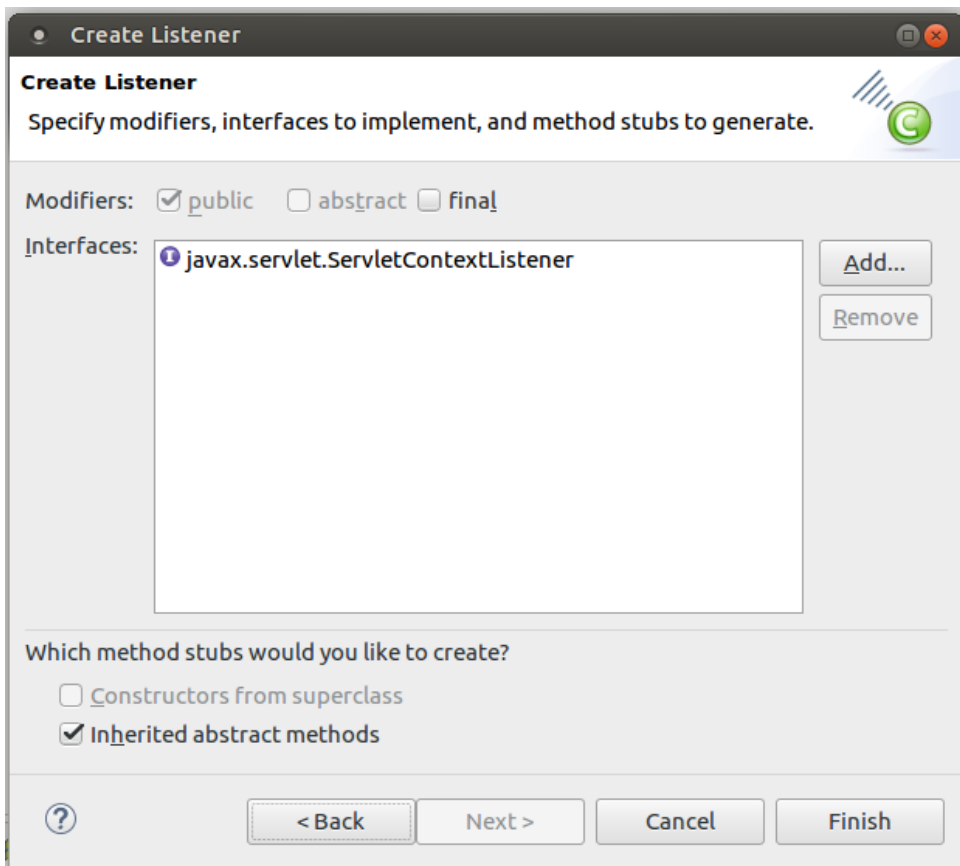
<input type="checkbox"/> Lifecycle	javax.servlet.http.HttpSessionListener
<input type="checkbox"/> Changes to attributes	javax.servlet.http.HttpSessionAttributeListener
<input type="checkbox"/> Session migration	javax.servlet.http.HttpSessionActivationListener
<input type="checkbox"/> Object binding	javax.servlet.http.HttpSessionBindingListener
<input type="checkbox"/> Changes to id	javax.servlet.http.HttpSessionIdListener

Servlet request events

<input type="checkbox"/> Lifecycle	javax.servlet.ServletRequestListener
<input type="checkbox"/> Changes to attributes	javax.servlet.ServletRequestAttributeListener
<input type="checkbox"/> Async events	javax.servlet.AsyncListener

Select All Deselect All

? < Back Next > Cancel Finish



Create Listener

Specify modifiers, interfaces to implement, and method stubs to generate.

Modifiers: ☒ public ☐ abstract ☐ final

Interfaces: javax.servlet.ServletContextListener

Add... Remove

Which method stubs would you like to create?

☐ Constructors from superclass

☒ Inherited abstract methods

? < Back Next > Cancel Finish

Nota: si no se usaran anotaciones, habría que añadir una etiqueta <listener> al fichero WEB_INF/web.xml.

4.11 Clase auxiliar: Filter

Los filtros permiten interceptar peticiones para realizar acciones antes de que éstas lleguen a los servlets, y también pueden actuar cuando los servlets han terminado. Todo ello sin que el servlet lo detecte.

Para ello se debe usar una clase que implemente la interfaz Filter, que tiene los siguientes métodos:

```
void destroy()
void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
void init(FilterConfig filterConfig)
```

Los filtros tienen un ciclo de vida similar a los servlets. Consulte los métodos en la documentación:

<https://docs.oracle.com/javaee/7/api/javax/servlet/Filter.html>

Se usa la anotación WebFilter. Consulte la documentación:

<http://docs.oracle.com/javaee/7/api/javax/servlet/annotation/WebFilter.html>

Observe el siguiente filtro:

AppWeb/src/fast/ anotaciones/TestFilter.java

```
package fast.anotaciones;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.http.HttpServletRequest;

/**
 * @author dit
 *
 */
@WebFilter(urlPatterns = { "*.app", "/filtro" })
public class TestFilter implements Filter {

    /**
     * (non-Javadoc)
     *
     * @see javax.servlet.Filter#destroy()
     */
    @Override
    public void destroy() {
        // Si hay que eliminar o guardar algo antes de terminar.
    }

    /**
     * (non-Javadoc)
     *
     * @see javax.servlet.Filter#doFilter(javax.servlet.ServletRequest,
     * javax.servlet.ServletResponse, javax.servlet.FilterChain)
     */
    @Override
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
        throws IOException, ServletException {
        /**
         * Ejemplos de filtrado
         * - Seguridad, bloquear el acceso a determinadas páginas.
         * - Registro y auditoría de páginas consultadas.
         */
    }
}
```

```

* - Conversión de imágenes (escalado automático).
* - Compresión de datos.
* - Servir distintas páginas según el idioma.
* - Modificar las peticiones y respuestas.
* - etc
*/
HttpServletRequest request = (HttpServletRequest) req;
String url = request.getServletPath();
System.out.println("TestFilter: filtrando");

if (url.endsWith("/filtro")) {
    PrintWriter salida = resp.getWriter();
    salida.println("<!DOCTYPE html>");
    salida.println("<html><body><div>" +
        "<h1>URL Filtrada</h1><br/>" +
        "</div>" +
        "</body></html>");
} else if (url.endsWith(".app")) {
    // Cambia la extensión de la petición
    String nuevaUrl = url.substring(0, url.length() - 4) + ".jsp";
    System.out.println("TestFilter: reenvio a "+nuevaUrl);

    req.getRequestDispatcher(nuevaUrl).forward(req, resp);
} else {

    // Continúa la cadena de filtros
    chain.doFilter(req, resp);

    // Si se quiere modificar la respuesta que se va a enviar, se haría
    // a continuación.
    // ...
}

}

/*
 * (non-Javadoc)
 *
 * @see javax.servlet.Filter#init(javax.servlet.FilterConfig)
 */
@Override
public void init(FilterConfig arg0) throws ServletException {
    // Inicialización
    System.out.println("TestFilter: Inicialización.");
}
}

```

TAREAS

- a) Estudie el código.
- b) ¿Qué peticiones captura el filtro? Desde el navegador intente acceder para que el filtro actúe. Halle a qué url debería acceder desde el navegador para que:
 - aparezca URL filtrada
 - se acceda a jsp/ejemplo1.jsp (sin poner esto en el navegador)
 Nota: el método forward se ve en el apartado siguiente.
- c) Modifique el filtro para que capture también las peticiones terminadas en .jsp. En ese caso no debe hacer nada antes de acceder al recurso, pero sí debe imprimir por consola un mensaje indicando el recurso o URL a la que ha accedido (después de haberlo hecho).
- d) Cree un filtro (de manera similar a un servlet) para que capture las peticiones terminadas en .php y las reenvíe a terminadas en .jsp, imprimiendo también mensajes por consola.

Nota: con las anotaciones no es posible determinar el orden en que se ejecutarían varios filtros asociados al mismo recurso, aunque eso no es un problema, ya que en un buen diseño los filtros deberían ser independientes unos de otros. Si no se usaran anotaciones, habría que añadir una etiqueta <filter> al fichero WEB_INF/web.xml.

4.12 Inclusión y reenvío: RequestDispatcher

De forma similar a cuando se vea en JSP las etiquetas `jsp:include` y `jsp:forward`, en los servlets existe la misma funcionalidad de poder acceder a otro recurso. Con `include` es como una llamada a una subrutina, y con `forward` es un reenvío sin vuelta.

Para poder realizarlo se utiliza un objeto con la interfaz `RequestDispatcher`, que tiene los métodos:

```
void include(ServletRequest request, ServletResponse response)
void forward(ServletRequest request, ServletResponse response)
```

Consulte los métodos en la documentación:

<https://docs.oracle.com/javaee/7/api/javax/servlet/RequestDispatcher.html>

El objeto `RequestDispatcher` se obtiene con el método `getRequestDispatcher`, que se puede usar en el contexto de la petición o de la aplicación:

```
RequestDispatcher request.getRequestDispatcher(String path)
```

```
RequestDispatcher getServletContext().getRequestDispatcher(String path)
```

Consulte la documentación y averigüe cómo se especifica el recurso.

<https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html>

<https://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html>

TAREAS

- En el ejemplo del filtro visto anteriormente se usa el método `forward`. Localícelo y observe su uso.
- El método `getRequestDispatcher` del ejemplo anterior, ¿se usa en el contexto de la petición o de la aplicación?
- Cámbielo al otro contexto y pruébelo.

5 JSP

5.1 Introducción de JSP

5.1.1 Motivación de JSP

A través de la sentencia “`salida.println`” del Servlet hemos conseguido generar la página web dinámica. Cuando el código HTML a producir es reducido, como el ejemplo presentado, no hay problema en generarlo mediante instrucciones `println` dentro de un Servlet. Sin embargo, por norma general, es muy complicado mantener el código y generar páginas HTML junto con ECMAScript u otro lenguaje dinámico en el cliente, y más cuando el contenido es extenso.

El desarrollo de Servlets es tedioso, ya que hay que programar en Java pero enviando a la salida código HTML. En el siguiente apartado comprobará la utilidad de JSP, que permite incrustar código Java dentro una página HTML en lugar de generar el HTML desde una clase Java como ocurre con los Servlets.

Los desarrolladores de aplicaciones conocen Java, y los diseñadores conocen HTML. Con JSP, los desarrolladores Java pueden seguir generando el código Java dentro de código HTML, y los diseñadores que conocen HTML pueden hacer páginas web apoyándose en la potencia del lenguaje Java.

5.1.2 Aspectos generales de JSP

JavaServer Pages (JSP) es una tecnología basada en el lenguaje Java que permite incorporar contenido dinámico a las páginas web HTML. JSP está especificado (versión 2.1) en la JSR245, que puede consultar en:

<https://jcp.org/aboutJava/communityprocess/final/jsr245/index.html>

Una página JSP es como una página HTML en aspecto, excepto que es posible incrustar código Java dentro de la página. Los archivos JSP combinan HTML con etiquetas especiales (acciones) y fragmentos de código Java (scriptlets). El código HTML se añade tal cual a la respuesta, mientras que los elementos dinámicos se evalúan con cada petición.

Un ejemplo de página JSP:

AppWeb/WebContent/jsp/fecha.jsp

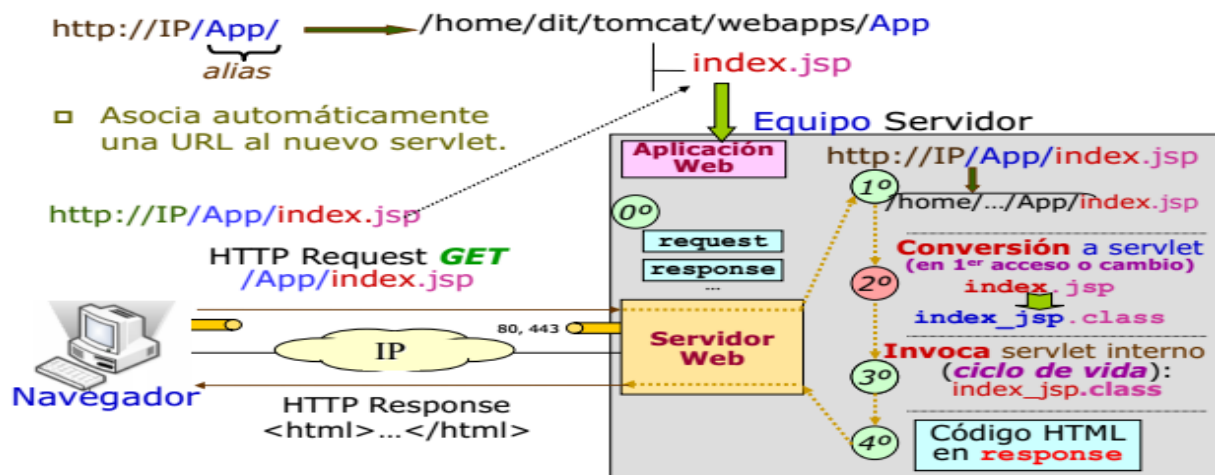
```
<%@ page contentType='text/html; charset=UTF-8' %>
<%@ page import='java.util.Date' %>

<!DOCTYPE html>

<html>
  <head><title>Hola Mundo</title></head>
  <body>
    <h1>Hola, esto es una página JSP</h1>
    <p>La hora del servidor es <%= new Date() %></p>
  </body>
</html>
```

En el ejemplo anterior, se observan las etiquetas que comienzan con “<%”, correspondientes a JSP.

El funcionamiento de aplicaciones web basadas en tecnología JSP es el siguiente: cuando se produce una solicitud de una página JSP por parte de un navegador, se comienza comprobando si se trata de la primera solicitud de dicha página. En ese caso se traduce, generándose un servlet de forma automática, que es compilado e interpretado, y cuya salida es devuelta al usuario que realizó la petición. Las siguientes solicitudes son más eficientes al no necesitar que dicha página sea compilada de nuevo, ya que sólo se invoca al servlet que se generó en la petición inicial. En este sentido, esta tecnología es más ventajosa que otras como ASP, que requieren una nueva compilación cada vez:



El proceso de las páginas jsp consiste en analizar su contenido buscando etiquetas JSP y traduciendo éstas a código Java equivalente. El contenido estático de las páginas (código HTML) es traducido a cadenas de caracteres en lenguaje Java. Las etiquetas de componentes JSP son traducidas a su correspondiente objeto Java. Todo este código fuente es usado para formar los métodos de servicio del servlet.

Al igual que sucede con los servlets, para hacer uso de páginas JSP se necesita un componente adicional instalado en el propio servidor web que sea capaz de manipular toda la lógica que dichas páginas implementan, esto es, se requiere un motor (engine) que se integre de algún modo con el servidor web a fin de que las peticiones hechas al servidor puedan ser recogidas por el código Java para generar la respuesta adecuada. Uno de estos motores es Tomcat.

5.1.3 Ciclo de vida de una página JSP

Si observamos el código de una página JSP, parece código HTML, no tiene apariencia de una clase Java. El contenedor de JSPs traduce las páginas JSP y crea la clase del servlet que se utiliza en la aplicación web. Las fases del ciclo de vida una página JSP son:

- 1) **Traducción:** el contenedor de JSPs revisa el código en la página JSP y lo procesa para generar el código fuente del servlet. Por ejemplo, en Tomcat, los ficheros de las clases de los servlets generados se encuentran en
`$TOMCAT/work/Catalina/localhost/WEBAPP/org/apache/jsp`

En el caso de la máquina virtual de la asignatura, para las aplicaciones jsp de la práctica, este directorio sería:

`/home/dit/tomcat/work/Catalina/localhost/AppWeb/org/apache/jsp`

Si el nombre de la página es:	<code>pagina.jsp</code>
entonces el nombre de la clase de servlet generado	
será, normalmente:	<code>pagina_jsp</code>
y el fichero será:	<code>pagina_jsp.java</code>

- 2) **Compilación:** el contenedor JSP compila la clase y produce el fichero de clase.
- 3) **Carga de clase:** el contenedor carga la clase en memoria.
- 4) **Instanciación:** el contenedor invoca al constructor sin argumentos de la clase generada para cargarlo en memoria e instanciarlo.
- 5) **Inicialización:** el contenedor invoca el método `init` del objeto instanciado e inicializa la configuración del servlet con los parámetros del descriptor de despliegue (`web.xml`). Después de esta fase ya está listo para manejar peticiones de los clientes. Normalmente, los pasos de traducción hasta inicialización ocurren cuando llega la primera petición, pero puede configurarse que sea cargado e inicializado en el momento del despliegue, como los servlets, empleando un elemento “load-on-startup”.
- 6) **Procesamiento de la petición:** el procesamiento es multi-hilo (como lo servlets), para cada petición se genera un hilo, donde se crean los objetos `ServletRequest` y `ServletResponse` y se invoca al método `service`.
- 7) **Destrucción:** el contenedor invoca el método `destroy` y es eliminado de la memoria. Normalmente ocurre cuando la aplicación se repliega en el servidor o se apaga el servidor.

TAREAS

- a) Acceda desde el navegador a la siguiente página JSP:

<http://localhost:8080/AppWeb/jsp/fecha.jsp>

- b) Compruebe que el código del Servlet generado está en el fichero fecha.jsp.java y el bytecode está en el fichero fecha.jsp.class, ambos en el directorio:

`/home/dit/tomcat/work/Catalina/localhost/AppWeb/org/apache/jsp/jsp`

- c) Acceda fichero .jsp (siempre que acceda a los fuentes, debe hacerse desde Eclipse) y añada su nombre al texto de la etiqueta h1. El fichero está en el directorio asociado al alias /AppWeb:

`AppWeb/WebContent/jsp/fecha.jsp`

- d) Grabe la modificación y vuelva a acceder al Servlet desde el navegador ¿aparece su nombre?

- e) Compruebe que el fichero .java y .class tienen fecha de modificación posterior al .jsp.

- f) Cree una copia del fichero fecha.jsp en:

`AppWeb/WebContent/jsp/fecha2.jsp`

- g) Acceda desde el navegador a:

<http://localhost:8080/AppWeb/jsp/fecha2.jsp>

- h) Compruebe el código del Servlet generado en el directorio correspondiente y muéstrelo por pantalla. Compruebe la clase de la que deriva (`extends org.apache.jasper.runtime.HttpJspBase`) Consulte esta clase en:

<https://tomcat.apache.org/tomcat-7.0-doc/api/org/apache/jasper/runtime/HttpJspBase.html>

Compruebe que la definición de esta clase abstracta:

- tiene los métodos `jspInit()` y `jspDestroy()` (se explicarán en el siguiente apartado)
- implementa la interfaz `HttpJspPage` (que deriva de `JspPage`, que a su vez deriva de `Servlet`)
- deriva de `HttpServlet`, que ya consultó cuando practicó con los Servlets, en:

<https://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServlet.html>

- i) Compruebe las diferencias entre el Servlet original y la copia con `diff` (en el directorio correspondiente):

`diff fecha.jsp.java fecha2.jsp.java`

Compruebe que difieren en el nombre de la clase y en el instante de generación

- j) Modifique esta copia y añada un párrafo al final de body con su UVUS. Grabe la modificación y vuelva a acceder desde el navegador. Vuelva a comprobar las diferencias con `diff`. Ahora además deben diferir en la línea que muestra su UVUS.

5.1.4 Métodos sobrescribibles en JSP

Los métodos de una página JSP que se pueden sobrescribir son:

- 1) **jspInit()**: este método se declara en la interfaz `JspPage` (que deriva de la interfaz `Servlet`) y es implementado por los distintos contenedores JSP. Este método se llama una vez al principio del ciclo de vida del JSP para inicializarlo con los parámetros de configuración del descriptor de despliegue. Se puede sobrescribir utilizando declaraciones de JSP (las veremos más adelante) para inicializar cualquier recurso que queramos emplear en la página JSP.
- 2) **jspDestroy()**: este método se declara en la interfaz `JspPage` y es llamado por el contenedor cuando el JSP es eliminado de la memoria, por ejemplo, cuando se deshabilita la aplicación. Este método sólo es llamado una vez en el ciclo de vida del JSP y debería ser sobrescrito cuando haya que liberar recursos creados previamente en el método `jspInit`.

Nota: existe también el método `_jspService()` que NO se debe intentar sobrescribir mediante declaraciones JSP (por ello su nombre empieza por subguión, y si se sobrescribe daría error). Este es el método JSP que es invocado por el contenedor para cada petición de cliente, pasando los objetos `request` y `response`. Todo el código JSP que va dentro de este método es generado por Tomcat. Este método se declara en la interfaz `HttpJspPage` (que deriva de la interfaz `JspPage`).

TAREAS

- a) Acceda al código del Servlet generado en la tarea anterior en el directorio correspondiente y muéstrelo por pantalla. Compruebe que están definidos los métodos `_jspInit()`, `_jspDestroy()` y `_jspService()`. Estos métodos NO deben ser sobrescritos.
- b) Compruebe que el método `_jspService()` contiene el código necesario para enviar a la salida el código HTML.

- c) Compruebe que la línea del fichero JSP

```
<p>La hora del servidor es <%= new Date() %></p>
```

Se ha convertido en el fichero java en

```
out.write(" <p>La hora del servidor es ");
out.print( new Date() );
out.write("</p>\n");
```

- d) Compruebe que NO están definidos los métodos `jspInit()`, `jspDestroy()`. Estos métodos pueden ser sobrescritos mediante declaraciones JSP (se verá más adelante).

5.1.5 Elementos de las páginas JSP

El código de una página JSP puede contener:

- **Directivas:** proporcionan al motor JSP información sobre lo que éste debe hacer a la hora de procesar la página. Estas acciones pueden ser estándar (ya definidas por la especificación JSP) o etiquetas personalizadas. Las directivas son instrucciones procesadas por el compilador JSP cuando la página se compila a un Servlet. En JSP se definen tres tipos diferentes de directivas:
 - *page* `<%@ page`
 - *include* `<%@ include`
 - *taglib* `<%@ taglib`
- **Elementos de script:**
 - *comentarios* `<%--`
 - *declaraciones* `<%!`
 - *expresiones* `<%=`
 - *scriptlets* `<%`
- **Etiquetas de Acción JSP (Action Tags):** proporcionan información global de la página e indican lo que el motor JSP debe hacer con la página JSP a la hora de la ejecución de la página. Las acciones son el elemento que se encarga de controlar el funcionamiento del motor JSP y siguen el formato clave/valor: `<accion parametro=valor ... />`. Aquí se incluyen los *JavaBeans*. Las principales son:
 - *Obtener JavaBean* `<jsp:useBean`
 - *Obtener propiedad* `<jsp:getProperty`
 - *Fijar propiedad* `<jsp:setProperty`
 - *Incluir recurso* `<jsp:include`
 - *Reenviar petición* `<jsp:forward`

A continuación, presentamos una tabla resumen con ejemplos de las etiquetas JSP principales:

Elemento		Etiquetas JSP	Funcionalidad
Directivas JSP		<code><%@ page import="libr" %></code> <code><%@ include file="URL_file"%></code> <code><%@ taglib</code>	Importar librerías Java Insertar fichero (sin interpretar). ...
Comen- tarios	HTML	<code><!-- Comentario</code> <code>[<%= Expresión %>] --></code>	Comentario HTML (llega al navegador). Posibles expresiones JSP
	JSP	<code><%-- Comentario --%></code>	Desaparece tras la interpretación JSP
Declaraciones		<code><%! Declaraciones_Java; %></code>	Declarar variables/funciones. Sólo se interpreta en el primer acceso (tras el arranque del servidor) a la página.
Expresiones		<code><%= Expresión_Java %></code>	Valor insertado dentro del código HTML (misma posición relativa al resto HTML)

Scriptlets	<code><% Código_Java_(posibles declaraciones_variables); %></code>	Código Java (puede declarar variables, pero no funciones). Se interpreta en todos los accesos .
Etiquetas de Acción JSP	<code><jsp:include page="URL_file"/></code>	Insertar resultado de interpretar fichero

En adición a lo anterior, el motor JSP dispone de una serie de objetos implícitamente creados, los cuales no necesitan ser declarados para ser usados, sino que pueden usarse directamente. Estos objetos resultan muy útiles para acceder a determinadas propiedades. Los **9 objetos implícitos** soportados son:

- **request:** es el objeto `HttpServletRequest` asociado con la solicitud del cliente. Con esta variable se puede acceder a los parámetros recibidos del formulario del cliente, así como a las cabeceras HTTP.
- **response:** es el objeto `HttpServletResponse` asociado con la respuesta al cliente.
- **session:** es el objeto `HttpSession` (sesión) asociado con la solicitud del cliente.
- **out:** corresponde al objeto de la clase Java `PrintWriter` que posee todo objeto de tipo `response`, usándose para enviar la salida al cliente o a otra página.
- **page:** sinónimo de `this` en lenguaje Java, empleado para llamar a los métodos definidos por la clase del servlet traducido.
- **pageContext:** el contexto de la página, contiene métodos para obtener información de la página.
- **exception:** es el objeto excepción producido en otra página. (sólo para páginas de manejo de errores).
- **application:** es el objeto `ServletContext`, asociado con el contexto de la aplicación.
- **config:** es el objeto `ServletConfig` asociado con la página.

JSP tiene bastantes ventajas frente a otros lenguajes como ASP o PHP. La ventaja fundamental es que se tiene toda la potencia del lenguaje Java, con sus ventajas: reusabilidad, robustez, multiplataforma, etc.

5.2 Comprobación del código

En el caso de JSP (y de los Servlets), la aplicación web se ejecuta "dentro del servidor", luego es un fragmento de código que se "integra" (invoca) dentro del programa del servidor web. Puede comprobarse el código de (además del "Análisis de sintaxis" mediante el entorno de desarrollo Java empleado):

- 1) **Clases Java** (ficheros ".class"): pueden depurarse independientemente, utilizando `jdb`.
- 2) **Aplicación Web completa:** para este caso pueden emplearse dos opciones:
 - a. **Depuración** mediante excepciones de la propia Aplicación web: en el fichero "web.xml" de la aplicación, se ha de configurar lo siguiente:

```
<!-- Las siguientes directivas indican que si se
encuentra un error 404 (recurso no encontrado) o se
lanza una excepción cualquiera (todas derivan de
java.lang.Throwable) se redirija la petición a la página
/jsp/exception/error.jsp -->

<error-page>
  <error-code>404</error-code>
  <location>/jsp/exception/error.jsp</location>
</error-page>

<error-page>
  <exception-type>java.lang.Throwable</exception-type>
  <location>/jsp/exception/error.jsp</location>
</error-page>
```

La página "error.jsp" se encargará de imprimir en la pantalla del navegador los errores encontrados.

- b. Comprobación mediante el **registro de actividad/errores** del servidor (contenedor de servlets): el servidor que interpreta el código Java es capaz de "capturar" los errores que se producen al interpretar la aplicación web. En la máquina virtual, el servidor Tomcat registra los errores producidos en los accesos de cada día en el fichero de texto /home/dit/tomcat/logs/localhost_access_log.fecha-dia-actual.txt ", pudiendo consultarlos por ejemplo ejecutando el siguiente comando:

```
cat /home/dit/tomcat/logs/localhost_access_log.2017-04-25.txt
```

- 3) Entornos de desarrollo web que incorporen su propio servidor web contenedor de servlets, como es el caso de Eclipse (que incluye, por ejemplo, Tomcat), permiten realizar la **depuración** (y análisis de sintaxis) de la Aplicación Web Java completa directamente desde el IDE.

5.3 Directivas

Las directivas JSP se emplean para dar instrucciones especiales al contenedor para la traducción de una página JSP al código del Servlet generado. Las directivas JSP se definen mediante las etiquetas `<%@` y `%>`.

Las tres directivas disponibles son:

- `page`
- `include`
- `taglib`

Cada directiva tiene un conjunto de atributos para especificar un tipo específico de instrucciones. La forma habitual de llamada es mediante la sentencia siguiente:

```
<%@ directiva atributo="valor" %>
```

5.3.1 Directiva page

Ejemplo de uso de directiva page:

```
<%@ page language='java' contentType='text/html'
      isErrorPage='false' errorPage='error.jsp' %>
```

La directiva page proporciona atributos que se aplican a toda la página JSP. Se pueden definir múltiples atributos dentro de una sola directiva page o se pueden emplear varias directivas page en una misma página JSP.

TAREAS

- a) Consulte la directiva page y sus atributos en el apartado JSP.1.10.1 de la especificación de JSP2.1 (JSR245):
<http://download.oracle.com/otndocs/jcp/jsp-2.1-fr-eval-spec-oth-JSpec/>
- b) Encuentre los 15 atributos posibles.

Los atributos más importantes son:

- a) **import**: éste es uno de los más empleados. Se utiliza para instruir al contenedor cómo importar las clases Java, interfaces, etc., mientras se genera el código del servlet. Es similar a las sentencias `import` de las clases e interfaces Java. Ejemplo:

```
<%@ page import="java.util.Date,java.util.List,java.io.*" %>
```

- b) **contentType**: este atributo se emplea para definir el tipo MIME de contenido y conjunto de caracteres de una respuesta. El valor por defecto es “text/html; charset=ISO-8859-1”. Ejemplo:

```
<%@ page contentType="text/html; charset=UTF-8" %>
```

- c) **isELIgnored**: se emplea para instruir al contenedor que ignore el uso de Expression Language (EL), el cual veremos más adelante. El valor por defecto es “false”. Ejemplo:

```
<%@ page isELIgnored="true" %>
```

- d) **isThreadSafe**: se puede utilizar para implementar la interfaz SingleThreadModel en el servlet generado. El valor por defecto es “true”, y si se ajusta a “false” el servlet generado implementará SingleThreadModel y, eventualmente, perderemos todos los beneficios de la tecnología multi-hilo, es por ello que no es aconsejable ajustar el valor a “false”. Ejemplo:

```
<%@ page isThreadSafe="false" %>
```

- e) **errorPage**: este atributo se emplea para indicar la URL que gestionará el error, si el JSP arroja una excepción que no es capturada. La petición es redirigida al manejador de error definido en este atributo, y si la URL es una página JSP entonces el objeto implícito exception de esta página referenciará a la excepción no capturada. Si se usa esta directiva, para esta página no se usa el valor definido en <error-page> en web.xml. Ejemplo:

```
<%@ page errorPage="errorHandler.jsp" %>
```

- f) **isErrorPage**: se utiliza para declarar que la página actual es una página JSP de error. Por defecto, es un Enum con valor “false”. Si no se pone a “true” no se puede usar el objeto implícito exception. Ejemplo:

```
<%@ page isErrorPage="true" %>
```

- g) **session**: por defecto, la página JSP crea una sesión, pero a veces no necesitamos disponer de una sesión en la página JSP. Se puede emplear este atributo para indicar al compilador que no cree la sesión por defecto. Ejemplo:

```
<%@ page session="false" %>
```

- h) **trimDirectiveWhitespaces**: este atributo se añadió en JSP 2.1, y se utiliza para eliminar los espacios en blanco extra de la salida de la página JSP. El valor por defecto es “false”. Esto ayuda a reducir el tamaño del código generado. Por ejemplo, no tendrán efectos sentencias del tipo out.write(“\n”) cuando el valor de este atributo es “true”. Ejemplo:

```
<%@ page trimDirectiveWhitespaces="true" %>
```

Ejemplo de uso de directivas:

AppWeb/WebContent/jsp/fecha.jsp

```
<%@ page contentType='text/html; charset=UTF-8' %>
<%@ page import='java.util.Date' %>

<!DOCTYPE html>
```

```
<html>
<head><title>Hola Mundo</title></head>
<body>
  <h1>Hola, esto es una página JSP</h1>
  <p>La hora del servidor es <%= new Date() %></p>
</body>
</html>
```

TAREAS

- a) Cree una copia del fichero fecha.jsp en

AppWeb/WebContent/jsp/fecha3.jsp

- b) Acceda mediante el navegador a:

<http://localhost:8080/AppWeb/jsp/fecha3.jsp> para ver su funcionamiento y que se genere el Servlet.

- c) Acceda al código del Servlet generado en la tarea anterior en el directorio correspondiente y busque dónde aparece UTF-8 y java.util.Date

- d) Edite fecha3.jsp (NO fecha3_jsp.jsp) y cambie UTF-8 por ISO-8859-1 en la directiva page import. Añada también la siguiente directiva:

```
<%@ page import='java.io.*' %>
```

- e) Acceda al código del Servlet generado en la tarea anterior en el directorio correspondiente y busque el resultado de las modificaciones.

5.3.2 Directiva include

La directiva `include` presenta las siguientes características:

- Añade el contenido del archivo especificado dentro de la página JSP. Puede ser un fichero HTML u otra JSP.
- Se resuelve **ANTES** de comenzar la fase de traducción a Servlet.
- El texto se incluye en el lugar de la directiva.

Es equivalente al `#include` del lenguaje C.

Ejemplo de uso:

```
<%@ include file="footer.html" %>
```

Es útil a la hora de crear plantillas, separando las páginas en cabecera, pie de página, menús laterales, etc., e importando estos ficheros en las páginas deseadas, reutilizando así código y disminuyendo el tiempo de desarrollo.

TAREAS

- a) Cree una copia del fichero fecha.jsp en
AppWeb/WebContent/jsp/fecha4.jsp
- b) Edite fecha4.jsp y añada la directiva include necesaria para incluir el fichero cabecerah2.html (compruebe que está en el mismo directorio y su contenido) entre el elemento h1 y el elemento p.
- c) Acceda mediante el navegador a:
<http://localhost:8080/AppWeb/jsp/fecha4.jsp>
- d) Use las herramientas de desarrollador para inspeccionar el código HTML (p. ej. con F12) y compruebe que se ha incluido un elemento h2.
- e) Acceda al código del Servlet generado en el directorio correspondiente y busque dónde aparece el contenido del fichero cabecera2.html.

5.3.3 Directiva taglib

La directiva **taglib** extiende el conjunto de etiquetas que interpreta el contenedor. Asocia un prefijo con una biblioteca de etiquetas, que es un conjunto de programas Java que contienen métodos asociados con las etiquetas de la página JSP.

Ejemplo de uso:

```
<%@ taglib uri="/WEB-INF/c.tld" prefix="c"%>
```

5.4 Elementos de script

Podemos insertar código Java dentro de JSP mediante tres métodos distintos: **declaraciones**, **expresiones** y **scriptlets**.

5.4.1 Declaraciones

Las declaraciones contienen declaraciones de variables o métodos, mediante `<%! declaración %>`. Estas variables o métodos serán accesibles desde cualquier lugar de la página JSP. Hay que tener en cuenta que el servidor transforma la página JSP en un *Servlet*, y éste es usado por múltiples peticiones, lo que provoca que las variables conserven su valor entre sucesivas ejecuciones.

Ejemplo:

AppWeb/WebContent/jsp/declarar1.jsp

```
<%! int numeroAccesos=0; %>

<%!
private String isEmptyCart() {
    return "No hay nada";
}
%>

<!DOCTYPE html>

<html>
<body>
```

```

<p>
<%= "La pagina ha sido accedida " + (++numeroAccesos) +
    " veces desde el arranque del servidor" %>
</p>
<p>
<%= "El carrito está vacío: " + isEmptyCart() %>
</p>
</body>
</html>

```

TAREAS

- Acceda mediante el navegador a: <http://localhost:8080/AppWeb/jsp/declara1.jsp>
- Acceda varias veces a la misma página para comprobar cómo se incrementa el contador.
- Acceda al código del Servlet generado en el directorio correspondiente y busque dónde aparece lo que está en las etiquetas de declaración (las etiquetas de expresión las verá en el siguiente apartado).

En las declaraciones se puede sobrescribir el método `jspInit()`, para que se ejecute antes de la primera petición. Observe el siguiente ejemplo:

AppWeb/WebContent/jsp/declara2.jsp

```

<%! int numeroAccesos=150; %>

<%!
    private String isEmptyCart() {
        return "No hay nada";
    }
%>

<%!
    public void jspInit() {
        numeroAccesos=0;
    }
%>

<!DOCTYPE html>

<html>
<body>
<p>
<%= "La pagina ha sido accedida " + (++numeroAccesos) +
    " veces desde el arranque del servidor" %>
</p>
<p>
<%= "El carrito está vacío: " + isEmptyCart() %>
</p>
</body>
</html>

```


TAREAS

- a) Acceda mediante el navegador a: <http://localhost:8080/AppWeb/jsp/declara2.jsp>
- b) Acceda varias veces a la misma página para comprobar cómo se incrementa el contador.
- c) Acceda al código del Servlet generado en el directorio correspondiente y busque dónde aparece lo que está en las etiquetas de declaración (las etiquetas de expresión las verá en el siguiente apartado).
- d) ¿Por qué el contador no se inicia al valor 150 tal como aparece en la primera declaración? Edite el fichero, cambie el valor que se le da en el método `jspInit()` y ponga 20, grabe la modificación y acceda a la URL, ¿empieza el contador en 120?
- e) Cree una copia del fichero `declara2.jsp` en
`AppWeb/WebContent/jsp/declara3.jsp`
- f) Edite el fichero y cambie de sitio la declaración `int numeroAccesos=150`, justo al principio del primer párrafo. Grabe la modificación y acceda a la página. ¿Empieza ahora en 150?
- g) Acceda al código del Servlet generado en el directorio correspondiente y compruebe que aunque la etiqueta de declaración ha cambiado de sitio, el código java sigue igual.
- h) Cree una copia del fichero `declara2.jsp` en
`AppWeb/WebContent/jsp/declara4.jsp`
- i) Edite el fichero, cambie el nombre del método `jspInit()` por `jspinit()`. Grabe la modificación y acceda a la página. ¿Empieza ahora en 150? ¿Se llega a ejecutar el método `jspinit()`?
- j) Edite el fichero, cambie el nombre del método `jspinit()` por `_jspInit()`. Grabe la modificación y acceda a la página. ¿Por qué obtenemos un error?
- k) Acceda al código del Servlet generado en el directorio correspondiente y busque el método `_jspInit()` ¿cuántas veces aparece?.

5.4.2 Expresiones

Las expresiones son fragmentos de código Java con la forma `<%= expresión %>`, que se evalúan en el momento de la ejecución. Se convierte automáticamente el resultado a String y se inserta en el flujo de salida (`out.println`), mostrándose en el navegador. En general, dentro de una expresión podemos usar cualquier cosa que usaríamos dentro de un `System.out.print(expr)` ;

Generalmente, se utiliza para obtener valores dinámicos a ser incluidos en el HTML generado.

Ejemplos:

```
<%= "Tamano de cadena: " + cadena.length() %>

<%= new java.util.Date() %>

<%= Math.PI*2 %>
```

TAREAS

- Abra el fichero `AppWeb/WebContent/jsp/accesos.jsp` y observe su contenido.
- Abra en un navegador la URL correspondiente:
<http://localhost:8080/AppWeb/jsp/accesos.jsp>
- Pruebe a recargar la página (pulsando F5) para ver cómo aumenta el número de accesos (la variable conserva su valor entre sucesivas ejecuciones).
- Abra la web de gestión de Tomcat (<http://localhost:8080/manager>) y recargue la aplicación /AppWeb.
- Pruebe a recargar de nuevo la página, comprobando que el servlet que se había generado en la primera ejecución de la página JSP se ha vuelto a crear.
- Puede probar también los ejemplos de `accesos2.jsp` y `hora.jsp`.

AppWeb/WebContent/jsp/accesos.jsp

```
<%! int numeroAccesos=0; %>

<!DOCTYPE html>

<html>
  <body>
    <p>
      <%= "La pagina ha sido accedida "+(++numeroAccesos)+
        " veces desde el arranque del servidor" %>
    </p>
  </body>
</html>
```

AppWeb/WebContent/jsp/accesos2.jsp

```
<%! java.util.Date primerAcceso=new java.util.Date(); %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html>
  <body>
    <p>
      El primer acceso a la pagina se realizo en:
      <%= primerAcceso %>
    </p>
  </body>
</html>
```

AppWeb/WebContent/jsp/hora.jsp

```
<%!
  private String ahora()
  {
    return ""+new java.util.Date();
  }
%>

<!DOCTYPE html>

<html>
  <body>
    <%= ahora() %>
```

```
</body>
</html>
```

5.4.3 Scriptlets

Los scriptlets son fragmentos de código Java con la forma `<% código %>`. En general, podemos insertar cualquier código que pudiéramos usar dentro de una función Java (acceso completo a la API de Java). El código java se incluye en el método `_jspService` del servlet generado. Para acceder a la salida del navegador, usamos el objeto implícito `out`.

Ejemplos:

AppWeb/WebContent/jsp/tabla1.jsp

```
<html>
<body>
<table>

  <% for (int i=0;i<10;i++)
  {
  %>
    <tr><td> <%=i%> </td></tr>
  <% } %>
</table>
</body>
</html>
```

AppWeb/WebContent/jsp/tabla2.jsp

```
<%
out.println("<html>\n<body>\n<table>");

for (int i=0;i<10;i++)
  out.println("<tr><td>" + i + "</td></tr>");

out.println("</table>\n</body>\n</html>");

%>
```

TAREAS

- Mediante un navegador, acceda a las páginas JSP correspondientes a los ficheros anteriores para ver su funcionamiento:
<http://localhost:8080/AppWeb/jsp/tabla1.jsp>
<http://localhost:8080/AppWeb/jsp/tabla2.jsp>
- Visualice desde el navegador el código HTML de la página, para comparar el resultado de la ejecución de los ejemplos.
- Acceda al código del Servlet generado en el directorio correspondiente y busque dónde aparece lo que está en las etiquetas de scriptlets.

Si observamos los dos ejemplos anteriores (que hacen lo mismo), podría parecer que la segunda opción es más deseable, pero en general hay que evitar el uso de `out.println()` para elementos HTML. En un proyecto en el que trabajen programadores y diseñadores conjuntamente, hay que separar presentación y código, tanto como sea posible.

Dentro de un *scriptlet* podemos usar cualquier librería de Java, incluyendo las propias, lo cual hace que resulte muy sencillo construir interfaces *web* de entrada y salida para nuestras clases. Ejemplo (más adelante se explica `request.getParameter`):

```
<%
    String parametro1=request.getParameter("parametro1");
    String parametro2=request.getParameter("parametro2");

    MiClase miClase=new MiClase();

    String salida=miClase.procesa(parametro1, parametro2);
%>
<%= salida %>
```

IMPORTANTE: a diferencia de las declaraciones, el scriptlet se ejecuta en todos los accesos.

5.4.4 Comentarios

Para introducir comentarios en JSP, usaremos las marcas `<%-- comentario --%>`. Dentro de un *scriptlet* o declaración podemos usar comentarios siguiendo la sintaxis de Java.

```
<%-- Comentario JSP --%>
<!-- Comentario HTML -->

<%
// Comentario
/* Comentario */
%>
```

Los comentarios JSP son ignorados a la hora de generar la página web dinámica, y no llegan al navegador del cliente. En cambio, los comentarios HTML sí viajan en la respuesta hacia el cliente.

AppWeb/WebContent/jsp/comenta_tabla.jsp

```
<html>
<body>
<table>
<!-- Este comentario HTML se ve en el navegador
      inspeccionando el código -->
<%-- empieza el bucle para la tabla
      este comentario no llega al navegador --%>
<% for (int i=0;i<10;i++)
{
%>
    <!-- esto es una fila de la tabla -->
    <tr><td> <%=i%> </td></tr>
    <% } %>
</table>
</body>
</html>
```

TAREAS

- Mediante un navegador, acceda a la página JSP correspondiente al fichero anterior.
- Visualice desde el navegador el código HTML de la página.
- Compruebe que llegan los comentarios HTML pero no los de JSP.

5.5 Ejercicios propuestos

5.5.1 Tabla bicolor

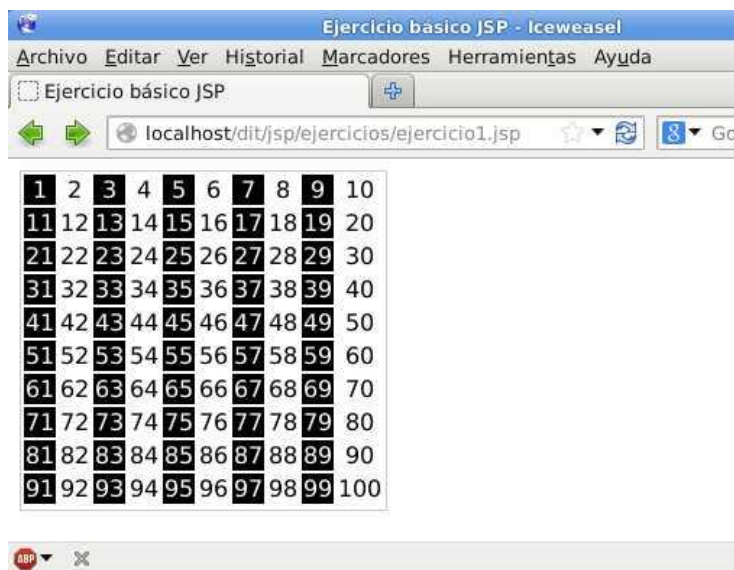
En este ejercicio, se pide mostrar en la ventana del navegador una tabla HTML de 10 por 10 elementos, con los números del 1 al 100. Las celdas que contengan un número par, deberán tener un color de fondo y fuente diferente a las que contengan un número impar. Para ello podrá utilizarse el siguiente código CSS:

```
<style>
  table {
    border:1px solid #ccc;
    font-family:sans-serif;
    text-align:center;
  }

  table td.par {
    background-color:#fff;
    color:#000;
  }

  table td.impar {
    background-color:#000;
    color:#fff;
  }
</style>
```

El resultado debe ser el siguiente:

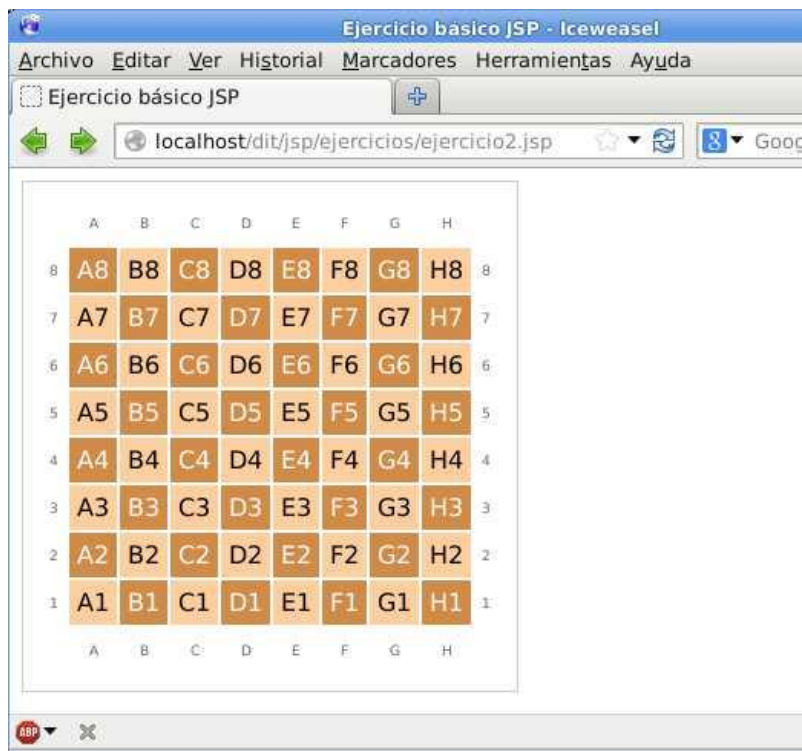


5.5.2 Tablero de ajedrez

Este ejercicio es una variación del ejercicio anterior. El ejercicio consiste en construir un tablero de ajedrez en HTML. La filas del tablero se enumeran con números (del 1 al 8) y las columnas con letras (de la A a la H). Los valores de los números y las letras indicadas, deberán almacenarse en 2 arrays:

- Filas: (1, 2, 3, 4, 5, 6, 7, 8)
- Columnas: ('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H')

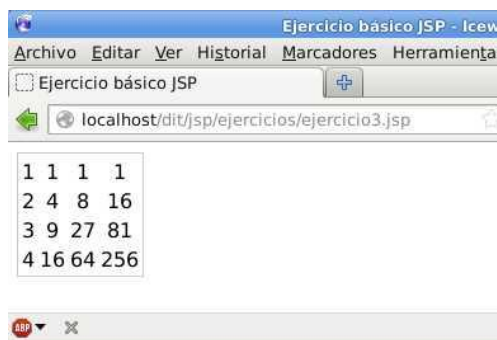
Recorreremos ambos arrays para generar el tablero. El resultado deberá ser el siguiente:



NOTA: el contenido de cada celda deberá ser la posición de dicha celda en el tablero (compuesta por el número de la fila y la letra de la columna).

5.5.3 Tabla potencias

Mostrar una tabla de 4 por 4 elementos, que muestre las primeras 4 potencias de los números del uno 1 al 4 (hacer una función que calcule las potencias). El resultado deberá ser el siguiente:



5.6 Etiquetas de Acción JSP (Action Tags o Standard Actions)

Las etiquetas de acción proporcionan funcionalidad adicional para desarrollar una página JSP, y presentan las siguientes características:

- Usan etiquetas al estilo XML, y se construyen mediante parejas *atributo/valor*.
- Se pueden crear otros usando bibliotecas de etiquetas (*tag libraries*).
- Emplean, modifican o crean objetos.

Las principales son:

Etiquetas de Acción JSP (Action Tag)	Descripción
<code>jsp:include</code>	Para incluir el resultado de un recurso en tiempo de ejecución, que puede ser una página HTML, JSP u otro fichero.
<code>jsp:param</code>	Para pasar parámetros a un recurso llamado con <code>jsp:include</code> o con <code>jsp:forward</code>
<code>jsp:forward</code>	Para reenviar la petición a otro recurso.
<code>jsp:useBean</code>	Para obtener el objeto Javabeen de un ámbito dado o crear un nuevo objeto Javabeen.
<code>jsp:getProperty</code>	Para obtener una propiedad de un Javabeen, utilizado con la acción <code>jsp:useBean</code>
<code>jsp:setProperty</code>	Para establecer la propiedad de un objeto Javabeen, utilizado con la acción <code>jsp:useBean</code> .

Las acciones más empleadas son las referentes al manejo de JavaBeans (`jsp:useBean`, `jsp:getProperty` y `jsp:setProperty`), las cuales veremos en el apartado correspondiente a JavaBeans, y las acciones `jsp:include` y `jsp:forward`, que veremos a continuación.

5.6.1 Etiqueta de Acción Include

La acción `jsp:include` incluye otro resultado de otro recurso en la página JSP. No confundir con la anteriormente vista directiva `include` (`<%@ include`, que era equivalente al `#include` del lenguaje C). Ejemplo:

```
< jsp:include page="header.jsp" />
```

Es posible pasar parámetros al recurso incluido mediante `jsp:param`. Ejemplo (fíjese que ahora no se cierra la etiqueta `include` hasta el final):

```
< jsp:include page="header.jsp">
    <jsp:param name="parametro" value="valor" />
</jsp:include>
```

La diferencia entre la directiva `include` y la acción `include` es que la directiva `include` añade el contenido del recurso seleccionado al código del servlet generado en el momento de la traducción, mientras que la acción `include`, que ocurre en tiempo de ejecución, lo que incluye es el resultado del recurso seleccionado (es como una llamada a subrutina). Además, con la acción `include` es posible pasar parámetros al recurso (no con la directiva `include`, en la que no tendría sentido)

Cuando se incluye un recurso estático como una cabecera, un pie de página o ficheros de imágenes, se emplearía la **directiva `include`** (`<%@ include`) para no tener que reescribir código, pero si se desea incluir recursos dinámicos y que requieren parámetros para el procesamiento, sería necesario utilizar la **etiqueta de acción `include`** (`< jsp:include`).

Veamos la diferencia con un ejemplo donde se hace referencia a un mismo recurso, primero con la directiva `include` y luego con la etiqueta de acción `include`. Cuando se usa la directiva, es como si se escribiera ahí (y se usa en la traducción). Cuando se usa la etiqueta de acción, se ejecuta el recurso y el resultado es lo que aparece ahí. Fíjese también que el atributo de la directiva es `file`, y en cambio en la etiqueta de acción es `page`.

AppWeb/WebContent/jsp/ejemploInclude/pagina1.jsp

```
<!DOCTYPE html>
<html>
<body>
<p>Uso de DIRECTIVA include con anexo.jsp<br/>
    El contador de esta página (pagina1.jsp):<br/>
<%@ include file="anexo.jsp" %>
</p>
<p>Uso de ETIQUETA DE ACCIÓN include con anexo.jsp<br/>
    El contador del anexo
    (cuenta tanto los accesos desde pagina1 como pagina2):<br/>
<jsp:include page="anexo.jsp" />
</p>
</body>
</html>
```


AppWeb/WebContent/jsp/ejemploInclude/pagina2.jsp

```
<!DOCTYPE html>
<html>
<body>
<p>Uso de DIRECTIVA include con anexo.jsp<br/>
    El contador de esta página (pagina2.jsp):<br/>
<%@ include file="anexo.jsp" %>
</p>
<p>Uso de ETIQUETA DE ACCIÓN include con anexo.jsp<br/>
    El contador del anexo
    (cuenta tanto los accesos desde pagina1 como pagina2):<br/>
<jsp:include page="anexo.jsp" />
</p>
</body>
</html>
```

AppWeb/WebContent/jsp/ejemploInclude/anexo.jsp

```
<%! int contador=0; %>
<br/>
contador = <%= ++contador %>
```

TAREAS

- Acceda a la URL <http://localhost:8080/AppWeb/jsp/ejemploInclude/pagina1.jsp> varias veces.
- Acceda a la URL <http://localhost:8080/AppWeb/jsp/ejemploInclude/pagina2.jsp> varias veces.
- Observe que hay 3 contadores: un contador para pagina1, un contador para pagina2 y otro para anexo. Tanto en pagina1 como en pagina2 aparecen primero el contador propio y luego el de anexo (que cuenta los accesos a ambas páginas).
- Acceda al código del Servlet de pagina1, de pagina2 y de anexo en el directorio correspondiente y observe los 3 contadores.
- Observe que la etiqueta de acción include se convierte en una llamada al Servlet de anexo.

5.6.2 Etiqueta de Acción Forward

La etiqueta de acción “jsp:forward” se emplea para reenviar la petición a otro recurso para ser manejada. Es parecida a la etiqueta de acción include, pero en este caso es como una llamada sin vuelta. También se pueden pasar parámetros. Ejemplo de uso:

```
<jsp:forward page="login.jsp" />
```

TAREAS

- Analice el contenido del fichero AppWeb/WebContent/jsp/ejemplo_forward.jsp y acceda a la URL http://localhost:8080/AppWeb/jsp/ejemplo_forward.jsp para comprobar el funcionamiento.
- Compruebe que al usar <jsp:forward se pierde todo lo que se hubiera preparado para la salida.
- Acceda al código del Servlet en el directorio correspondiente y observe en qué se convierte <jsp:forward.

5.7 Objetos implícitos

Como ya se ha presentado anteriormente, en JSP se dispone de 9 objetos implícitos (variables predefinidas), que permiten acceder a diferente información y realizar diversas acciones. Estos objetos los crea y presenta el contenedor de JSP, y los hace disponibles en cada página para ser empleados sin necesidad de definirlos previamente.

5.7.1 Atributos en los objetos implícitos

De los 9 objetos implícitos:

- request
- session
- application
- pageContext

permiten almacenar y recuperar atributos que son parejas nombre-objeto (el nombre sirve para identificar el objeto). El uso de atributos es un mecanismo muy útil para compartir información entre las distintas partes de una aplicación web.

Según se almacene un atributo usando (o a través de) un objeto u otro, así serán accesibles en distintos ámbitos:

Ámbito (id)	Objetos con acceso	Accesible desde
APPLICATION_SCOPE	application pageContext	Todo el código de la aplicación
SESSION_SCOPE	session pageContext	Todo el código que procese las peticiones de una misma sesión
REQUEST_SCOPE	request pageContext	Todo el código que procese una petición. Por ejemplo, podrían estar involucrados varios servlets por inclusión o redirección
PAGE_SCOPE	pageContext	El servlet y petición actual

Los siguientes objetos implícitos:

- request
- session
- application

tienen los siguientes métodos para almacenar y recuperar atributos:

<code>void setAttribute(String name, Object o)</code>	Almacena un atributo como una pareja nombre - objeto.
<code>Object getAttribute(String name)</code>	Devuelve el valor del atributo nombrado, como un objeto, o null si no existe ningún atributo del nombre dado.
<code>Enumeration<String> getAttributeNames()</code>	Devuelve una enumeración que contiene los nombres de los atributos disponibles.
<code>void removeAttribute(String name)</code>	Elimina un atributo

Además, el siguiente objeto implícito:

- `pageContext`

permite almacenar y recuperar atributos en los ámbitos de los 3 anteriores y también en el ámbito de la página, dependiendo de un parámetro adicional que puede tomar 1 de los 4 siguientes valores (están definidos en el objeto):

Valor del parámetro	Ámbito
REQUEST_SCOPE	Petición
SESSION_SCOPE	Sesión
APPLICATION_SCOPE	Aplicación
PAGE_SCOPE	Página

El ámbito de la página (PAGE_SCOPE) es el más restrictivo y lo que se almacene en ese ámbito se pierde cuando termina de ejecutarse la página (es el equivalente a las variables locales de las funciones en C). El almacenar información en este ámbito tiene sentido cuando es información perecedera (como las variables locales o variables auxiliares) pero la ventaja es que se puede usar esa información mediante las etiquetas de javabeans y también mediante expression language (se verán más adelante).

El objeto implícito `pageContext` tiene los siguientes métodos (similares a los de los otros 3) donde el último parámetro indica el ámbito:

<code>void setAttribute(String name, Object o, int scope)</code>	Almacena un atributo en un ámbito
<code>Object getAttribute(String name, int scope)</code>	Devuelve atributo de un ámbito
<code>Enumeration<String> getAttributeNamesInScope(int scope)</code>	Devuelve enumeración con nombres de atributos en un ámbito
<code>Void removeAttribute(String name, int scope)</code>	Elimina atributo en un ámbito

Nota: Los métodos anteriores están sobrecargados, de tal forma que si no se da el último parámetro es como si se le diera el valor PAGE_SCOPE. Los nombres son iguales excepto `getAttributeNamesInScope()`.

Además tiene otros métodos como:

<code>Object findAttribute(String name)</code>	Busca el atributo, por orden, en los ámbitos: página, petición, sesión y aplicación.
<code>int getAttributesScope(String name)</code>	Devuelve el ámbito en que un atributo se ha definido

En el apartado dedicado a `pageContext` se realizarán tareas relacionadas con sus métodos.

Veamos ahora cada uno de los 9 objetos implícitos.

5.7.2 request

Es una instancia de un objeto que implementa la interfaz:

```
javax.servlet.http.HttpServletRequest
```

Cada vez que un cliente pide una página, el motor JSP crea un nuevo objeto (nuevo para cada petición) que permite acceder a la información que viaja dentro del mensaje HTTP de la petición. Su uso principal es, además de para acceder a la información de la petición, para que se añada información que pueda ser procesada por otro recurso (por ejemplo un recurso accedido mediante `jsp:include` o `jsp:forward`).

TAREAS

- Consulte la documentación de la interfaz en:
<http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html>
- ¿Qué es lo que devuelve el método `getMethod()` ?
- ¿Qué es lo que devuelve el método `getRemoteAddr()` ?
- El método anterior, ¿es heredado de otra interfaz?
- Busque los métodos `getHeaderNames()` y `getHeader()` que se van a usar en los ejemplos que verá a continuación.
- Para averiguar los métodos que puede usar con el valor devuelto por `getHeaderNames()`, acceda a su documentación haciendo click sobre él (en Enumeration).

Observe en el siguiente ejemplo la diferencia entre el método `getHeaderNames()` y `getHeader()`. El primero devuelve los nombres de todas las cabeceras (en un Enumeration cuyos elementos son String que se puede recorrer fácilmente con los métodos de la interfaz Enumeration). Al segundo se le pasa el nombre de la cabecera y devuelve el valor.

AppWeb/WebContent/jsp/request.jsp

```
<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>HTTP Header Request</title>
</head>
<body>
<h2>Cabeceras en la petición HTTP</h2>
<table border="1">
<tr>
<th>Header Name</th><th>Header Value(s)</th>
</tr>
<%
    Enumeration<String> nombresCabecera = request.getHeaderNames();
    while(nombresCabecera.hasMoreElements()) {
        String nombreCab = (String)nombresCabecera.nextElement();
        out.print("<tr><td>" + nombreCab + "</td>\n");
        String valorCab = request.getHeader(nombreCab);
        out.println("<td> " + valorCab + "</td></tr>\n");
    }
%>
</table>
</body>
</html>
```

AppWeb/WebContent/jsp/request3.jsp

```
<html>
  <body>
    <br/><br/>
    Su IP: <%=request.getRemoteAddr()%>
    <br/>
    Su Puerto: <%=request.getRemotePort()%>
    <br/>
    Su nombre de host: <%= request.getRemoteHost() %>
  </body>
</html>
```

TAREAS

- Abra en un navegador las URLs correspondientes:
<http://localhost:8080/AppWeb/jsp/request.jsp>
<http://localhost:8080/AppWeb/jsp/request3.jsp>
- Edite el fichero request3.jsp y haga que muestre también el método (Method) usado en la petición y la cadena que está al final de la URL (QueryString).
- Acceda desde el navegador:
<http://localhost:8080/AppWeb/jsp/request3.jsp?nombrePara=valorP>
 y compruebe que muestra lo que está después de la “?”

5.7.2.1 Formularios y request

Para obtener los valores de los parámetros de un formulario, se pueden usar los métodos `getParameter()` y `getParameterValues()` (se verá la diferencia al realizar la tarea).

Observe en los siguientes ficheros JSP: `form.jsp` contiene el formulario (contiene sólo html) y `procesa_form.jsp` contiene lo necesario para procesarlo.

AppWeb/WebContent/jsp/form.jsp

```
<html>
<head>
<title>Formulario JSP</title>
</head>

<body>

  <form action="procesa_form.jsp">
    <input type="text" name="parametro1" value="valor por defecto"/>
    <br/>
    <input type="password" name="clave"/>
    <br/>
    <textarea name="parametro2">Texto por defecto</textarea>
    <br/>
    <select name="selectMultiple" multiple="multiple">
      <option value="1">Uno</option>
      <option>Dos</option>
      <option>Tres</option>
      <option>Cuatro</option>
    </select>
    <input type="submit"/>
  </form>
</body>
</html>
```

AppWeb/WebContent/jsp/procesa_form.jsp

```

<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>Formulario JSP</title>
</head>
<body>
  <h2>Valor de:</h2>
  <p>parametro1: <%= request.getParameter("parametro1") %> </p><br/>
  <p>parametro2: <%= request.getParameter("parametro2") %> </p><br/>
  <p>clave: <%= request.getParameter("clave") %> </p><br/>
  <p>selectMultiple:</p>
  <%
    String[] seleccion= request.getParameterValues("selectMultiple");
    if (seleccion!=null) {
      for (int i=0;i<seleccion.length;i++)
      {
        <%>
        <%= seleccion[i] %>
        <br/>
        <%
      }
    }
  <%>
</body>
</html>

```

TAREAS

- Acceda desde un navegador a <http://localhost:8080/AppWeb/jsp/form.jsp> y compruebe el funcionamiento. ¿Cuándo se usa `getParameterValues()`?
- Busque en la documentación de la interfaz `HttpServletRequest` los métodos usados en `procesa_form.jsp`. Busque y pruebe los métodos que devuelven un `String`.
- Añada otros campos al formulario, como `radio-buttons` y pruebe a obtener los valores seleccionados.
- Edite el fichero `form.jsp` y copie al final del body (dentro) el contenido del body de `procesa_form.jsp` (después de `</form>` estará `<h2>`). Elimine el atributo `action` de `form` (debe quedar `<form>`). Acceda desde el navegador a <http://localhost:8080/AppWeb/jsp/form.jsp>
- ¿Qué es lo que ocurre? Recuerde que si el atributo `action` no existe o está vacío, al hacer click sobre `submit` se accede al mismo recurso, y en este caso la petición contiene `GET /AppWeb/jsp/form.jsp`. Compruébelo con `wireshark` o con `F12` en el navegador.

5.7.3 response

Es una instancia de un objeto que implementa la interfaz:

```
javax.servlet.http.HttpServletResponse
```

Junto con el objeto request, el servidor crea un objeto para representar la respuesta al cliente, que es este objeto response. Este objeto tiene los métodos para tratar la creación de nuevas cabeceras HTTP. Es posible, a través de este objeto, añadir cookies, marcas de tiempo, códigos de estado HTTP, etc.

Cuando un servidor web responde a una petición HTTP de un navegador, la respuesta consiste en una línea de estado (por ejemplo HTTP/1.1 200 OK), algunas cabeceras de la respuesta (por ejemplo Content-Type: text/html), una línea en blanco, y el documento en sí de repuesta (en el cuerpo del mensaje HTTP).

TAREAS

a) Consulte la documentación de la interfaz en:

<http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html>

b) Busque el método setIntHeader() que se va a usar en el ejemplo que verá a continuación.

Ejemplo en el que se añade la cabecera "Refresh: 5" a la respuesta HTTP:

AppWeb/WebContent/jsp/response.jsp

```
<%@ page import="java.io.*,java.util.*" %>
<html>
<head>
<title>Auto Refresh Header</title>
</head>
<body>
<h2>Auto Refresh Header</h2>
<%
    // Ajustar autorrefresco
    response.setIntHeader("Refresh", 5);
    // Obtener hora actual
    Calendar calendar = new GregorianCalendar();
    String am_pm;
    int hour = calendar.get(Calendar.HOUR);
    int minute = calendar.get(Calendar.MINUTE);
    int second = calendar.get(Calendar.SECOND);
    if(calendar.get(Calendar.AM_PM) == 0)
        am_pm = "AM";
    else
        am_pm = "PM";
    String CT = hour+":"+minute+":"+second+" "+am_pm;
    out.println("Hora actual con Calendar: " + CT + "\n");
    %>
<br/>
<%
    out.println("Hora actual con Date: " + new java.util.Date() +
"\n");
    %>
</body>
</html>
```

TAREAS

- Abra en un navegador la URL <http://localhost:8080/AppWeb/jsp/response.jsp> y observe cómo cambia la hora cada 5 segundos, debido a que el navegador refresca la página cada 5 segundos (nueva petición cada vez), según está especificado en la cabecera HTTP “Refresh” de la respuesta del servidor (aunque no es estándar).
- Se ha mostrado el uso del método `setIntHeader()`. Pruebe otros métodos como `setStatus()`, etc.
- Fíjese que se ha usado `out.println()`, pero compruebe que no se ha inicializado la variable `out`. En el próximo apartado encontrará la explicación.

5.7.4 out

Es un objeto de la clase `javax.servlet.jsp.JspWriter`, y es empleado para enviar contenido en una respuesta (se usa para imprimir la salida html).

Puede consultar: <https://docs.oracle.com/javaee/7/api/javax/servlet/jsp/JspWriter.html>

El objeto inicial `JspWriter` es instanciado de forma diferente dependiendo de si la página utiliza un buffer o no. El buffer puede ser desactivado mediante el atributo `buffered='false'` en la directiva `page`.

Métodos principales:

Método	Descripción
<code>out.print(dataType dt)</code>	Imprime un valor
<code>out.println(dataType dt)</code>	Imprime un valor y añade el carácter nueva línea
<code>out.flush()</code>	Vacía el buffer (fuerza la salida)

Ejemplo:

```
<%
  out.print("cadena");
  out.println("cadena");
%>
```

En una página JSP, el contenedor web (Tomcat) obtiene la referencia del objeto implícito `out` mediante un método del objeto implícito `pageContext` (este objeto se explica más adelante):

```
javax.servlet.jsp.JspWriter out = null;
out = pageContext.getOut();
```

en cambio, cuando se codifica manualmente un Servlet, dentro del método `doGet()` o `doPost()`, se obtiene un objeto equivalente (allí no es implícito) mediante un método de `response` (que es parámetro de `doGet()` o `doPost()`):

```
PrintWriter salida = response.getWriter();
```


TAREAS

- a) Muestre por pantalla el código del Servlet (`response.jsp.java`) en el directorio correspondiente generado en la tarea anterior (al acceder a `response.jsp`) y compruebe que el valor asignado al objeto implícito `out` se obtiene a partir de `pageContext`.
- b) Muestre por pantalla el Servlet que se vio como ejemplo en:
`AppWeb/src/ServletFecha.java`
y compruebe que el objeto similar a `out` se obtiene a partir de `response`.

5.7.5 session

Es un objeto que implementa la interfaz `javax.servlet.http.HttpSession`. Permite acceder a todo lo relacionado con la sesión asociada a la petición de un cliente (desde todas las peticiones que provienen de un mismo cliente). A través de este objeto se puede, entre otras cosas, guardar objetos que serán accesibles desde cualquier JSP de la misma sesión. También permite invalidarla (la siguiente petición pertenecería a una nueva sesión).

TAREAS

- a) Consulte la documentación de la interfaz en:
<http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpSession.html>
y compruebe que contiene los métodos para gestionar atributos: `setAttribute()`, `getAttribute()`, `getAttributeNames()`, `removeAttribute()`
- b) Busque también los siguientes métodos:
 - `invalidate()`
 - `isNew()`
 ¿qué ocurre con los atributos de una sesión cuando se usa el método `invalidate()`?

Por la importancia de las sesiones, veremos un ejemplo completo de su uso en una sección posterior dedicada al efecto.

5.7.6 application

Es un objeto que implementa la interfaz `javax.servlet.ServletContext`. Este objeto es común para toda la aplicación web y, entre otras cosas, nos permite almacenar información que será accesible desde todas las páginas de la aplicación web, independientemente de la sesión.

Nota: la nomenclatura `ServletContext` es engañosa, y realmente se corresponde con el ámbito de aplicación.

TAREAS

- a) Consulte la documentación de la interfaz en:
<http://docs.oracle.com/javaee/7/api/javax/servlet/ServletContext.html>
y compruebe que contiene los métodos para gestionar atributos: `setAttribute()`, `getAttribute()`, `getAttributeNames()`, `removeAttribute()`
- b) Busque también el siguiente método:
 - `getServletContextName()`
 ¿con qué elemento de `web.xml` está relacionado? Cree una página JSP que muestre el valor devuelto.

En el siguiente ejemplo se van almacenando las direcciones IP en un objeto de la clase `HashSet`, que no almacena elementos repetidos y puede ser recorrido fácilmente con un objeto `Iterator`. Puede consultar la documentación en:

<http://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html>

AppWeb/WebContent/jsp/ej_application.jsp

```

<%@ page session="true" %>
<%
    java.util.HashSet<String> direcciones=
        (java.util.HashSet<String>)
application.getAttribute("direcciones");

    if (direcciones==null){
        direcciones=new java.util.HashSet<String>();
        application.setAttribute("direcciones", direcciones);
    }
    direcciones.add(request.getRemoteAddr());
%>

<html>
<body> El servidor fue accedido desde las siguientes direcciones IP:
<br>
<%
    java.util.Iterator<String> e=direcciones.iterator();
    while (e.hasNext())
    {
        %>
        <%= e.next() %>
        <br>
        <%
    }
    %>
</body>
</html>

```

AppWeb/WebContent/jsp/hits.jsp

```

<html>
<head>
<title>Contador de visitas </title>
</head>
<body>
<%
    Integer contador =
        (Integer)application.getAttribute("cuentaVisitas");
    if( contador ==null || contador == 0 ){
        /* Primera visita */
        out.println(";Bienvenido a mi website!");
        contador = 1;
    }else{
        /* Siguietes visitas */
        out.println(";Bienvenido de nuevo a mi website!");
        contador++;
    }
    application.setAttribute("cuentaVisitas", contador);
%>
<p>Numero total de visitas: <%= contador%></p>
</body>
</html>

```

TAREAS

- a) Abra en un navegador las URLs http://localhost:8080/AppWeb/jsp/ej_application.jsp y http://IP_eth0:8080/AppWeb/jsp/ej_application.jsp, donde **IP_eth0** la puede obtener con el comando `ifconfig` o con el comando `ip`. Pruebe a recargar la página cada vez.
- b) Edite el fichero `ej_application.jsp` y grabe alguna modificación (por ejemplo poga su UVUS).
- c) Vuelva a acceder a través de `localhost` y de `IP_eth0`. Compruebe que siguen apareciendo las dos direcciones a pesar de que Tomcat habrá instanciado un nuevo servlet, ya que el atributo está en el ámbito de la aplicación.
- d) Usando el gestor de aplicaciones de Tomcat, recargue la aplicación web `/AppWeb` como se ha explicado anteriormente (<http://localhost:8080/manager>). Vuelva a acceder a través de `localhost` y de `IP_eth0`. Compruebe que ahora sí aparecen las direcciones como al principio, ya que el atributo se ha borrado y se ha vuelto a crear.
- e) Acceda también a la URL <http://localhost:8080/AppWeb/jsp/hits.jsp> y compruebe que el contador se incrementa.
- f) Copie el fichero `hits.jsp` en `hits2.jsp` y modifique el título para poder diferenciarlo. Acceda a <http://localhost:8080/AppWeb/jsp/hits2.jsp> y compruebe que el contador sigue incrementándose, ya que el atributo está compartido a través del ámbito de la aplicación. Recargue la aplicación `/AppWeb` y compruebe que el contador se inicializa.
- g) Examine ahora las páginas JSP contenidas en `AppWeb/WebContent/jsp/application/index.jsp` y `modificaAtributos.jsp`. Observe también el contenido de `AppWeb/WebContent/WEB-INF/web.xml`, el descriptor de despliegue, especialmente la sección `<servlet>`.
- h) Abra ahora la URL <http://localhost:8080/AppWeb/jsp/application/index.jsp> y observe el funcionamiento:
 - Fíjese en el valor de los parámetros de contexto presentados.
 - Observe también los atributos de la aplicación, aparecen valores de las ejecuciones de las anteriores páginas JSP. Al final de la tabla hay un enlace para añadir nuevos atributos: añada cuantos estime necesarios y pruebe el funcionamiento.

5.7.7 config

Es un objeto que implementa la interfaz `javax.servlet.ServletConfig`. Permite acceder a parámetros de inicialización del servlet o motor JSP y a su contexto, tales como rutas o localización de ficheros, etc.

TAREAS

- a) Consulte la documentación en: <http://docs.oracle.com/javaee/7/api/javax/servlet/ServletConfig.html> y busque el método `getServerName()`, que devuelve el nombre del servlet, que es la cadena contenida en el elemento `<servlet-name>` definida en el fichero `WEB-INF/web.xml` o el nombre de la clase.
- b) Examine la página JSP `AppWeb/WebContent/jsp/config/index.jsp` y acceda a la URL <http://localhost:8080/AppWeb/jsp/config/index.jsp> para comprobar el funcionamiento del objeto `config`.

5.7.8 pageContext

Es un objeto de la clase `javax.servlet.jsp.pageContext`, y además de permitir almacenar información en el ámbito de la página (recuerde que es el más restrictivo y es como las variables locales de las funciones en C), permite acceder a otros objetos. El objeto `pageContext` almacena referencias

a los objetos request y response de cada petición. Los objetos application, config, session y out son accedidos a través de métodos del objeto pageContext.

El objeto también contiene información sobre las directivas empleadas en la página JSP, incluyendo la información del búffer, y la errorPageURL.

Tal como se explicó en el apartado de “Atributos en los objetos implícitos“, la clase pageContext define varios campos, incluyendo PAGE_SCOPE, REQUEST_SCOPE, SESSION_SCOPE, y APPLICATION_SCOPE, que identifican los cuatro ámbitos.

Ejemplos de uso:

- Almacenar en ámbito de página (el método está sobrecargado, y la página es el ámbito por defecto):

```
pageContext.setAttribute("clave", obj, PageContext.PAGE_SCOPE);  
pageContext.setAttribute("clave", obj);
```

- Almacenar en ámbito de petición (a través del objeto implícito request se puede hacer de forma equivalente):

```
pageContext.setAttribute("clave", obj, PageContext.REQUEST_SCOPE);  
request.setAttribute("clave", objeto);
```

- Almacenar en ámbito de sesión (a través del objeto implícito session se puede hacer de forma equivalente):

```
pageContext.setAttribute("clave", obj, PageContext.SESSION_SCOPE);  
session.setAttribute("clave", objeto);
```

- Almacenar en ámbito de aplicación (a través del objeto implícito application se puede hacer de forma equivalente):

```
pageContext.setAttribute("clave", obj, PageContext.APPLICATION_SCOPE);  
application.setAttribute("clave", objeto);
```

TAREAS

- a) Consulte la documentación en:
<http://docs.oracle.com/javaee/7/api/javax/servlet/jsp/PageContext.html>
y compruebe que el método `setAttribute()` está heredado de `JspContext`, que está sobrecargado (permite 2 ó 3 argumentos), y que si no se da el tercer argumento es equivalente a darle `PageContext.PAGE_SCOPE`.
- b) Examine el contenido de `AppWeb/WebContent/jsp/pageContext/`.
- c) Acceda a <http://localhost:8080/AppWeb/jsp/pageContext/index.jsp>
- d) Compruebe el funcionamiento de `atributos.jsp` en el enlace “Ver ejemplo”.
 - ¿por qué el atributo `action` del formulario está vacío?
 - ¿qué método se utiliza para obtener los datos del formulario y con qué objeto implícito?
 - ¿aparecen más objetos implícitos? ¿cuál o cuáles?
 - ¿qué devuelve el método `getAttributeNamesInScope`?
 - ¿qué devuelve el método `getAttributeScope`?
 - Cree dos atributos con el mismo nombre y distinto valor, uno en el ámbito de sesión y otro en el de aplicación respectivamente. Compruebe que están cada uno en su ámbito. Busque (ver ámbito) el nombre en todos los ámbitos ¿en cuál aparece? Borre el atributo del ámbito de sesión y búsquelo después ¿en qué ámbito aparece ahora?
 - ¿qué ocurre si intenta borrar un atributo inexistente? ¿el método que borra lo indica?
 - Modifique el formulario para que tenga un nuevo botón (encuentra) que busque el atributo en todos los ámbitos (de menor a mayor alcance) y devuelva el valor asociado o null si no lo encuentra (use un método que hace esto).
- e) Compruebe el funcionamiento del resto de fichero mediante el último enlace de `index.jsp`. Analice el contenido de los ficheros implicados: `redireccionador.jsp`, `incluido.jsp` y `nuevodestino.jsp`.
- f) Respecto a `redireccionador.jsp`:
 - ¿por qué el atributo `action` del formulario está vacío?
 - ¿qué es lo que hace cuando se llama sin parámetro y cuando se llama con parámetro?
 - ¿cuántos atributos se crean?
 - cuando se hace `forward` ¿qué ocurre con el `html` previo?
 - ¿qué ocurre con los atributos del ámbito `PAGE_SCOPE` cuando se cambia de `jsp`? Observe que cada página tiene su propio ámbito `PAGE_SCOPE`.
 - modifique los tres ficheros para que se creen y se muestren otros atributos (en el ámbito de sesión y de aplicación).
 - cambie el `include` y el `forward` por etiquetas de acción `jsp` (modifique el `html` para que sea tangible el cambio).

5.7.9 page

El objeto es una referencia a la instancia de la página. Es un sinónimo de `this`, no tiene utilidad en el estado actual de la especificación.

5.7.10 exception

El objeto `exception` es un envoltorio de la excepción arrojada por la página anterior. Se suele utilizar para generar una página de respuesta apropiada para la condición de error.

TAREAS

- a) Examine el contenido de `AppWeb/WebContent/jsp/exception/`.
- b) Acceda a <http://localhost:8080/AppWeb/jsp/exception/index.jsp> y compruebe el funcionamiento.

5.8 JavaBeans

5.8.1 Introducción

Un JavaBean, o simplemente “bean”, en el entorno JSP no es más que una **clase Java** que sigue ciertas convenciones en el nombrado de sus métodos, con el objeto de permitir que los motores JSP (como Tomcat) puedan conocer y acceder a las propiedades que contiene. Así, el JavaBean tiene propiedades (las cuales no son más que variables de esa clase) y, para cada una de ellas, deben definirse los métodos `getPropiedad()` y `setPropiedad()` que permiten leer y escribir el valor de la propiedad, respectivamente.

Las convenciones de codificación que exige la especificación para una clase JavaBeans son las siguientes:

- Una clase JavaBean debe tener un constructor sin argumentos.
- Una clase JavaBean no debería tener variables de instancia públicas.
- Para cada variable (o propiedad) deben crearse dos métodos cuyo nombre sea el nombre de la propiedad comenzando por letra mayúscula y precedidos por “get” (para método de lectura) o “set” (para método de escritura).

Así, un ejemplo de clase que sigue la convención Javabeen podría ser la siguiente (fíjese que lo importante no es el nombre de la variable privada, sino que al ser los métodos `setProp` y `getProp`, la propiedad se llama `prop`):

clase_javabeen.java

```
public class clase_javabeen {  
    public clase_javabeen() {}  
    private String propiedad = "valor";  
    public String getProp() {  
        return propiedad;  
    }  
    public void setProp(String p) {  
        propiedad = p;  
    }  
}
```

De esta forma, las propiedades del Javabeen se exponen públicamente mediante esos métodos, cuyo nombre es conocido de antemano al seguir la regla de nombrado Javabeen. Esto confiere la característica de introspección que no es más que la capacidad que tienen los motores JSP como Tomcat de acceder directamente a las variables de la clase, al conocerse los métodos necesarios para ello.

Adicionalmente a lo anterior, las clases Javabeen tienen la característica de que pueden implementar la interfaz “`java.io.Serializable`”. Se denomina serialización al proceso de escribir el estado de un objeto en un flujo de octetos, permitiendo con ello almacenar objetos en archivos, transmitirlos por una red de comunicaciones y utilizarlos en aplicaciones distribuidas. Por otro lado, se conoce como persistencia a la capacidad que tiene un objeto para existir más allá de la ejecución de un programa (en un archivo, por ejemplo). De acuerdo con ello, puede observarse cómo la serialización es la clave para

la implementación de la persistencia, ya que permite escribir un objeto en un flujo y volver a leerlo posteriormente.

En un flujo de bytes solamente se puede guardar y restablecer un objeto que implemente la interfaz `Serializable`. Esta interfaz no dispone de métodos, sólo sirve para indicar que una clase se puede serializar, para lo cual será necesario que todas las instancias de clases usadas sean de tipo primario o clases que también se puedan serializar (por ejemplo, esto no ocurre con la clase `Object`, pero sí con `String`). Aunque en esta asignatura no se va a usar la persistencia, se verá en asignaturas de cursos posteriores.

Consecuentemente con lo anterior, una definición de clase `Javabeen` que presente la capacidad de persistencia podría ser la siguiente:

`clase_javabeen_persistente.java`

```
package paquete;
import java.io.Serializable;

public class clase_javabeen_persistente implements Serializable {
    private static final long serialVersionUID = 1L;

    public clase_javabeen_persistente() { }

    private String propiedad = "valor";

    public String getPropiedad() {
        return propiedad;
    }
    public void setPropiedad(String prop) {
        propiedad = prop;
    }
}
```

Conforme con lo visto, el uso de Javabeans podría seguir haciéndose mediante la inclusión de código Java dentro del fichero JSP, creando por ejemplo una instancia del Javabeen del siguiente modo:

```
<% paquete.clase_javabeen instancia = new paquete.clase_javabeen ();
%>
```

o recuperar el valor de una variable mediante la llamada:

```
<%= instancia.getPropiedad() %>
```

No obstante, esto no implicaría ventaja substancial respecto al uso de los scriptlets. La principal ventaja de los Javabeen es que tanto la creación de la instancia de la clase Javabeen como el acceso a sus variables pueden realizarse mediante etiquetas JSP, eliminando así el código Java de la página JSP.

Las etiquetas de acción JSP (action tags) que permiten esto son:

a) `jsp:useBean`: crea (o si ya existe la usa) una instancia “nombre_ins” de la clase `Javabean` “clase”:

```
<jsp:useBean id="nombre_ins" class="clase" type="tipo" scope="ámbito" />
```

donde se distinguen los siguientes parámetros (`type` y `scope` son opcionales):

- `id`: especifica el nombre de la instancia del `JavaBean` con el que será conocido dentro del ámbito en el que se define.
- `class`: indica la clase `JavaBean` que se va a instanciar.
- `type`: indica otra clase o interfaz de la que deriva el bean, haciendo que éste deba usarse como tipo de datos de esa clase o interfaz. Su valor puede ser el mismo que el del parámetro “`class`”, una superclase de “`class`” o una interfaz implementada por “`class`”. Así, por ejemplo, si el `JavaBean` va a implementar la interfaz `Runnable` para ejecución en segundo plano, se indicaría como tipo `type="Runnable"`. Por omisión se toma el mismo valor que `class`.
- `scope`: determina el ámbito en el que puede usarse el `JavaBean`, pudiendo ser:
 - = “`page`” : (por omisión) es accesible únicamente dentro de la página en la que se define y para la petición actual.
 - = “`request`” : es accesible desde todas las páginas leídas a consecuencia de la solicitud `http request` recibida.
 - = “`session`” : es accesible desde todas las páginas que procesen peticiones dentro de la misma sesión (peticiones que provengan del mismo cliente, que ayuda a administrar datos específicos del mismo en la aplicación Web).
 - = “`application`” : es accesible desde todas las páginas de la misma aplicación Web.

b) `jsp:getProperty`: obtiene el valor de la propiedad de nombre “propiedad” de la instancia de clase `Javabean` “nombre_ins” (lo que hace es llamar a `nombre_ins.getProperty()`) y lo inserta en la salida HTML:

```
<jsp:getProperty name="nombre_ins" property="propiedad" />
```

c) `jsp:setProperty`: establece a “val” el valor de la propiedad de nombre “propiedad” de la instancia de clase `Javabean` “nombre_ins” (lo que hace es llamar a `nombre_ins.setProperty("val")`):

```
<jsp:setProperty name="nombre_ins" property="propiedad" value="val" />
```

Como ejemplo de uso de estas etiquetas, se crea una instancia fuente de la clase `FuenteBean`, se modifica el valor de la propiedad `codigo` de esa instancia fuente y se recupera a continuación dicho valor:

```
<jsp:useBean id="fuente" class="FuenteBean" />

<jsp:setProperty name="fuente" property="codigo" value="YO"/>

<jsp:getProperty name="fuente" property="codigo"/>
```


En el ejemplo anterior, la clase FuenteBean debe tener los métodos `setCodigo()` y `getCodigo()`.

Resumiendo, el uso de JavaBeans permite eliminar gran parte del código Java de las páginas JSP, de forma que casi toda la lógica de programación de la página se encapsule en JavaBeans (reduciendo la página a código HTML o XML) que, además, podrán ser reutilizados. Esas clases o código Java podrán ser compilados y depurados utilizando las herramientas habituales de programación, ventaja importante frente a la inserción de código Java dentro de la página (que tiene que ser depurado en la propia carga de la página).

Para mostrar el funcionamiento se presenta un ejemplo sencillo que se limita a llamar a la clase que implementa el JavaBean, recuperando en primer lugar el valor inicial del parámetro “saludo” del mismo (inicialmente “saludo = Hola”) y, posteriormente, cambiando dicho valor a uno distinto (al finalizar, “saludo = Adios”), recogiendo esos valores en el código HTML resultante. Se usan así dos ficheros:

- `ej_jbean.jsp`: página JSP que realiza las llamadas al código JavaBeans a través de directivas JSP.

AppWeb/WebContent/jsp/ej_jbean.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- jbean.jsp
Ejemplo para mostrar el uso de un JavaBean siguiendo la
convención de la especificación JavaBean
-->
<html>
<head>
<title>Titulo del Ejemplo: Pagina JSP basada en JavaBeans</title>
</head>

<body>
<!-- Indicamos que se desea crear una instancia del objeto jspJB -->
<jsp:useBean id="miBean" scope="page" class="ej_jbean.jspJB" />
<h2>Ejemplo: Pagina JSP basada en JavaBeans</h2>
<p>Saludo inicial:<br>
<!-- Se imprime el valor inicial de la propiedad Saludo -->
<jsp:getProperty name="miBean" property="saludo" />
<p>Saludo final:<br>
<!-- Se fija e imprime un nuevo valor para la propiedad Saludo -->
<jsp:setProperty name="miBean" property="saludo" value="Adios" />
<jsp:getProperty name="miBean" property="saludo" />
</body>
</html>
```

- `ej_jspJB.class`: clase Java que implementa el JavaBeans. Se ha incluido dentro del paquete “ej_jbean”. Su código fuente es (`jspJB.java`):

AppWeb/src/ej_jbean/jspJB.java

```
// jspJB.java
/* Clase Java que sigue la convencion JavaBean
* Bean que describe la propiedad saludo. Constructor por defecto, sin
```

```
* argumentos, necesario para que esta clase sea un JavaBean.
*/
package ej_jbean;
public class jspJB {
    private String saludo = "Hola";
    // Métodos set/get para asignar/recuperar el valor de la propiedad
    public void setSaludo( String _saludo ) {
        saludo = _saludo;
    }
    public String getSaludo() {
        return( saludo );
    }
    public String saludoCompleto() {
        return( saludo+", dit" );
    }
}
```

En este caso, al no necesitar pasar información inicial a ninguna clase Java, no resulta necesario el descriptor de la aplicación “web.xml”, por lo que en este caso se prescindirá de él con la intención de mostrar su no obligatoriedad en aplicaciones simples (puede probar a ejecutar este mismo ejemplo borrando previamente el fichero “web.xml”, restaurándolo posteriormente).

Puede observarse cómo el uso de JavaBeans reduce considerablemente la cantidad de código JSP insertado dentro de la página HTML, al tiempo que permite la reutilización de código Java. El uso de la directiva “include” también permite reducir parte del código Java insertado en la página, pero no ofrece las posibilidades de los JavaBeans, presentando un funcionamiento a nivel de programación Java totalmente diferente.

No obstante, a pesar de las mejoras que introducen los JavaBeans, el desarrollo de páginas mediante su uso aún sigue requiriendo ciertos conocimientos en programación Java (debido a que se están usando directivas JSP, no etiquetas HTML).

Para probar la aplicación realice las siguientes tareas.

TAREAS

- a) Compruebe que el fichero fuente `jspJB.java` se encuentra en:

```
AppWeb/src/ej_jbean/jspJB.java
```

Recuerde que puede acceder desde el Project Explorer de Eclipse:

```
AppWeb > Java Resources > src > ej_jbean > jspJB.java
```

- b) Compruebe que el fichero class `jspJB.class` (resultado de compilar el fichero java) se encuentra en:

```
AppWeb/build/classes/ej_jbean/jspJB.class
```

- c) Compruebe que el fichero class `jspJB.class` (copia del anterior) se encuentra en:

```
/home/dit/tomcat/webapps/AppWeb/WEB-INF/classes/ej_jbean/jspJB.class
```

- d) Modifique el fichero (siempre desde Eclipse) `jspJB.java` y grábelo (por ejemplo añadiendo su UVUS a la cadena “Hola”).
- e) Compruebe que Eclipse detecta el cambio, compila el fichero java, genera el fichero class, lo copia en tomcat y reinicia tomcat (observe la consola y si no es así reinicielo manualmente). Si no se usara in IDE como Eclipse, habría que realizar estas acciones manualmente, compilando con `javac`.
- f) Acceda a la página http://localhost:8080/AppWeb/jsp/ej_jbean.jsp y observe si se ha modificado el saludo inicial.
- g) Modifique el fichero (siempre desde Eclipse) `ej_jbean.jsp` y grábelo (por ejemplo añadiendo su UVUS a la cadena “Adios”).
- h) Si la modificación realizada no se muestra inmediatamente, usando el gestor de aplicaciones de Tomcat, recargue la aplicación web /AppWeb como se ha explicado anteriormente (<http://localhost:8080/manager>).
- i) Acceda al código del Servlet (`ej_jbean_jsp.java` o similar debido al carácter “_” dentro del nombre) en el directorio correspondiente de Tomcat:

```
/home/dit/tomcat/work/Catalina/localhost/AppWeb/org/apache/jsp/jsp
```

y observe en qué se traduce la etiqueta `<jsp:useBean id="miBean" scope="page" class="ej_jbean.jspJB" />`

- Declara una variable de nombre `miBean` y la inicializa a null
- Busca en el ámbito de página un atributo de nombre “`miBean`” y el objeto devuelto lo asigna a `miBean`
- Si no lo encuentra:
 - instancia un objeto de la clase `ej_jbean.jspJB`
 - guarda un atributo de nombre “`miBean`” con el objeto `miBean` en el ámbito de página

5.8.2 Asignación condicional en la creación

Cuando se usa la etiqueta `<jsp:useBean ... />`, puede ser que el `javabeen` ya exista o se cree.

Si se quiere que se le asigne un valor a una propiedad sólo si se crea, hay que usar la etiqueta `<jsp:setProperty ... />` en el cuerpo de `<jsp:useBean ...>` (ahora sin la barra de cierre) y añadir la etiqueta de cierre `</jsp:useBean>`. Todo lo que esté en el cuerpo se ejecutará sólo si se crea el `javabeen`.

Por ejemplo:

```
<jsp:useBean id="fuente" class="FuenteBean">
  <jsp:setProperty name="fuente" property="codigo" value="YO"/>
</jsp:useBean>
<jsp:getProperty name="fuente" property="codigo"/>
```

En el ejemplo anterior, si no existe el javabean de nombre “fuente” (en el ámbito de página, que es el que se usa por omisión) entonces se le asigna valor a la propiedad código.

TAREAS

- Modifique el fichero ej_bean.jsp para añadirle cuerpo a <jsp:useBean, de tal forma que quede:

```
<jsp:useBean id="miBean" scope="page" class="ej_bean.jspJB">
  <jsp:setProperty name="miBean" property="saludo" value="Nuevo" />
</jsp:useBean>
```

- Acceda a la página http://localhost:8080/AppWeb/jsp/ej_jbean.jsp
- Acceda al código del Servlet en el directorio correspondiente de Tomcat y observe en qué se traduce la etiqueta <jsp:useBean id="miBean" scope="page" class="ej_bean.jspJB"> con su cuerpo:
 - Declara una variable de nombre miBean
 - Busca en el ámbito de página un atributo de nombre “miBean” y el objeto devuelto lo asigna a miBean
 - Si no lo encuentra:
 - instancia un objeto de la clase ej_bean.jspJP
 - guarda un atributo de nombre “miBean” con el objeto miBean en el ámbito de página
 - además llama a un método donde se supone que asigna el valor “Nuevo” a la propiedad “saludo” del javabean “miBean”

5.8.3 Asignación de parámetros de un formulario a las propiedades de un javabean

Sea una página JSP que procesa los datos enviados mediante un formulario, y esta página usa un javabean.

Si al usar <jsp:setProperty para asignar el valor a una propiedad de un javabean se utiliza el atributo param (en vez de value) con un valor igual al nombre de un parámetro del formulario, entonces la propiedad toma el valor de ese parámetro.

```
<jsp:setProperty property="nombre"      name="personaActual" param="campoNombre"/>
<jsp:setProperty property="apellidos"   name="personaActual" param="campoApellidos"/>
<jsp:setProperty property="edad"        name="personaActual" param="campoEdad"/>
```

TAREAS

- Acceda al directorio `AppWeb/WebContent/jsp/javabeans`. El fichero `index.jsp` contiene un enlace a cada ejemplo presentado. Acceda a <http://localhost:8080/AppWeb/jsp/javabeans/index.jsp> para visualizar los ejemplos, y
- Acceda a Inicialización con parámetros de la petición (1)
- Compruebe que está accediendo a `persona_form1.jsp` y rellene el formulario.
- Envíe los datos del formulario y compruebe que está accediendo a `bean_ini_param1.jsp`
- Abra con un editor ambos ficheros y compruebe que los nombres de los campos del formulario (atributo `name` de `input`) coinciden con los valores de los atributos `param` de `jsp:setProperty`.
- Compruebe que el javabean usado está en:
`AppWeb/src/fast/Persona.java`
- Si añadiera un nuevo campo al formulario, tendría que modificar el javabean para que tuviera una nueva propiedad, pero ¿tendría que modificar el `jsp`?

Sea una página JSP que procesa los datos enviados mediante un formulario, y esta página usa un javabean en el que los nombres de las propiedades coinciden con los nombres de los campos del formulario.

Si al usar `<jsp:setProperty` de un javabean se utiliza el atributo `property` con un valor igual a `"*"`, entonces cada propiedad toma el valor del parámetro de nombre coincidente.

```
<jsp:setProperty property="*" name="personaActual"/>
```

TAREAS

- Acceda a <http://localhost:8080/AppWeb/jsp/javabeans/index.jsp>, y acceda a Inicialización con parámetros de la petición (2)
- Compruebe que está accediendo a `persona_form2.jsp` y rellene el formulario.
- Envíe los datos del formulario y compruebe que está accediendo a `bean_ini_param2.jsp`
- Compruebe que el javabean usado está en:
- `AppWeb/src/fast/Persona.java`
- Abra con un editor `persona_form2.jsp` y `Persona.java` y compruebe que los nombres de los campos del formulario (atributo `name` de `input`) coinciden con los valores de las propiedades del javabean.
- Compruebe que el atributo `param` de `jsp:setProperty` vale `"*"`
- Si añadiera un nuevo campo al formulario, tendría que modificar el javabean para que tuviera una nueva propiedad, pero ¿tendría que modificar el `<jsp:setProperty` del `jsp`?

5.8.4 Ejemplos

A continuación, puede ejecutar algunos ejemplos, estudiarlos y modificarlos.

- a) Acceda al directorio `AppWeb/WebContent/jsp/javabeans`. El fichero `index.jsp` contiene un enlace a cada ejemplo presentado. Acceda a <http://localhost:8080/AppWeb/jsp/javabeans/index.jsp> para visualizar los ejemplos.

Los distintos ejemplos hacen uso de los siguientes ficheros Java:

```
AppWeb/src/fast/BeanMinimo.java
AppWeb/src/fast/BeanPruebaScope.java
AppWeb/src/fast/Persona.java
```

- b) Céntrese en observar el contenido de las distintas páginas JSP ubicadas en `AppWeb/WebContent/jsp/javabeans/beans`:
- Ejemplo básico con scriptlets:
 - `basico_scriptlets.jsp`
 - Modifique `BeanMinimo` y añádale otras propiedades (numéricos, lógicos, etc). Cree otro bean, asigne las propiedades y muéstrelas.
 - Ejemplo: básico con etiquetas:
 - `basico_etiquetas.jsp`
 - Haga lo mismo que en el anterior, pero ahora con etiquetas.
 - Ejemplo de ámbitos page y request:
 - `bean_scope_page_vs_request.jsp`
 - `mostrar_bean_page.jsp`
 - `mostrar_bean_request.jsp`
 - ¿Cuál es la clase utilizada para los bean?
 - ¿Cómo se asigna el valor a la propiedad id? ¿Se muestra algo en la consola de Eclipse?
 - Recargue la página, pero antes prediga el valor de los id que deben aparecer.
 - ¿Cómo se consigue que la propiedad valor sólo sea asignada la primera vez?
 - ¿Cómo se importan los ficheros, con `<jsp:include` o con `<@include`? ¿Cómo afecta?
 - Ejemplo de ámbitos session y application:
 - `bean_scope_session_vs_application.jsp`
 - `mostrar_bean_application.jsp`
 - `mostrar_bean_session.jsp`
 - `cerrar_sesion.jsp`
 - Acceda la página y recargue. Observe los valores de id.
 - Antes de pulsar el botón cerrar sesión, prediga los nuevos valores de id.
 - ¿Qué ocurre al pulsar el botón cerrar sesión? En el apartado Sesiones se volverá a incidir en este tema.

5.9 Cookies

Las cookies son ficheros de texto almacenados en el ordenador del cliente, que guardan información enviada por el servidor. JSP soporta cookies HTTP de forma transparente.

Hay tres pasos involucrados en la identificación de los usuarios recurrentes:

- La página JSP del servidor envía un conjunto de cookies al navegador que contienen, por ejemplo, el nombre del usuario, la edad, un número identificativo, etc.
- El navegador almacena esta información en la máquina local para uso futuro.
- La próxima vez que el navegador envía una petición al servidor web, también envía las cookies, las cuales utiliza el servidor para identificar al usuario o para algún otro propósito.

A continuación, veremos el uso de cookies con JSP.

5.9.1 La estructura de una cookie

Las cookies, normalmente, se manejan mediante un conjunto de cabeceras HTTP (aunque JavaScript puede también crear una cookie directamente en un navegador). Una página JSP que envía una cookie puede enviar cabeceras que se parezcan a lo siguiente:

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday, 04-Apr-14 22:03:38 GMT;
            path=/; domain=fast.com
Connection: close
Content-Type: text/html
```

Como se puede apreciar, la cabecera “Set-Cookie” contiene un par nombre-valor, la fecha GMT, una ruta y un dominio. El nombre y el valor serán codificados como nombre=valor. El campo de expiración es una instrucción para el navegador con el fin de “olvidar” la cookie después de la hora y la fecha dada.

Si se configura el navegador para almacenar cookies, guardará esta información hasta la fecha de expiración. Si el usuario navega hacia una página que corresponda con la ruta y el dominio de la cookie, el navegador reenviará la cookie al servidor. Las cabeceras del navegador serán similares a lo siguiente:

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: fas01.us.es:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1, *, utf-8
Cookie: name=xyz
```

Una página JSP tendrá acceso a las cookies recibidas a través del método del objeto request `request.getCookies()`, que devuelve un array de objetos Cookie.

TAREAS

- a) Consulte la documentación de la interfaz del objeto request en: <http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletRequest.html> y compruebe que el método `getCookies()` devuelve un array de objetos `Cookie`.
- b) Haciendo click sobre el valor devuelto consulte la documentación de la clase `Cookie` y compruebe que el constructor tiene dos parámetros: nombre y valor.
- c) Compruebe que existen los métodos `getName()`, `getValue()` y `setMaxAge()`.

Una página JSP podrá añadir cookies a la respuesta a través del método del objeto `response` `response.addCookie()`.

TAREAS

- a) Consulte la documentación de la interfaz del objeto `response` en: <http://docs.oracle.com/javaee/7/api/javax/servlet/http/HttpServletResponse.html> y compruebe que el método `addCookie()` toma como parámetro un objeto `Cookie`.

5.9.2 Definición de cookies

La definición de cookies implica tres pasos:

- 1) Creación del objeto `Cookie`:

```
Cookie cookie = new Cookie("clave", "valor");
```

Ni el nombre ni el valor deberían contener espacios en blanco ni ninguno de estos caracteres: [] () = , " / ? @ : ;

- 2) Ajustar la edad máxima (en segundos) durante el cual la cookie es válida, mediante el método `setMaxAge`:

```
cookie.setMaxAge(60*60*24);
```

- 3) Enviar la cookie dentro de las cabeceras de respuesta HTTP, mediante el método `addCookie` del objeto `response`:

```
response.addCookie(cookie);
```

Observe los siguientes ficheros:

AppWeb/WebContent/jsp/ej_cookies_ini.html

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Formulario Cookies</title>
</head>

<body>
<form action="ej_cookies_main.jsp" method="GET">
```



```

Nombre: <input type="text" name="first_name">
<br/>
Apellidos: <input type="text" name="last_name" />
<input type="submit" value="Enviar"/>
</form>
</body>
</html>

```

AppWeb/WebContent/jsp/ej_cookies_main.jsp

```

<?xml version="1.0" encoding="UTF-8" ?>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>

<%
    // Crear cookies para nombre y apellidos.
    Cookie      firstName      =      new      Cookie("first_name",
request.getParameter("first_name"));
    Cookie      lastName       =      new      Cookie("last_name",
request.getParameter("last_name"));

    // Ajustasmo el tiempo de expiracion a 24h para ambas cookies:
    firstName.setMaxAge(60*60*24);
    lastName.setMaxAge(60*60*24);

    // Las anyadimos a la respuesta
    response.addCookie( firstName );
    response.addCookie( lastName );
%>

<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Creacion de Cookies</title>
</head>

<body>
<h1>Creacion de Cookies</h1>
<ul>
<li><p><b>Nombre:</b>
    <%= request.getParameter("first_name") %>
</p></li>
<li><p><b>Apellidos:</b>
    <%= request.getParameter("last_name") %>
</p></li>
</ul>
</body>
</html>

```

TAREAS

- Acceda a http://localhost:8080/AppWeb/jsp/ej_cookies_ini.html para probar el funcionamiento.
- Mediante las herramientas de desarrollador del navegador, vea las cabeceras en la respuesta HTTP y localice las relacionadas con las cookies.

A continuación, veremos cómo recuperar las cookies para emplearlas en la aplicación web.

5.9.3 Lectura de cookies

Para obtener las cookies, es necesario crear una tabla de objetos `javax.servlet.http.Cookie`, mediante la llamada al método `getCookies()` de `HttpServletRequest`. Entonces, es posible recorrer la tabla utilizando los métodos `getName()` y `getValue()` para acceder a cada cookie y a su valor asociado.

Observe el siguiente fichero:

AppWeb/WebContent/jsp/ej_cookies_lectura.jsp

```
<!DOCTYPE html>
<html>
<head>
<title>Lectura de Cookies</title>
</head>
<body>
<h1>Lectura de Cookies</h1>
<%
    Cookie cookie = null;
    Cookie[] cookies = null;

    // Obtenemos una array de Cookies asociadas con este dominio:
    cookies = request.getCookies();
    if( cookies != null ){
        out.println("<h2> Cookies encontradas</h2>");
        for (int i = 0; i < cookies.length; i++){
            cookie = cookies[i];
            out.print("Nombre : " + cookie.getName( ) + ", ");
            out.print("Valor: " + cookie.getValue( )+" <br/>");
        }
    }else{
        out.println("<h2>No se encontraron cookies</h2>");
    }
%>
</body>
</html>
```

TAREAS

- Acceda a http://localhost:8080/AppWeb/jsp/ej_cookies_lectura.jsp para probar el funcionamiento.

5.9.4 Eliminación de cookies

Eliminar cookies es muy simple, y se realiza a través de varios pasos:

- Ajustar la edad a 0 mediante el método `setMaxAge()` para eliminar la cookie existente.
- Añadir esta cookie de vuelta en la cabecera de respuesta.

Observe el siguiente ejemplo:

AppWeb/WebContent/jsp/ej_cookies_del.jsp

```
<!DOCTYPE html>
<html>
<head>
<title>Eliminación de Cookies</title>
</head>
<body>
<h1>Eliminación de Cookies</h1>

<%
    Cookie cookie = null;
    Cookie[] cookies = null;

    cookies = request.getCookies();
    if( cookies != null ){
        out.println("<h2> Cookies encontradas:</h2>");
        for (int i = 0; i < cookies.length; i++){
            cookie = cookies[i];

            //Eliminamos la cookie "first_name"
            if((cookie.getName( )).compareTo("first_name") == 0 ){
                cookie.setMaxAge(0);
                response.addCookie(cookie);
                out.print("Cookie eliminada: " +
                    cookie.getName( ) + "<br/>");
            }
            out.print("Nombre : " + cookie.getName( ) + ", ");
            out.print("Valor: " + cookie.getValue( )+" <br/>");
        }
    }else{
        out.println(
            "<h2>No se encontraron cookies</h2>");
    }
%>
</body>
</html>
```

TAREAS

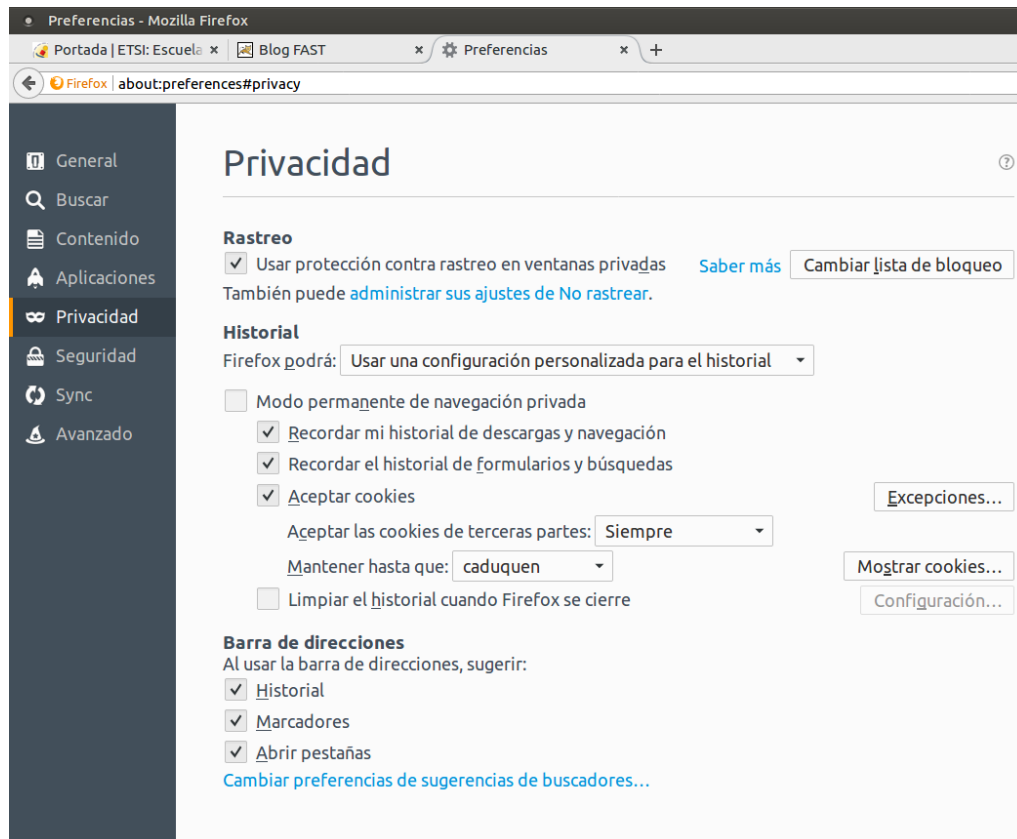
- Acceda a http://localhost:8080/AppWeb/jsp/ej_cookies_del.jsp para probar el funcionamiento.
- Recargue la página para comprobar que la cookie “first_name” se ha eliminado correctamente.

5.9.5 Ejemplos

Nota:

Si al reiniciar el navegador se siguen borrando las cookies que no han caducado, puede que tenga que cambiar la configuración de Firefox:

Menú > Preferencias > Privacidad > Desactivar Modo permanente de navegación privada > Aceptar cookies > Mantener hasta que caduquen



Puede que tenga que reiniciar Firefox.

a) Acceda a `AppWeb/WebContent/jsp/cookies/`.

b) En este directorio tiene dos ejemplos:

- **cookiebasico:**

Analice el contenido de los ficheros ubicados en
`AppWeb/WebContent/jsp/cookies/cookiebasico`.

Acceda a <http://localhost:8080/AppWeb/jsp/cookies/cookiebasico/listarcookies.jsp> y haga click en todos los enlaces mostrados para probar el funcionamiento.

- **blog:**

Analice el contenido de los ficheros ubicados en
`AppWeb/WebContent/jsp/cookies/blog`.

Acceda a <http://localhost:8080/AppWeb/jsp/cookies/blog/blog.jsp> y haga click en todos los enlaces mostrados para probar el funcionamiento.
Aquí se hace uso de las clases “EntradaBlog” y “GestorEntradasBlog” ubicadas en `AppWeb/src/fast/`

Acceda a varias entradas del blog y observe cómo se va rellenando el historial. Con F12 en el navegador observe la cookie que se envía y compruebe que contiene los índices de las entradas mostradas en el historial, separadas por comas. ¿Cómo se llama la cookie? El valor de la cookie ¿es una cadena o una tabla? ¿Cómo se pasa en el servidor de los valores separados por comas a una tabla y viceversa?

Acceda a la página desde otra pestaña del navegador (u otra ventana) ¿qué aparece en el historial? ¿se comparten las cookies? ¿Qué ocurre cuando se pulsa borrar historial?

Para acceder desde otro navegador (y no compartir las cookies) se va a usar el que tiene incorporado Eclipse. Para ello, sobre `blog.jsp` pulse:

botón derecho > Run as > Run on server

Se abrirá una pestaña y mostrará la página en el propio Eclipse (esto se puede hacer varias veces y tendrá varias pestañas) ¿qué aparece en el historial? ¿se comparten las cookies con Firefox? ¿y con las otras pestañas del navegador de Eclipse? Observe que como el historial se almacena en el cliente (en la cookie) puede haber varios clientes accediendo simultáneamente (varios navegadores) y cada uno tiene su historial. Nota: si se abre el navegador por defecto del sistema, Firefox, entonces acceda a la barra de menú de Eclipse y configure:

Window > Web Browser > Internal Web Browser

Cierre todas las ventanas del navegador y vuelva a acceder ¿Se han mantenido las cookies (se conserva el historial)? Modifique el código de `blog.jsp` y haga que la cookie caduque en 100000 segundos. Puede consultar:

<http://docs.oracle.com/javaee/7/api/javax/servlet/http/Cookie.html>

Nota: al cambiar alguna parte del código Java de las clases, recuerde que al hacerlo desde Eclipse, éste se encarga de compilar, copiar a Tomcat y reiniciarlo. Si no usa Eclipse necesitaría recompilar las clases con `javac` y hacer las operaciones manualmente.

5.10 Sesiones

5.10.1 Introducción

Se dice que varias peticiones pertenecen a una misma sesión si provienen del mismo cliente (al menos durante un periodo de tiempo).

El problema es que HTTP es un protocolo “sin estado”, que quiere decir que cada vez que un cliente realiza una petición a una web, el cliente puede abrir una conexión distinta al servidor web, y el servidor no guarda de forma automática ningún registro de peticiones previas del cliente.

No obstante, existen varios mecanismos (vistos en teoría: cookies, reescritura de URLs y campos ocultos en formularios) para mantener una sesión entre un cliente y un servidor (es decir, decidir si varias peticiones provienen del mismo cliente).

Desde el punto de vista de JSP el uso de estos mecanismos es transparente al programador, y lo que realmente permite trabajar con el concepto de sesión es el objeto implícito `session`.

5.10.2 El objeto session

Este objeto ofrece la interfaz `HttpSession`, que proporciona una manera de identificar a un usuario a través de las distintas peticiones al sitio web servidor, y permite almacenar información sobre dicho usuario.

Por defecto, JSP habilita el rastreo de sesiones, e instancia un nuevo objeto `HttpSession` para cada nuevo cliente de forma automática. Se puede deshabilitar este comportamiento de forma explícita mediante la directiva `page`:

```
<%@ page session="false" %>
```

En el apartado “Objetos implícitos” ya se vio que el objeto `session` tiene los métodos para gestionar atributos: `setAttribute()`, `getAttribute()`, `getAttributeNames()`, `removeAttribute()`. Cuando desde una página JSP se establece el valor de un atributo usando el objeto `session`, ese atributo será accesible desde otra página que use el objeto `session` y sea accedida por el mismo usuario (se dice que pertenece a la misma sesión).

5.10.3 Seguimiento de sesión

Este ejemplo describe cómo utilizar el objeto `session` para averiguar el instante de creación y el momento de último acceso para una sesión. Se asocia una nueva sesión con la petición si no existe ya una.

AppWeb/WebContent/jsp/ej_session.jsp

```

<%@ page import="java.io.*,java.util.*" %>
<%
    // Obtenemos el instante de creacion
    Date createTime = new Date(session.getCreationTime());

    // Obtenemos el instante de ultimo acceso a la pagina
    Date lastAccessTime = new Date(session.getLastAccessedTime());

    String title = "Bienvenido a mi website";
    Integer visitCount = new Integer(0);
    String visitCountKey = new String("visitCount");
    String userIDKey = new String("userID");
    String userID = new String("ABCD");

    // Comprobamos si el visitante es nuevo en la web
    if (session.isNew() || session.getAttribute(visitCountKey)==null ){
        title = "Bienvenido a una NUEVA session a mi website ";
        session.setAttribute(userIDKey, userID);
        session.setAttribute(visitCountKey, visitCount);
    }
    visitCount = (Integer)session.getAttribute(visitCountKey);
    visitCount = visitCount + 1;
    userID = (String)session.getAttribute(userIDKey);
    session.setAttribute(visitCountKey, visitCount);
%>

<html>
<head>
<title><%= title %></title>
</head>

<body>

<h1>Seguimiento de sesión <%= title %></h1>
<table border="1">
<tr bgcolor="#949494">
    <th>Info de sesion</th>
    <th>Valor</th>
</tr>
<tr>
    <td>id</td>
    <td><% out.print( session.getId()); %></td>
</tr>
<tr>
    <td>Instante de creacion</td>
    <td><% out.print(createTime); %></td>
</tr>
<tr>
    <td>Instante de ultimo acceso</td>
    <td><% out.print(lastAccessTime); %></td>
</tr>
<tr>
    <td>ID de usuario</td>
    <td><% out.print(userID); %></td>
</tr>
<tr>
    <td>Numero de visitas</td>
    <td><% out.print(visitCount); %></td>
</tr>
</table>

</body>
</html>

```

TAREAS

- Abra en un navegador la URL http://localhost:8080/AppWeb/jsp/ej_session.jsp y compruebe el funcionamiento recargando varias veces la página.
- Edite el fichero ej_session.jsp con alguna modificación y grábela. Vuelva a recargar la página. Compruebe que no cambian los valores asociados a la sesión.

5.10.4 Eliminación de sesión

Cuando se obtienen los datos de la sesión del usuario, se pueden realizar varias acciones:

- Eliminar un atributo, mediante el método `removeAttribute`.
- Eliminar la sesión completa, mediante el método `invalidate`.
- Ajustar el timeout de la sesión, mediante `setMaxInactiveInterval`.
- Configurar el DD `web.xml`, para configurar el timeout de sesión. Ejemplo:

```
<session-config>
  <session-timeout>15</session-timeout>
</session-config>
```

El timeout se expresa en minutos, y sobrescribe el valor por defecto de 30 minutos en Tomcat.

El método `getMaxInactiveInterval` devuelve un periodo de timeout para la sesión en segundos. Si la sesión se configure en el DD `web.xml` a 15 minutos, este método devuelve un valor de 900.

Observe el siguiente ejemplo:

AppWeb/WebContent/jsp/ej_session2.jsp

```
<%@ page import="java.io.*,java.util.*" %>

<%@ page session="true" %>

<%
java.util.ArrayList<String> accesos=
    (java.util.ArrayList<String>) session.getAttribute("accesos");
if (accesos==null)
    accesos=new java.util.ArrayList<String>();

accesos.add(new java.util.Date().toString());
session.setAttribute("accesos", accesos);

if (request.getParameter("invalidaSesion")!=null)
    session.invalidate();
%>

<html>
<body>
<form>
    <input type="submit" name="invalidaSesion" value="Invalidar
sesion"/>
    <input type="submit" value="Recargar pagina"/>
</form>
```



```

<br/>
Usted accedio a esta pagina en los siguientes momentos: <br />
<%
    for (int i=0;i<accesos.size();i++)
    {
    %>
    <%= accesos.get(i) %>
    <br />
    <%
    }
    %>
</body>
</html>

```

TAREAS

- Abra en un navegador la URL http://localhost:8080/AppWeb/jsp/ej_session2.jsp y compruebe el funcionamiento.
- Recargue la página varias veces, entonces, invalide la sesión y vuelva a recargar la página.

5.10.5 Ejemplos

TAREAS

- Acceda al directorio
AppWeb/WebContent/jsp/sesiones
y observe las páginas JSP del subdirectorio session.
- Abra en un navegador la URL <http://localhost:8080/AppWeb/jsp/sesiones/index.jsp> y compruebe el funcionamiento de cada uno de los ejemplos.

El ejemplo de acceso con usuario y clave hace uso de los siguientes ficheros Java:

```

AppWeb/src/fast/Usuario.java
AppWeb/src/fast/GestorUsuarios.java

```

- En el uso básico: ¿Cómo se consigue mostrar todos los atributos de sesión?
- En el ejemplo de usuario y clave:
 - ¿Dónde están almacenados los usuarios permitidos con sus claves? ¿Cómo se comprueban?
 - En inicio.jsp cuando se intenta acceder a las páginas soloAutorizados.jsp y soloAutorizados2.jsp sin haberse autenticado (pulse el símbolo + para que aparezcan los enlaces), ¿cuál es la diferencia entre las dos?
 - En las todas las páginas de acceso restringido se comprueba al principio que el usuario se ha autenticado. ¿Se podría quitar esa comprobación usando filtros? Hágalo.

Nota: al cambiar alguna parte del código Java de las clases, recuerde que, al hacerlo desde Eclipse, éste se encarga de compilar, copiar a Tomcat y reiniciarlo. Si no usa Eclipse necesitaría recompilar las clases con `javac` y hacer las operaciones manualmente.

5.11 Expression Language (EL)

Expression Language (EL), permite el acceso a parámetros y atributos fácilmente para visualizarlos.

Como ya se ha visto, en la mayoría de ocasiones, se emplea JSP para la parte de visualización, y toda la parte de lógica de la aplicación web se presenta en código de servlets. Cuando se recibe la petición de un cliente en un servlet, se procesa y se añaden los atributos en los posibles ámbitos de:

- Aplicación
- Sesión
- Petición

para ser recuperados por el código de la página JSP. También se utilizan parámetros de la petición, cabeceras, cookies y parámetros de inicio con JSP para crear las repuestas.

En JSP también se pueden usar atributos en el ámbito de:

- Página

para cuando la información no interesa que sea accedida por todas las páginas (ámbito de Aplicación) ni mantenerla para una próxima petición del mismo cliente (ámbito de Sesión) ni accederla desde otra página que se incluya o reenvíe (ámbito de Petición).

Se pueden emplear scriptlets y expresiones JSP para recuperar atributos y parámetros en JSP mediante código Java y utilizarlo con propósitos de visualización. Sin embargo, para diseñadores web, el código Java es complicado.

Por todo lo anterior, desde JSP 2.0 se introdujo **Expression Language (EL)**, a través del cual es posible recuperar parámetros y atributos fácilmente.

5.11.1 Objetos implícitos de EL

JSP EL proporciona muchos objetos implícitos, que son distintos a los objetos implícitos de JSP ya vistos. El único objeto implícito común de una página JSP y EL es el objeto `pageContext`.

A continuación, una tabla con los objetos implícitos de EL:

Objetos implícitos EL	Descripción
<code>pageScope</code>	Mapa que contiene los atributos del ámbito de página.
<code>requestScope</code>	Mapa que contiene los atributos del ámbito de petición.
<code>sessionScope</code>	Mapa que contiene los atributos del ámbito de sesión.

applicationScope	Mapa que contiene los atributos del ámbito de aplicación.
param	Mapa utilizado para obtener el valor del parámetro de request, devuelve un solo valor.
paramValues	Mapa utilizado para obtener los valores de los parámetros de request en un array, útil cuando el parámetro de request contiene múltiples valores.
header	Mapa utilizado para obtener la información de la cabecera HTTP request.
headerValues	Mapa utilizado para obtener los valores de la cabecera HTTP request en un array, útil cuando contiene múltiples valores.
cookie	Mapa utilizado para obtener el valor de una cookie en JSP.
initParam	Mapa utilizado para obtener los parámetros de inicio del context (es decir, de la aplicación , y podría haberse llamado contextParam, pero desafortunadamente no fue así). No se puede emplear para los parámetros de inicio del servlet.

Estos objetos implícitos de EL son distintos de los objetos implícitos de JSP, y solo se pueden emplear con EL.

Además, se puede usar pageContext:

Objeto implícito EL que coincide con JSP	Descripción
pageContext	El mismo objeto que el objeto implícito de JSP pageContext, utilizado para obtener las referencias a request, session, etc.

Observe que todos los objetos implícitos son mapas (java.util.Map) a excepción de pageContext (que es un javaBean).

5.11.2 Sintaxis

La sintaxis de EL es `${expression}`, y se pueden usar objetos implícitos y operadores de EL para recuperar atributos desde los diferentes ámbitos (scopes) y utilizarlos en las páginas JSP.

Dentro de la expresión, lo primero (o lo único si no hay operadores) que debe aparecer es uno de los **objetos implícitos** (que son mapas excepto pageContext que es un JavaBean) o un **atributo** (almacenado previamente en alguno de los ámbitos).

El operador más utilizado es el ‘.’ (aunque el operador ‘[]’ es más potente como se verá más adelante) y lo que hay después del ‘.’ debe ser una clave del mapa o una propiedad del atributo (que en este caso debe ser un JavaBean).

Un ejemplo donde lo primero que aparece es el objeto implícito param (que es un mapa) y que mostraría los valores de los campos usu y contra de una petición de un formulario sería:

```
<li>El usuario es: ${param.usu}</li>
<li>La contraseña es: ${param.contra}</li>
```

Si queremos recuperar la propiedad nombre de un JavaBean llamado cliente (el JavaBean cliente será un atributo almacenado en alguno de los ámbitos):

```
<p>El nombre del cliente es: ${cliente.nombre}<p/>
```

El ejemplo anterior busca el atributo JavaBean cliente en los distintos ámbitos en orden de menor a mayor alcance: página, petición, sesión y aplicación.

Si tuviéramos varios atributos JavaBean llamados cliente en distintos ámbitos, y queremos acceder a uno en concreto, por ejemplo, al del ámbito de sesión:

```
<p>El nombre del cliente es: ${sessionScope.cliente.nombre}<p/>
```

Si el atributo no es un JavaBean, entonces puede ser una lista (java.util.list) o un array (tabla declarada con []), y para acceder a los elementos hay que usar el operador [] (no se puede usar el operador ‘.’). Si tuviéramos un atributo que fuera una tabla de clientes (el atributo se llama tabclie):

```
<p>Nombre primer cliente: ${tabclie[0].cliente.nombre}<p/>
<p>Nombre segundo cliente: ${tabclie[1].cliente.nombre}<p/>
<p>Nombre tercer cliente: ${tabclie[2].cliente.nombre}<p/>
```

El operador [] siempre se puede usar en lugar del operador ‘.’, pero es menos cómodo. Por ejemplo, los valores de los campos de un formulario vistos anteriormente serían (hay que poner comillas para que no se intente evaluar):

```
<li>El usuario es: ${param["usu"]}</li>
<li>La contraseña es: ${param["contra"]}</li>
```

Otro caso en que no se puede utilizar el operador ‘.’ es cuando el nombre que debe aparecer a la derecha del punto contiene a su vez un punto. Por ejemplo, si se quiere acceder a un atributo en el ámbito de aplicación y ese atributo se llama “foo.bar”:

```
${applicationScope["foo.bar"]}
```

Otro uso muy interesante de las EL es poder asignar a una propiedad de un bean un valor que no sea un String o un tipo básico. Si tiene por ejemplo un bean llamado grupo que tiene una propiedad llamada

alumnos que es una lista. y si tiene una variable de tipo lista que quiere asignar a la propiedad, puede crear un atributo llamado `listaAlum` cuyo valor sea esa variable de tipo lista, y asignarlo a la propiedad mediante:

```
<jsp:setProperty property="alumnos" name="grupo" value="${listaAlum}"/>
```

5.11.3 El objeto `pageContext`

El objeto `pageContext` tiene su utilidad basada en el hecho de que los objetos implícitos de EL terminados en `Scope` permiten acceder a los **atributos** en el ámbito correspondiente pero NO permiten acceder a métodos de los objetos. Por ejemplo, intentar utilizar `requestScope.queryString` (intentando emular a `request.getQueryString()` de una expresión JSP) dentro de una EL no funcionaría.

Para acceder a los métodos del objeto `request` de JSP (no confundir con el objeto `requestScope` de EL) es posible utilizar el objeto `pageContext` de EL (que sí coincide con el objeto `pageContext` de JSP) para (mediante el operador `'.'`) obtener la referencia al objeto `request` (como si se llamara a `getRequest()`), y volver a utilizar el operador `'.'` para acceder a la cadena de petición (como si se llamara a `getQueryString()`):

```
${pageContext.request.queryString}
```

Otro ejemplo, para obtener el nombre del método HTTP de la petición (como si se llamara a `getMethod()`):

```
${pageContext.request.method}
```

Lo mismo habría que hacer con los otros objetos de JSP, por ejemplo, para el identificador de sesión:

```
${pageContext.session.id}
```

Para el objeto `application` es distinto, ya que el método para obtenerlo, a diferencia de `getRequest()` y de `getSession()`, se llama `getServletContext()` (podría haberse llamado `getApplication()`..., pero desafortunadamente no es así):

```
${pageContext.servletContext.contextPath}
```

5.11.4 Operadores JSP EL

Aunque ya se han visto los operadores `"."` y `[]`, se añaden aquí por completitud.

Operador `[]`

Se emplea para obtener datos de una Lista o array (usando valores numéricos), o propiedades usando una cadena de caracteres como nombre de la propiedad. Ejemplos:

- `${myList[1]}` y `${myList["1"]}`

Son lo mismo, se puede emplear tanto un índice de Lista o Array como un literal de tipo String

- `${myMap[expr]}`

Si el parámetro dentro de “[]” no es un String, se evalúa como EL.

- `${myMap[myList[1]]}`

El operador [] se puede anidar.

- `${requestScope["foo.bar"]}`

El operador [] es de gran utilidad cuando los nombres de los atributos contienen puntos, ya que entonces no se puede emplear el operador ‘.’, pero sí el operador [].

Operador ‘.’

Se utiliza para obtener valores de atributos para el caso particular en el que el identificador no contenga el carácter punto ni sea un número (en esos casos habría que utilizar []):

```
${firstObj.secondObj}
```

firstObj puede ser un objeto implícito EL o un atributo del ámbito de página, petición, sesión o aplicación. Ejemplo donde se usa el objeto implícito requestScope para buscar un atributo de nombre empleado:

```
${requestScope.empleado.direccion}
```

Operadores aritméticos

Se emplean para cálculos simples en expresiones EL: +, -, *, / ó div, % ó mod.

Operadores lógicos

Son estos: && (and), || (or) y ! (not).

Operadores relacionales

Son estos: == (eq), != (ne), < (lt), > (gt), <= (le) y >= (ge).

Precedencia de operadores EL

Las expresiones EL se evalúan de izquierda a derecha, similar a Java.

5.11.5 Palabras reservadas de JSP EL

Las siguientes palabras no se pueden emplear para identificadores de variables JSP:

and	or	not	eq	ne
lt	gt	le	ge	true
false	null	instanceof	empty	div, mod

5.11.6 JSP EL: puntos importantes

- a) Las expresiones EL siempre se encuentran entre "\${" y "}".
- b) Se pueden deshabilitar las expresiones EL mediante la directiva `page` de JSP, con el atributo `isELIgnored` ajustado a `true`.
- c) JSP EL se puede emplear para obtener atributos, cabeceras, cookies, parámetros de inicio, etc, pero no para establecer valores de manera directa (de manera indirecta puede existir código en las funciones que se ejecutan que sí establezcan valores).
- d) Los objetos implícitos de JSP EL son distintos a los de JSP, excepto `pageContext`.
- e) El objeto `pageContext` se utiliza para obtener propiedades adicionales de `request`, `response`, etc., por ejemplo, para obtener el método HTTP `request` de una petición:

```
${pageContext.request.method}
```
- f) Si JSP EL no encuentra un atributo o una expresión devuelve `null`, no arroja una excepción. Trata `null` como `0` para expresiones aritméticas y como `false` para expresiones lógicas.
- g) El operador `[]` es más potente que el operador `.` porque además de poder acceder a lo mismo que ese operador (a mapas y a JavaBeans), puede acceder a una lista (`List`) o tabla (`array`), y puede ser anidado, incluso cuando el argumento no es un literal de tipo `string`.
- h) Las expresiones EL se podrían usar dentro de código Java usando funciones de librería.
- i) Se pueden emplear las funciones EL para llamar a métodos de una clase Java.

5.11.7 Ejemplos

A continuación, puede ejecutar algunos ejemplos, estudiarlos y modificarlos.

- a) Acceda al directorio
AppWeb/WebContent/jsp/expressionlang
El fichero index.jsp contiene un enlace a cada ejemplo presentado. Acceda a <http://localhost:8080/AppWeb/jsp/expressionlang/index.jsp> para visualizar los ejemplos.

Los distintos ejemplos hacen uso de los siguientes ficheros Java:

```
AppWeb/src/fast/Direccion.java
AppWeb/src/fast/Persona.java
AppWeb/src/fast/Empresa.java
```

- b) Céntrese en observar el contenido de las distintas páginas JSP ubicadas en:

AppWeb/WebContent/jsp/expressionlang/el:

- Ejemplo de acceso a datos: el_acceso.jsp
 - ¿Por qué en la parte de tablas asociativas \${colores["verde"]} aparece en verde? Añada un nuevo elemento a la tabla asociativa que represente el color gris (#777) y haga que \${colores.gris} aparezca en gris.
 - ¿Por qué desaparece el mensaje “Debe recargar la página, haciendo click en este enlace” cuando se hace click en el enlace?.
 - Acceda a esa página cambiando los parámetros ?paramA=azul¶mB=1 por ?paramA=gris¶mB=1 y observe dónde cambia el resultado
 - Cree un nuevo bean llamado holding de tipo Holding relleno con una dirección y varias empresas, en el ámbito request. Las propiedades de ese bean serían:


```
private String nombre;
private Direccion direccion;
private List<Empresa> empresas;
```

Añada una EL para acceder al nombre de un empleado de una empresa del holding. Añada otras EL para acceder otros valores (use índices para acceder a las distintas empresas).
 - En el apartado anterior habrá creado varias variables del tipo Empresa y las habrá añadido a la propiedad empresas mediante código java en un scriptlet. Modifique ese código para crear una variable tipo lista que contenga las empresas y asígnela a la propiedad empresa mediante una etiqueta de acción: <jsp:setProperty>. Para ello tendrá que crearse un atributo cuyo valor sea la variable de tipo lista y usar una EL.
- Ejemplo de uso de operadores de EL: el_operadores.jsp
 - Estudie el uso de los operadores, por ejemplo ¿por qué cuando se asigna al bean pepito “\${param.paramA}\${noExiste} negro” aparece NEGRO en mayúsculas?
 - Cree una variable booleana y añada una EL de tal forma que valga “claseA” si la variable es cierta o “claseB” si la variable es falsa. Use el operador ?:. Tenga en cuenta que desde la EL puede acceder a los atributos, pero no a directamente las variables ¿en qué ámbito va a crear el atributo cuyo valor sea la variable? ¿debe llamarse igual?

- c) Modifique el fichero:

AppWeb/WebContent/jsp/expressionlang/el/el_acceso.jsp

y añada unas líneas que contengan EL para que a través de `pageContext` acceda a los métodos:

- del objeto `request` de JSP
 - `getQueryString`
 - `getMethod`
- del objeto `session` de JSP
 - `getId`
- del objeto `application` de JSP
 - `getContextPath`

Nota: al cambiar alguna parte del código Java de las clases, recuerde que, al hacerlo desde Eclipse, éste se encarga de compilar, copiar a Tomcat y reiniciarlo. Si no usa Eclipse necesitaría recompilar las clases con `javac` y hacer las operaciones.

6 Ejemplos de aplicaciones

6.1 formQuiz

Como ejemplo de aplicación, se propone estudiar una aplicación web denominada `formQuiz`.

`formQuiz` es en una aplicación web que permite generar un formulario en HTML sin necesidad de conocer dicho lenguaje. Esto es posible gracias a un formulario JSP de creación de elementos que nos permite seleccionar un tipo de elemento (de formulario HTML), un nombre y, en caso de que sea un elemento que incluya opciones, nos permite definir el número de opciones que deseamos generar. La aplicación proporciona al usuario un resultado doble:

- El código fuente de los elementos generados.
- La visualización de los elementos de formulario generados.

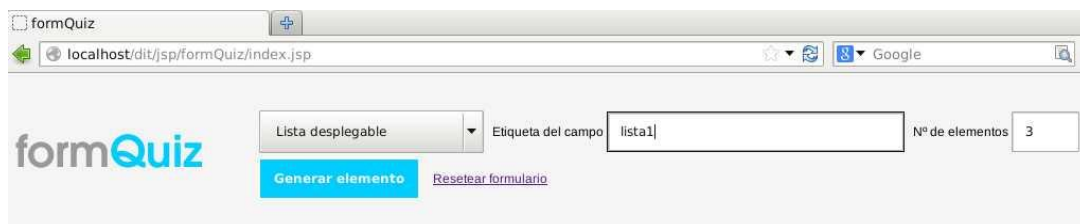
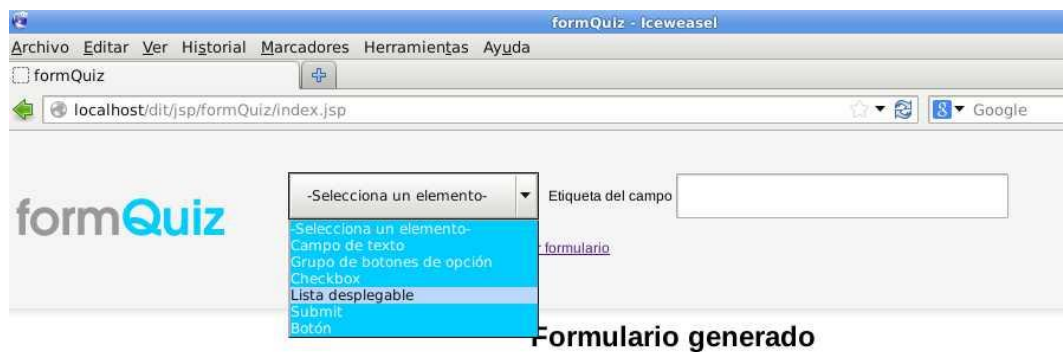
Observe las siguientes capturas, que son el resultado de la ejecución de la aplicación web finalizada.

- Página principal, accesible desde:

<http://localhost:8080/AppWeb/jsp/formQuiz/index.jsp>:



- Añadimos un nuevo campo:



- Resultado:

Formulario generado

Código fuente

```
<!-- formulario en HTML -->
<ul>
<li>
<label for='fq0'>lista1: </label>
<select name='fq0' id='fq0' >
<option value='fq0-0'>Opción0</option>
<option value='fq0-1'>Opción1</option>
<option value='fq0-2'>Opción2</option>
</select>
</li>
</ul>
```

Vista previa

lista1:

- Añadimos otro campo:

- Resultado:

Formulario generado

Código fuente

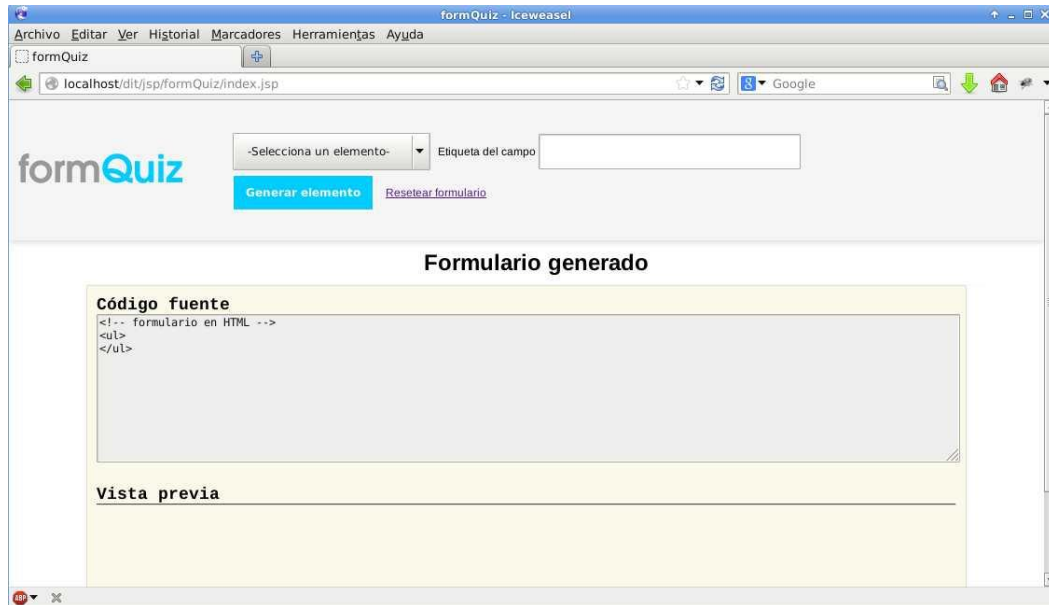
```
<li>
<label>grupob1: </label>
<span>
<input type='radio' name='fq1' id='fq1-0' value='fq1-0' />
<label for='fq1-0'>Opción0</label>
</span>
<span>
<input type='radio' name='fq1' id='fq1-1' value='fq1-1' />
<label for='fq1-1'>Opción1</label>
</span>
</li>
```

Vista previa

lista1:

grupob1: ☐ Opción0 ☐ Opción1

- Reset del formulario:



A continuación, se describen los ficheros de clases Java que se emplean como JavaBeans en la página JSP, así como otros ficheros de apoyo:

- AppWeb/src/fast/DatosInput.java: clase que genera un nuevo campo de formulario.
- AppWeb/src/fast/DatosForm.java: clase que añade un nuevo campo al formulario existente como un nuevo elemento ArrayList, apoyándose en la clase DatosInput.java.
- AppWeb/WebContent/jsp/formQuiz/images/logo_formquiz.png: logo de la aplicación formQuiz.
- AppWeb/WebContent/jsp/formQuiz/style.css: hoja de estilos.

Las clases Java mencionadas se emplean para obtener los nuevos campos y añadirlos al formulario.

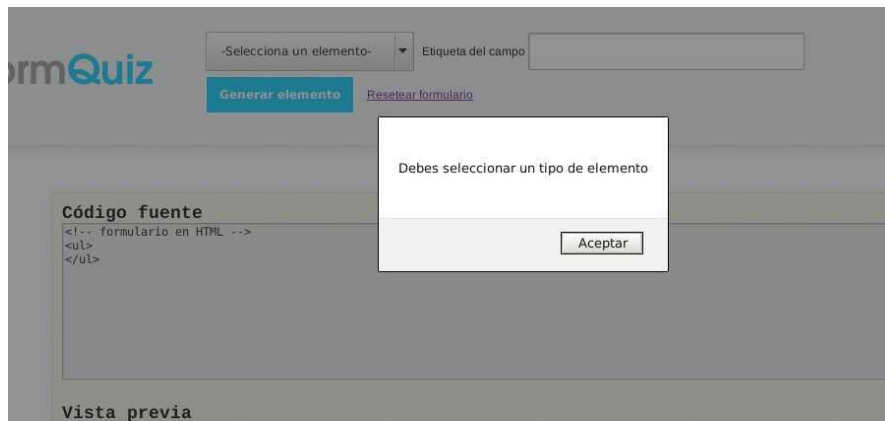
A continuación, se describen los ficheros jsp y js:

- AppWeb/WebContent/jsp/formQuiz/index.jsp: es la página principal (y única que se le presenta al usuario) y hace uso de las clases mencionadas para procesar los nuevos campos.
- AppWeb/WebContent/jsp/formQuiz/reset.jsp: hay un enlace a esta página en index.jsp ("Resetar formulario"). Su objetivo es invalidar la sesión (receta el formulario) y redirige mediante el objeto response a la página de inicio (index.jsp). El código que hace esto es:

```
session.invalidate();  
response.sendRedirect("index.jsp");
```

Compruebe que `sendRedirect` es distinto a `forward`, ya que `forward` es un reenvío interno dentro del servidor y en cambio `sendRedirect` envía una respuesta al navegador para que éste vuelva a enviar una petición. Analice el mensaje de respuesta (con F12 en el navegador) y averigüe qué hace que el navegador vuelva a enviar la petición.

- `AppWeb/WebContent/jsp/formQuiz/formQuiz.js`: fichero JavaScript encargado de mostrar el campo “Nº de elementos” cuando se elige la opción 2 (“Grupo de botones de opción”) o 4 (“Lista desplegable”).



También se comprueba lo siguiente al enviar el formulario:

- Si se ha seleccionado un tipo de elemento
- Si el número de elementos es un número comprendido entre 1 y 5 para las opciones 2 y 4.

6.2 formQuiz y Ajax

`formQuizAjax` es una modificación a la aplicación `formQuiz`, a través del uso de Ajax para mostrar los nuevos elementos de formulario añadidos.

Se emplean los mismos JavaBeans.

Se reutiliza el código de `index.jsp`, aunque con algunas modificaciones: en lugar de mostrar el formulario resultante a través de peticiones al JavaBean correspondiente, se recupera la nueva información mediante peticiones Ajax a la página `generar.jsp`, la cual realiza la petición de información al JavaBean correspondiente que antes se ejecutaba en `index.jsp`.

En resumen, estos son los ficheros más importantes modificados:

- `AppWeb/WebContent/jsp/formQuizAjax/index.jsp`: `index.jsp` de `formQuiz` modificado.
- `AppWeb/WebContent/jsp/formQuizAjax/formQuizAjax.js`: modificaciones a `formQuiz.js` para añadir la función de peticiones Ajax a la página `generar.jsp`.

- AppWeb/WebContent/jsp/formQuizAjax/generar.jsp: utiliza los JavaBeans para añadir el nuevo elemento de formulario y responder con el formulario resultante.

Las capturas de ejecución son idénticas a las de la aplicación formQuiz original.

6.3 Aplicación “Comentarios”

Se presenta la aplicación “Comentarios” como código de ejemplo, para su estudio y modificación.

La aplicación utiliza el objeto implícito `application` para ir guardando una serie de mensajes. Los clientes usan Ajax para mostrar instantáneamente los mensajes nuevos que se van añadiendo, utilizando sondeo cada 1 segundo.

TAREAS

- Acceda al directorio AppWeb/WebContent/jsp/appcomentarios y observe las páginas JSP contenidas en el mismo.
- Abra en un navegador la URL:
<http://localhost:8080/AppWeb/jsp/appcomentarios/comentarios.html>
y compruebe el funcionamiento.
El ejemplo hace uso de los siguientes ficheros Java:

AppWeb/src/fast/AlmacenComentarios.java
AppWeb/src/fast/Comentario.java
- Modifique la aplicación para que el sondeo sea cada 2 segundos.
Nota: al cambiar alguna parte del código Java de las clases, recuerde que al hacerlo desde Eclipse, éste se encarga de compilar, copiar a Tomcat y reiniciarlo. Si no usa Eclipse necesitaría recompilar las clases con `javac` y hacer las operaciones manualmente.

7 Anexo no evaluable

7.1 Ejecución de comandos externos

JSP, al permitir la ejecución de código Java, también permite la ejecución de comandos externos, mediante los objetos `Process` e `InputStream`.

TAREAS

- Acceda al directorio AppWeb/WebContent/jsp/ejecutarComando y observe el contenido del fichero `ejecutarComando.jsp`.
- Acceda a <http://127.0.0.1:8080/AppWeb/jsp/ejecutarComando/ejecutarComando.jsp> para probar el funcionamiento. Pruebe a ejecutar estos comandos:
 - `ls -l /`
 - `pwd`
 - `ping -c 5 8.8.8.8`
 - `tracert www.google.es`
 - Otros comandos que considere oportunos.
 En el caso de la ejecución del comando `ping`, observe cómo van apareciendo los resultados según se van generando.