PROGRAMACIÓN DIRIGIDA A OBJETOS

SEMINARIO

JSP (Java Server Pages)

Eduardo Haro Amate
Lydia López Ruiz

INDICE

1 DIFERENTES TECNOLOGÍAS PARA LA CREACIÓN DE PÁGINAS DINÁMICAS DE SERVIDOR

2 INTRODUCCIÓN A JSP

3 FORMATO DE UNA PÁGINA JSP

4 SERVIDORES Y CONTENEDORES DE SERVLETS/JSPS

5 INSTALANDO UN CONTENEDOR DE SERVLETS

6 NUESTRO PRIMER SERVLET

7 NUESTRA PRIMERA PÁGINA JSP

APÉNDICE: CONCEPTOS DE INTERÉS

BIBLIOGRAFIA

1 DIFERENTES TECNOLOGÍAS PARA LA CREACIÓN DE PÁGINAS DINÁMICAS DE SERVIDOR

Todas las tecnologías de creación dinámica de páginas Web en servidor persiguen el mismo fin, crear páginas Web utilizando un lenguaje de programación de forma que una vez solicitadas al servidor, son interpretadas devolviendo al usuario la página generada. Este tipo de tecnología permite tanto facilitar el mantenimiento de un sitio web (debido al grado de automatización del manejo de datos) así como incrementar la interactividad con dicho sitio web del usuario.
Si bien hay más tecnologías, a continuación mencionaremos las más utilizadas en nuestros días.

CGi (Common Gateway Interface)
Plataforma .Net (Páginas ASP)
Páginas PHP (Hypertext Preprocessor)
Páginas JSP (Java Server Pages)

2 INTRODUCCIÓN A JSP

Java Server Pages (**JSP**) es la tecnología para generar páginas web de forma dinámica en el servidor, desarrollado por Sun Microsystems, basado en scripts que utilizan una variante del lenguaje java.

Sin embargo JSP no se puede considerar un script al 100% ya que antes de ejecutarse el servidor web compila el script y genera un servlet, por lo tanto se puede decir que aunque este proceso sea transparente para el programador no deja de ser una aplicacion compilada.

La tecnología JSP, o de JavaServer Pages, es una tecnología Java que permite a los programadores generar dinámicamente HTML, XML o algún otro tipo de página web. Esta tecnología permite al código Java y a algunas acciones predefinidas ser embebidas en el contenido estático. En las JSP, se escribe el texto que va a ser devuelto en la salida (normalmente código HTML) incluyendo código java dentro de él para poder modificar o generar contenido dinámicamente. El código java se incluye dentro de las marcas de etiqueta <% y %>.

En una posterior especificación, se incluyeron taglib; esto es, la posibilidad de definir etiquetas nuevas que ejecuten código de clases java. La asociación de las etiquetas con las clases java se declaran en archivos de configuración en XML.

La clave de la programación en JSP, como ya hemos dicho, esta en Java , lenguaje que ha permitido que a partir de él se hayan ido desarrollando distintas tecnologías. Así, la especificación de JSP proviene de unos componentes ya desarrollados en Java denominados Java Servlets. Ambas especificaciones forman parte del entorno J2EE (Java 2 Entreprise Environment) utilizado en desarrollo de tipo empresarial. JSP por tanto, si bien puede llegar a ser más complejo que otro tipo de tecnologías (PHP por ejemplo), proporciona muchísima más potencia a la hora de desarrollar componentes de lógica de negocio (ERP, Enterprise Resource Planning), así como un mejor mantenimiento y posibles ampliaciones en sus componentes.

Las principales ventajas serían, por tanto:

-La principal ventaja de JSP frente a otros lenguajes es que permite integrarse con clases Java (.class) lo que permite separar en niveles las aplicaciones web, almacenando en clases java las partes que consumen más recursos así como las que requieren más seguridad, y dejando la parte encargada de formatear el documento html en el archivo jsp.

- Si microsoft está detrás de .NET, Sun está detras de JSP (Además de otras muchas compañías del sector)

- Se puede usar la tecnología de forma gratuita. Conforme se vayan necesitando otros componentes del J2EE, como los EJB, o servidores potentes como Weblogic, habrá que pagar las licencias, pero llegar a este punto significará estar desarrollando componentes de lógica de negocio para empresas

- Java se caracteriza por ser un lenguaje que puede ejecutarse en cualquier sistema, lo que sumado a jsp le da mucha versatilidad.

- Proporciona mucha más potencia, así como un mejor mantenimiento, a la hora de desarrollar componentes de lógica de negocio .

Desventajas

- La principal desventaja puede ser una cierta complejidad inicial en el aprendizaje y en el manejo de los conceptos relacionados con los entornos Java, J2EE y J2SE

3 FORMATO DE UNA PÁGINA JSP

Un fichero JSP consta de las siguientes secciones en este orden:

- ?? Comentarios Iniciales
- ?? Directiva(s) JSP page
- ?? Directiva(s) tag library opcionales
- ?? Declaración(es) JSP opcionales
- ?? Código HTML y JSP

Un fichero JSP empieza con un comentario del lado del servidor:

```
<%--
  - Author(s):
  - Date:
  - Copyright Notice:
  - @(#)
  - Description:
  --%>
```

Este comentario sólo es visible en el lado del servidor porque se elimina durante la traducción JSP. Dentro de este comentario están los autores, la fecha, la nota de copyright de la revisión, un identificador y una descripción sobre el JSP para los desarrolladores web. La combinación de caracteres "@(#) " es reconocida por ciertos programas para indicar el inicio de un identificador. Aunque dichos programas se utilizan muy raramente, el uso de estos strings no hace nada. Además, esta combinación algunas veces se le añade "Id" para que la información de identificación se inserte automáticamente en el JSP en algunas versiones de programas de control. La parte Description proporciona información concisa sobre los propósitos del JSP. No debe ser mayor de un párrafo.
En algunas situaciones, se necesita retener los comentarios de inicio y propagarlos al lado del cliente (visibles para los navegadores) para propósitos legales y de autenticidad. Esto se puede conseguir dividiendo el bloque de comentarios en dos partes, primero el comentario del lado del cliente:

```
<!--
  - Author(s):
  - Date:
  - Copyright Notice:
  -->
```

y luego un breve comentario del lado del servidor:

```
<%--
  - @(#)
  - Description:
  --%>
```

Directiva(s) JSP Page

Una directiva page define atributos asociados con la página JSP en tiempo de traducción. La especificación JSP no impone ninguna obligación sobre cuántas directivas page se pueden definir en la misma página. Por eso los dos siguientes fragmentos de código son equivalentes (excepto en que el primero de ello introduce dos líneas en blanco extras en la salida):

```
<%@ page session="false" %>
<%@ page import="java.util.*" %>
<%@ page errorPage="/common/errorPage.jsp" %>
```

Si la longitud de cualquier directiva, como una directiva page, excede de la anchura normal de una página JSP (80 caracteres), se debe dividir en varias líneas:

```
<%@ page    session="false"
 import="java.util.*"
 errorPage="/common/errorPage.jsp"
%>
```

En general, el segundo ejemplo es la opción preferida para definir la directiva page. Hay una excepción cuando necesitamos importar varios paquetes Java en la página JSP, dejando un atributo import muy largo:

```
<%@ page    session="false"
 import="java.util.*,java.text.*,
        com.mycorp.myapp.taglib.*,
        com.mycorp.myapp.sql.*, ..."
...
%>
```

En este escenario, se prefiere dividir la directiva page de esta forma:

```
<%-- all attributes except import ones --%>
<%@ page
...
%>
<%-- import attributes start here --%>
<%@ page import="java.util.*" %>
<%@ page import="java.text.*" %>
...
```

Observa que en general las sentencias import siguen las convenciones de codificación Java. Por ejemplo, generalmente se podría aceptar que cuando se utilicen hasta tres clases del mismo paquete, la sentencia import debería especificar las clases individualmente, en vez de su paquete. Si son más de tres clases, es el desarrollador web el que tiene que decidir si listar todas las clases individualmente o utilizar la notación ".*". El primer caso, hace más fácil identificar una clase externa, especialmente cuando intentamos localizar una clase o entender cómo el JSP interactúa con el código Java. Por ejemplo, sin conocer los paquetes Java importados como se muestra abajo, un desarrollador web tendría que buscar en todos esos paquetes para localizar una clase Customer:

```
<%@ page import="com.mycorp.bank.savings.*" %>
<%@ page import="com.thirdpartycorp.cashmanagement.*" %>
<%@ page import="com.mycorp.bank.foreignexchange.*" %>
...
```

En el último caso, es más difícil localizar las clases. En general, si una JSP tiene demasiada sentencias import, es que contiene demasiado código Java. Una mejor opción sería usar más etiquetas JSP.

Directiva(s) Tag Library Opcionales

Una directiva taglib declara las librerías de etiquetas usadas por el JSP. Un directiva corta se declara en una sola línea. Si tenemos varias directivas taglib se deben almacenar juntas en la misma localización dentro del cuerpo JSP:

```
<%@ taglib uri="URI1" prefix="tagPrefix1" %>
<%@ taglib uri="URI2" prefix="tagPrefix2" %>
```

...

Al igual que la directiva page, si la longitud de una directiva taglib excede la anchura de 80 caracteres, debemos dividirla en varias líneas

```
<%@ taglib
   uri="URI2"
   prefix="tagPrefix2"
%>
```

Sólo deberíamos importar librerías de etiquetas que realmente se van a utilizar en la página JSP.

Desde JSP 1.2, esta altamente recomendado utilizar la JSP Standard Tag Library en las aplicaciones web para reducir la necesidad de scriptlets JSP en las páginas. Las páginas que usan JSTL son, en general, más fáciles de leer y de mantener.

Declaraciones JSP Opcionales

Las declaraciones JSP declaran métodos y variables pertenecientes a la JSP. Estos métodos y variables no se diferencian de los declarados en el lenguaje Java, y por lo tanto se deberían seguir las convenciones de codificación más importantes. Es preferible que las declaraciones estén en un sólo bloque de declaración JSP <%! ... %>, para centralizar las declaraciones dentro de un área del cuerpo JSP. Aquí tenemos un ejemplo:

Bloque de Declaraciones Desaconsejadas

```
<%! private int hitCount; %>
<%! private Date today; %>
...
<%! public int getHitCount() {
       return hitCount;
    }
%>
```

Bloque de Declaraciones Preferidas

```
<%!
    private int hitCount;
    private Date today;

    public int getHitCount()
{
        return hitCount;
    }
%>
```

Declaraciones JSP

Al igual que en las convenciones de código Java, las declaraciones de variables de los mismos tipos deberían ir en línea separadas:

No recomendado

```
<%! private int x, y; %>
```

Recomendado

```
<%! private int x; %>
<%! private int y; %>
```

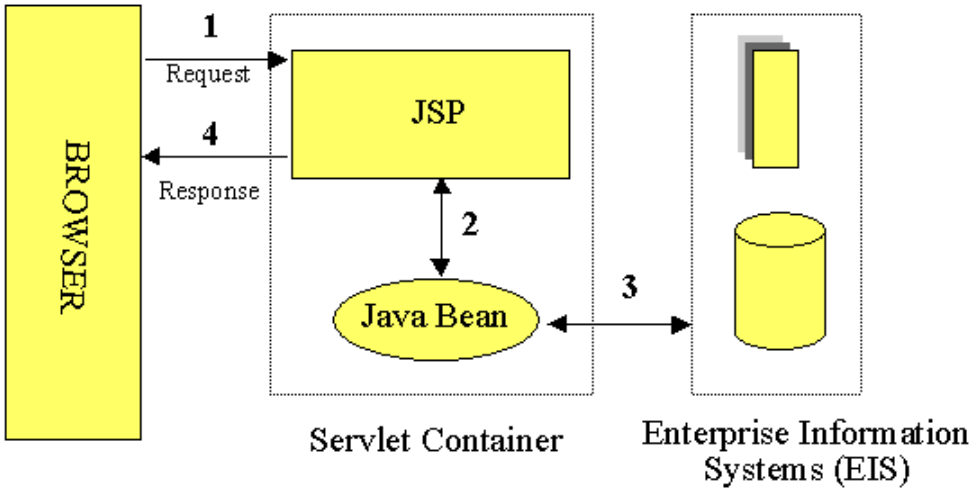
En general, se desaconsejan las declaraciones JSP para variables, ya que usan el lenguaje de script para mezclar la lógica de negocio y el código Java en un JSP que está diseñado para propósitos de presentación, y debido a la sobrecarga del manejo del ámbito de las variables.

Hay muchas más características de las páginas JSP que resultan de gran utilidad para poder desarrollar este tipo de aplicaciones, pero que no veremos en este seminario debido al carácter introductorio del mismo. Por este motivo, en los siguientes puntos, vamos a intentar crear un servlet y una página JSP a modo de ejemplo, para clarificar y aplicar lo visto hasta ahora.

4 SERVIDORES Y CONTENEDORES DE SERVLETS/JSPS

Para empezar, los JSPs y servlets se ejecutan en una máquina virtual Java, lo cual permite que, en principio, se puedan usar en cualquier tipo de ordenador, siempre que exista una máquina virtual Java para él. Cada servlet se ejecuta en su propia hebra, es decir, en su propio contexto; pero no se comienza a ejecutar cada vez que recibe una petición, sino que persiste de una petición a la siguiente, de forma que no se pierde tiempo en invocarlo (cargar programa + intérprete). Su persistencia le permite también hacer una serie de cosas de forma más eficiente, como conexión a bases de datos y manejo de sesiones, por ejemplo.

Los JSPs son en realidad como servlets: un JSP se compila a un programa en Java la primera vez que se invoca, y del programa en Java se crea una clase que se empieza a ejecutar en el servidor como un servlet. La principal diferencia entre los servlets y los JSPs es el enfoque de la programación: un JSP es una página Web con etiquetas especiales y código Java incrustado, mientras que un servlet es un programa que recibe peticiones y genera a partir de ellas una página web.



Ambos necesitan un programa que los contenga, y sea el que envíe páginas web al servidor, y reciba las peticiones, las distribuya entre los servlets, y lleve a cabo todas las tareas de gestión propias de un servidor web.

Mientras que servidores como el Apache están especialmente pensados para páginas web estáticas CGIs, y programas ejecutados por el servidor, tales como el PHP, hay otros servidores específicos para servlets y JSPs llamados *contenedores de servlets (servlet containers)* o *servlet engines*.

Los principales son los siguientes:

- ?? Resin, de Caucho Technologies, un motor especialmente enfocado al servicio de páginas XML, con una licencia libre para desarrolladores. Dice ser bastante rápido. Incluye soporte para Javascript además de Java. Incluye también un lenguaje de templates llamado XTP. Es bastante fácil de instalar, y en dos minutos, se pueden empezar a servir páginas JSP.

- ?? BEA Weblogic es un servidor de aplicaciones de alto nivel, y también de alto precio. Está escrito íntegramente en Java, y se combina con otra serie de productos, tales como Tuxedo, un servidor de bases de datos para XML.

- ?? JRun, de Macromediaun servidor de aplicaciones de Java, de precio medio y probablemente prestaciones medias. Se puede bajar una versión de evaluación gratuita

- ?? Lutris Enhydra, otro servidor gratuito y Open Source, aunque tiene una versión de pago. También enfocado a servir XML, y para plataformas móviles. Las versiones más actualizadas son de pago.

- ?? El más popular, y continuamente en desarrollo, es el Jakarta Tomcat, del consorcio Apache, un contenedor de servlets con muchos desarrollos adicionales alrededor; por ejemplo, Cocoon para servir páginas XML. Puede servir páginas sólo o bien como un añadido al servidor Apache. Es Open Source, relativamente rápido, y fácil de instalar.

Otros muchos se pueden encontrar en la página de Sun sobre la industria del servlet/JSP y en la página de contenedores de servlets en servlets.com

5 INSTALANDO UN CONTENEDOR DE SERVLETS

Vamos a ver ahora, a modo de ejemplo, como se instala un contenedor de Servlets, para luego ver un ejemplo de ejecución de JSPs.

Nos fijaremos especialmente en la versión 4.1 de Tomcat, (aunque lo que se cuente valdrá para casi todos los otros contenedores Open Source), puesTomcat es un contenedor de Servlets con un entorno JSP.

Podemos dividir los contenedores de Servlets en:

- **Contenedores de Servlets Stand-alone (Independientes)** Estos son una parte integral del servidor web., cuando este es un servidor web basado en Java. En estos casos, el contenedor de servlets forma parte de JavaWebServer (actualmente sustituido por **iPlanet**). Este el modo por defecto usado por Tomcat. Sin embargo, la mayoría de los servidores, no están basados en Java, los que nos lleva los dos siguientes tipos de contenedores:

- **Contenedores de Servlets dentro-de-Proceso** Este tipo de contenedor Servlet es una combinación de un plugin para el servidor web y una implementación de contenedor Java. El plugin del servidor web abre una JVM (Máquina Virtual Java) dentro del espacio de direcciones del servidor web y permite que el contenedor Java se ejecute en él. Si una cierta petición debería ejecutar un servlet, el plugin toma el control sobre la petición y lo pasa al contenedor Java. Un contenedor de este tipo es adecuado para servidores multi-thread de un sólo proceso y proporciona un buen rendimiento pero está limitado en escalabilidad

- **Contenedores de Servlets fuera-de-proceso** Este tipo de contenedor Servlet es una combinación de un plugin para el servidor web y una implementación de contenedor Java que se ejecuta en una JVM fuera del servidor web. El plugin del servidor web y el JVM del contenedor Java se comunican usando algún mecanismo IPC (normalmente sockets TCP/IP). Si una cierta petición debería ejecutar un servlet, el plugin toma el control sobre la petición y lo pasa al contenedor Java (usando IPCs). El tiempo de respuesta en este tipo de contenedores no es tan bueno como el anterior, pero obtiene mejores rendimientos en otras cosas (escalabilidad, estabilidad, etc.).

Tomcat puede utilizarse como un contenedor solitario (principalmente para desarrollo y depuración) o como plugin para un servidor web existente (actualmente se soportan los servidores Apache, IIS y Netscape). Esto significa que siempre que despleguemos Tomcat tendremos que decidir cómo usarlo, y, si seleccionamos las dos últimas opciones, también necesitaremos instalar un adaptador de servidor web.
Antes siquiera de bajarse el programa en cuestión, hay que considerar previamente de la máquina virtual Java que vamos a usar para ejecutarlo. Principalmente, hay dos opciones: las JVM de Sun (que son las originales) o las de IBM, que son algo más rápidas, pero que no siempre están actualizadas hasta la última versión.

Un contenedor de servlets necesita el JDK completo, no sólo el *runtime environment*, principalmente por el compilador de java contenido en un fichero llamado tools.jar. En cualquier caso, nos podemos bajar la JVM de Sun en sus versiones para Linux, o para cualquier otra plataforma (en la versión 1.4), o bien la versión 1.3 de IBM . Si no se consigue una versión de esas, es aconsejable conseguir una que sea compatible con la versión "2" de Java, es decir, JVMs a partir de la versión 1.2.

Una vez instalada la JVM, nos bajamos el servidor de su sitio correspondiente (http://jakarta.apache.org/site/binindex.cgi), si puede ser, en versión binaria, ya compilada.
Si disponemos de una distribución de Linux que use el RPM para instalación de paquetes (como Mandrake, RedHat o SuSE), se puede uno bajar los RPMs e instalarlos

directamente. Dependiendo de lo que elijamos, habrá que bajarse sólo un paquete, o bien varios. En todo caso, habrá que incluir lo siguiente: Xerces-J, regexp, servletapi, tomcat y tomcat-webapps.
Si bajamos el fichero .tar.gz, viene todo incluido.
Para instalar el RPM se hace lo siguiente:

```
[yo@mimaquina]$ rpm -Uvh tomcat4-4.1.noarch.rpm
```

(esto después de haber instalado todos los paquetes previos). Si se ha bajado el tar:

```
[yo@mimaquina]$    tar xvfz tomcat4-4.1.tar.gz
```

Ya desempaquetado, tendremos el Tomcat listo para funcionar. Dependiendo de la manera de descargargarlo, tendremos el servidor en un directorio u otro, pero siempre habrá directorios conf, lib, bin y webapps. En otros sistemas operativos, o con otro tipo de instalación, habrá que definir una variable de entorno, de esta forma:

```
[yo@mimaquina]$ export JAVA_HOME=/opt/j2sdk1.4
```

o bien

```
[yo@mimaquina]$ setenv JAVA_HOME /opt/j2sdk1.4
```

, dependiendo de si se trata del intérprete de comandos bash (el primero) o *csh (el segundo). En WinXX habrá que dar una orden similar. Finalmente para ejecutar el servidor, fijándonos en el caso de haberlo instalado usando el RPM, hacemos:

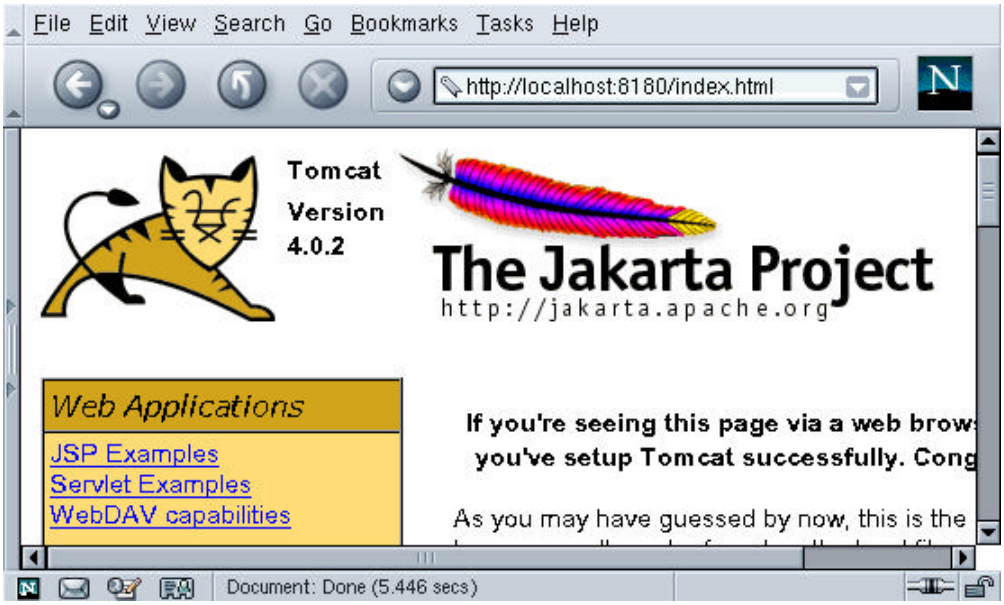
```
[yo@mimaquina yo]$ /etc/rc.d/init.d/tomcat4
start
```

En otros casos, habrá que ir al directorio bin de la distribución y ejecutar

```
[yo@mimaquina yo]$ <home_tomcat>/bin/startup.sh
```

o startup si se trata de Windows.

Por defecto, el servidor viene configurado para lanzarse en el puerto 8080 u 8180 (dependiendo de las versiones). Si todo ha ido bien, tendremos funcionando en nuestro ordenador al Tomcat, y al introducir una dirección tal como http://localhost:8080 o http://localhost:8180 obtendremos algo asi:



A partir de este momento, se pueden realizar una serie de cambios en la configuración de Tomcat para ahorrar recursos, mejorar rendimiento.. No entraremos en esos detalles, pués nos interesa centrarnos en el objetivo final: crear una página JSP.

6 NUESTRO PRIMER SERVLET

Una página JSP es un fichero de texto plano, que combina el uso de etiquetas HTML y código Java para ser interpretado dentro de un contenedor de JSP que forma parte del servidor web. Cuando el servidor recibe una solicitud de mostrar una página se desencadenan una serie de acciones que dan como resultado un Servlet que contiene la funcionalidad de la página y que será ejecutado en ese entorno.

Un Servlet, por tanto, es un programa Java capaz de procesar datos, y generar una respuesta utilizando el protocolo HTTP. Nuestro primer servlet tiene el siguiente código Java: (HolaMundo.java):

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HolaMundo extends HttpServlet{
    static int count=0;

    public void doGet(HttpServletRequest req,
    HttpServletResponse res)
    throws ServletException,IOException{
            res.setContentType("text/html");
            PrintWriter salidaHTML =
res.getWriter();
            salidaHTML.println("<html>");
            salidaHTML.println("<head>");
            salidaHTML.println("<title>");
```

```
            salidaHTML.println("Hola y números.
Intro to JSP");
            salidaHTML.println("</title>");
            salidaHTML.println("</head>");
            salidaHTML.println("<body
bgcolor='white'>");
            salidaHTML.println("Hola, mundo. Te
lo repito "+count++);
            if (count == 1){
                    salidaHTML.println("vez");
            }else{
                    salidaHTML.println("veces");
            }
            salidaHTML.println("</body>");
            salidaHTML.println("</html>");

}

}
```

El siguiente paso será compilar el servlet como cualquier programa java, con la diferencia de que necesitamos añadir al classpath la ruta del archivo .jar que contiene la utilidad de los servlets, es decir `servlet.jar`

```
javac -classpath
"$CLASSPATH:/usr/tomcat/common/lib/servlet.jar"
HolaMundo.java
```

Tras esto obtendremos el fichero compilado `HolaMundo.class`. Hasta este punto difiere poco de cualquier aplicación java que hayamos realizado con anterioridad. La gran diferencia es cómo tenemos que configurar el servidor de aplicaciones para que se pueda mostrar nuestro servlet al exterior.

?? En este paso colocaremos el servlet compilado (HolaMundo.class).Las clases del servlet compiladas deberán ir en

```
?<tomcat_home>/webapps/<directorio_contexto>/
   WEB_INF/classes
```

?? Por último tendremos que decirle al servidor donde se encuentra el servlet que hemos creado (puesto que cuando una página JSP es interpretada y compilada en un servlet, este es colocado por el propio contenedor en otro directorio "directorio work"). La forma de hacer esto es editando el archivo de configuración web.xml que se encuentra en el directorio WEB-INF de

nuestro contexto e indicando en el mismo donde se encuentra el servlet y como mapearlo en una llamada, tal y como muestra el siguiente ejemplo:

```
?<web-app>
??  <display-name>Tutorial JSP</display-name>
??  <description>
??      Tutorial de programacion JSP
??   </description>
??  <servlet>
??        <servlet-name>HolaMundo</servlet-
    name>
??        <servlet-class>HolaMundo</servlet-
    class>
??  </servlet>
??
??  <servlet-mapping>
??        <servlet-name>HolaMundo</servlet-
    name>
??        <url-pattern>/servlet/*</url-
    pattern>
??  </servlet-mapping>
?</web-app>
```

Así con servlet-name indicamos el nombre que le daremos en el navegador al servlet, y con servlet-class indicamos el nombre de la clase del servlet.
A continuación realizamos un mapeo donde indicamos que para llamar al servlet HolaMundo (Esto se hace con <servlet-name>HolaMundo</servlet-name>) hay que hacerlo a traves del directorio virtual /servlet/. Por tanto, si la dirección de nuestro servidor fuese http://www.miEjemploJSP.com, accederiamos al servlet HolaMundo con la linea http://www.miEjemploJSP.com/servlet/HolaMundo

7 NUESTRA PRIMERA PÁGINA JSP

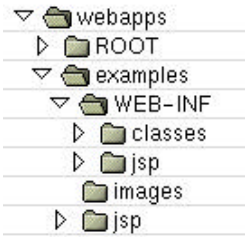
Si todo va bien, y los ejemplos se ejecutan correctamente, ya está uno listo para crear su primer programa, o página JSP (depende de como se mire). Como las páginas JSP son básicamente páginas HTML con un poco de Java, sirve, en principio, cualquier editor que comprenda la sintaxis HTML y/o XML; como por ejemplo, el XEmacs.

Un buen IDE para el desarrollo de JSPs sería eclipse al que se le puede añadir el plugin lomboz, para proporcionarle la interpretación de los JSP. (Tanto eclipse como lomboz son gratuitos)

Con cualquier editor de texto se puede crear la primera página JSP (`hola.jsp`):

```
<%@ page language='java'
contentType="text/html" %>
<%! int count=0; %>
<html>
<head><title>Hola y números. Intro to
JSP</title></head>
<body bgcolor="white">
Hola, mundo. Te lo repito  <%= count++ %>
<% if (count == 1) { %>
 vez
<% } else { %>
 veces
<% } %>
</body></html>
```

Tras editar esa página, habrá que ponerla en algún sitio. La estructura de directorios de Tomcat (y de otros contenedores de servlets) es un poco más compleja que la de los servidores web normales. Todos los ficheros cuelgan del directorio ***webapps***, pero no se pueden colocar directamente ahí. De ese directorio descienden otros subdirectorios, que es donde efectivamente se colocan las aplicaciones.



Cada directorio corresponde a un *contexto*; hay un contexto raíz `ROOT` y tantos otros contextos como queramos definir. Dentro de cada contexto, la estructura de directorios es también siempre la misma.

Directamente descendiendo del directorio se pueden poner los JSPs, y hay también un directorio `WEB-INF` donde hay una serie de ficheros de configuración y propiedades, y además otros directorios: `classes` y `lib` (que no aparece en la figura). En esos directorios se pondrán más adelante los ficheros `.class` y `.jar`, respectivamente.

Por tanto, el fichero anterior, irá a parar al directorio ROOT (o bien al directorio del contexto propio que hayamos definido en el ejercicio anterior). También se puede colocar en cualquiera de los subdirectorios del directorio `webapps`, pero no en el principal

El fichero contiene la mayoría de los elementos de un JSP. Para empezar, la línea 2 `<%@ page language='java' contentType="text/html" %>` incluye una ***directiva***, que son órdenes que se ejecutan antes de que se comience a procesar el JSP, y modifican de alguna forma el resultado del mismo. Todas las directivas en JSP se indican con una `@` después del comienzo de la orden JSP. En este caso, le indicamos que la página que se va a ejecutar está en lenguaje Java, y que el contenido que va a generar es de tipo `text/html`; el estándar JSP, a pesar de su nombre, no está limitado a un solo lenguaje, aunque en la práctica se usa casi siempre Java, y en algunos casos JavaScript.

En cuanto al contenido, JSP trabaja, por defecto, con HTML, pero se podrían generar páginas en otros estándares tales como el WML que se usa en los móviles WAP, o XML.

La directiva `page` puede incluir otros atributos:

```
<%@ page language='java'
           contentType="text/html"
           info='Mi primera página en JSP"
           import='java.util.*'
           errorPage='vayaerror.jsp' %>
```

`info` es una especie de comentario, que usa el contenedor para describir la clase en su interfaz de administración; `import` permite especificar una serie de paquetes en Java para importar, y `errorPage` indica la página a la que habría que saltar en caso de que se genere una excepción.

Ahora mismo, ninguno de estos atributos son necesarios, y, de hecho, la página funciona perfectamente bien sin ellos; en ese caso, tomaría los valores por defecto, es decir, lenguaje HTML para la página y Java para los scriptlets.

Justo a continuación, la línea `<%! int count=1; %>` es una *declaración*; la admiración (`!`) se usa para indicar declaraciones de variables globales, es decir, variables persistentes de una llamada a otra del JSP; y es compartida por todas las llamadas a una página. La declaración se ejecutará la primera vez que se llame a la página, y se volverá a ejecutar cada vez que se recompile la página (o se rearranque el servidor); el efecto que tendrá esta declaración será un *contador de visitas*, que se incrementará cada vez que se visite.

Por lo demás, la declaración es una declaración normal en Java; se pueden incluir tantas declaraciones como se quieran, no necesariamente al principio de la página.

A continuación vienen una serie de elementos HTML, hasta la línea 7, donde se encuentra la orden `<%= count++ %>`. Como dijimos al principio, todo lo que comience por `<%=` es una *expresión* JSP y el efecto que tiene es la evaluación de la expresión y la impresión del resultado. En este caso, se ejecuta la expresión y luego se imprime el valor, que es lo que hace el operador post-incremento `++`. Las expresiones en JSP pueden contener sólo expresiones Java que devuelvan un valor, y no hace falta que se terminen con `;`, como en el caso de las declaraciones.

A continuación se encuentra una combinación de código Java y elementos de texto que se suele denominar *scriptlet*. Los scriptlets usan la sintaxis JSP normal, `<%` y `%>`; y dentro, se puede incluir cualquier trozo de programa en Java que se quiera. En este caso es una sentencia `if`, y lo único relevante es que el código Java está mezclado con el texto; en este caso, se incluiría condicionalmente un texto u otro en la página de salida. Se podría hacer usando sólo Java, de esta forma

```
<% if (count == 1)
           System.out.println('vez')
} else {
           System.out.println('veces')
} %>
```

Sin embargo, de esta forma, se pueden incluir más fácilmente todo tipo de estructuras HTML, sin necesidad de meter montones de `System.out`.

Si no queremos tomarnos toda la molestia de meterlo en el contenedor de servlets, podemos intentar compilarlo antes, para ver si hay algún error que se nos hubiera saltado, por ejemplo de sintaxis. Se puede usar para ello el compilador de JSPs offline. Tomcat incluye `jasper`, que se puede usar de la siguiente forma (versión 4):

```
djasper4 jspc hola.jsp
```

Si todo va bien, se genera un fichero `hola.java` que, sería el que, compilado, daría el servlet; de hecho, este fichero se podría compilar y ponerse como servlet. Esta compilación no es necesaria, pero si estamos trabajando con un fichero grande y no tenemos muy claro si la sintaxis es correcta, se puede usar.

Si nos hemos equivocado en alguna cosa, el contenedor de servlets dará los errores correspondientes. Por ejemplo, si alguna etiqueta de JSP no la hemos puesto bien (`< } %>`), saldrá algo así:

```
org.apache.jasper.JasperException: No se puede
compilar la clase para JSP
/var/tomcat4/work/localhost/jmerelo/hola$jsp.ja
va:90: 'catch' without 'try'.
        } catch (Throwable t) {
          ^

/var/tomcat4/work/localhost/jmerelo/hola$jsp.ja
va:98:
'try' without 'catch' or 'finally'.
}
^

/var/tomcat4/work/localhost/jmerelo/hola$jsp.ja
va:98: '}' expected.
}
 ^
3 errors
```

A lo que hay que prestarle atención es a la última línea, la que dice que se esperaba un `}`. Como el cierre de llave está fuera de una etiqueta JSP, el intérprete no lo entiende. Sin embargo, si cometemos un error de sintaxis Java, la cosa estará un poco más clara:

```
org.apache.jasper.JasperException: No se puede
compilar la clase para JSP

An error occurred at line: 10 in the jsp file:
/hola.jsp

Generated servlet error:
```

```
/var/tomcat4/work/localhost/jmerelo/hola$jsp.ja
va:83:
Invalid expression statement.
                } ese {
                  ^


An error occurred at line: 10 in the jsp file:
/hola.jsp

Generated servlet error:
/var/tomcat4/work/localhost/jmerelo/hola$jsp.ja
va:83: ';' expected.
                } ese {
```

APÉNDICE: CONCEPTOS DE INTERÉS

SCRIPT

Los lenguajes interpretados o scripts forman un subconjunto de los lenguajes de programación, que incluye a aquellos lenguajes cuyos programas son habitualmente ejecutados en un intérprete en vez de compilados.

SERVLET

Los servlets son objetos que corren dentro del contexto de un servidor web y extienden su funcionalidad.

La palabra *servlet* deriva de otra anterior, *applet*, que se refería a pequeños programas escritos en Java que se ejecutan en el contexto de un navegador web. Por contraposición, un *servlet* es un programa que se ejecuta en un servidor web.

El uso más común de los *servlets* es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

CONTENEDOR DE SERVLETS

Es un shell de ejecución que maneja e invoca servlets por cuenta del usuario.

TCP

El Protocolo de Control de Transmisión (TCP en sus siglas en inglés, Transmission Control Protocol) es uno de los protocolos fundamentales en Internet. Muchos programas dentro de una red de ordenadores pueden usar TCP para crear *conexiones* entre ellos a través de las cuales enviarse datos. El protocolo garantiza que los datos serán entregados en su destino sin errores y en el mismo orden en que se transmitieron.

IP

El Protocolo de Internet (IP, de sus siglas en inglés *Internet Protocol*) es un protocolo orientado a conexión usado tanto por el origen como por el destino para la comunicación de datos a través de una red de paquetes conmutados.

HTML

El HTML, acrónimo inglés de Hypertext Markup Language (lenguaje de formato de documentos de hipertexto), es un lenguaje de marcas diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web.

HTTP

HTTP es el protocolo de la Web (WWW), usado en cada transacción. Las letras significan Hyper Text Transfer Protocol, es decir, protocolo de transferencia de hipertexto. El hipertexto es el contenido de las páginas web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceder a una página web, y la respuesta de esa web, remitiendo la información que se verá en pantalla.

IDE

Un entorno (o ambiente) integrado de desarrollo o en inglés *Integrated Development Environment* (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Consta de los siguientes componentes:

- ?? Un editor de texto.
- ?? Un compilador.
- ?? Un intérprete.
- ?? Herramientas de automatización.
- ?? Un depurador.
- ?? Posibilidad de ofrecer un sistema de control de versiones.
- ?? Factibilidad para ayudar en la construcción de interfaces gráficas de usuarios.

BIBLIOGRAFIA

http://www.programacion.com/java/articulo/conv_jsp/#jsp_file

http://webexperto.com/tutoriales/listado.php?cod=15

http://geneura.ugr.es/~jmerelo/JSP/

http://es.wikipedia.org/wiki/Portada