TUTORIAL JSP

INDICE

1. INTRODUCCIÓN A JSP

A modo de introducción, hay que decir que la web surgió como un sistema de información distribuido basado en red y en el protocolo de hipertexto (HTTP), ofreciendo por tanto, información estática.

En la actualidad, las aplicaciones web son cada vez más sofisticadas y requieren de una tecnología más avanzada para presentar la información dinámicamente. La tecnología JavaServer Pages (JSP) está posicionada en la "cresta de la ola" en la evolución del desarrollo de aplicaciones web.

La tecnología JSP está basada en el lenguaje de programación Java y encaminada a facilitar el desarrollo de sitios web. Mediante el uso de páginas JSP podemos incorporar contenido dinámico en sitios web mediante código Java embebido a través de etiquetas especiales < % % >.

Las páginas JSP son archivos de texto con extensión .jsp que contienen etiquetas HTML, junto con código Java embebido, que permite el acceso de la página a datos desde ese código Java ejecutado en el servidor.

Cuando se solicita una página JSP, la parte HTML se procesa en el cliente, sin embargo, el código Java se ejecuta en el momento de recibir la petición y el contenido dinámico generado por ese código se inserta en la página antes de devolverla al usuario. Esto proporciona una separación entre la parte de presentación HTML de la página y la parte de lógica de programación incluida en el código Java.

2. CLASS SERVLET

Los servlets son objetos que corren dentro del contexto de un contenedor de servlets (ejemplo: Tomcat) y extienden su funcionalidad. También podrían correr dentro de un servidor de aplicaciones (ej: OC4J Oracle), que, además de contenedor para servlet, tendrá contenedor para objetos más avanzados, como son los EJB (Tomcat sólo es un contenedor de servlets). La palabra servlet deriva de otra anterior, applet, que se refería a pequeños programas que se ejecutan en el contexto de un navegador web. Por contraposición, un servlet es un programa que se ejecuta en un servidor. 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.

El siguiente es el código de un servlet que crea una bean del anterior tipo la primera vez que se lo llama (método doGet ()), mientras que en las ocasiones posteriores modifica el valor de la variable msg con la fecha y horas actuales.

Tras ello, introduce la bean como un atributo del objeto request y transfiere el control a la página JSP

Llamada “/pag.jsp”.

```
Import java.util.Date;

Import java.io.*;

Import javax.servlet.*;

Import javax.servlet.http.*;

Import Xbean.AuxBean;

Public class ejBean extends HttpServlet {

AuxBean ab;

Protected void doGet (HttpServletRequest request, HttpServletResponse response)

Throws ServletException, IOException {

Ab = new AuxBean ();

ab.setMsg ("Pulsa para ver el mensaje de la bean...");

request.setAttribute ("laBean", ab);

ServletContext sc = getServletContext ();

RequestDispatcher rd = sc.getRequestDispatcher ("/pag.jsp");

rd.forward (request, response);
```

```
}
Protected void doPost (HttpServletRequest request, HttpServletResponse response)
Throws ServletException, IOException {
Date fecha = new Date ();
ab.setMsg (fecha.toString ());
request.setAttribute ("laBean", ab);
ServletContext sc = getServletContext ();
RequestDispatcher rd = sc.getRequestDispatcher ("/pag.jsp");
rd.forward (request, response);
}
}
```

3. FORMULARIOS EN HTML

Un formulario HTML es una sección de un documento que contiene contenido normal, código, elementos especiales llamados controles (casillas de verificación (checkboxes), radiobotones (radio buttons), menúes, etc.), y rótulos (labels) en esos controles. Los usuarios normalmente "completan" un formulario modificando sus controles (introduciendo texto, seleccionando objetos de un menú, etc.), antes de enviar el formulario a un agente para que lo procese (por ejemplo, a un servidor web, a un servidor de correo, etc.).

Aquí se muestra un ejemplo de un formulario simple que incluye rótulos, radiobotones y botones para reinicializar el formulario o para enviarlo:

```
<FORM action="http://algunsitio.com/prog/usuarionuevo" method="post">
  <P>
  <LABEL for="nombre">Nombre: </LABEL>
        <INPUT type="text" id="nombre"><BR>
  <LABEL for="apellido">Apellido: </LABEL>
        <INPUT type="text" id="apellido"><BR>
  <LABEL for="email">email: </LABEL>
```

```
        <INPUT type="text" id="email"><BR>
  <INPUT type="radio" name="sexo" value="Varón"> Varón<BR>
  <INPUT type="radio" name="sexo" value="Mujer"> Mujer<BR>
  <INPUT type="submit" value="Enviar"> <INPUT type="reset">
  </P>
</FORM>
```

3.1. ELEMENTO TEXTAREA

El elemento TEXTAREA crea un control de entrada de texto multilínea. Los agentes de usuario deberían usar los contenidos de este elemento como valor inicial del control y representar este texto inicialmente.

Este ejemplo crea un control TEXTAREA de 20 filas por 80 columnas que contiene inicialmente dos líneas de texto. El elemento TEXTAREA va seguido de botones de envío y reinicialización.

```
<FORM action="http://algunsitio.com/prog/leer-texto" method="post">
  <P>
  <TEXTAREA name="eltexto" rows="20" cols="80">
  Primera línea del texto inicial.
  Segunda línea del texto inicial.
  </TEXTAREA>
  <INPUT type="submit" value="Enviar"><INPUT type="reset">
  </P>
</FORM>
```

3.2. ELEMENTO SELECT

El elemento SELECT crea un menú. Cada opción ofrecida por el menú se representa por un elemento OPTION. Un elemento SELECT debe contener al menos un elemento OPTION.

Definiciones de atributos de SELECT:

Name = cdata [CI]

Este atributo asigna el nombre de control.

Size = número [CN]

Si un elemento SELECT se presenta como una lista con desplazamiento ("scrolled list box"), este atributo especifica el número de filas de la lista que deberían ser visibles al mismo tiempo. No es preciso que los agentes visuales presenten un elemento SELECT como una lista ("list box"); pueden usar cualquier otro mecanismo, como por ejemplo un menú desplegable ("drop-down menu").

Multiple [CI]

Si está activado, este atributo booleano permite selecciones múltiples. Si no está activado, el elemento SELECT sólo permite selecciones simples.

3.3. ELEMENTO FORM

El elemento FORM actúa como contenedor de controles. Especifica:

La disposición ("layout") del formulario (dada por los contenidos del elemento).

El programa que manejará el formulario completado y enviado (el atributo action). El programa receptor debe ser capaz de interpretar las parejas nombre/valor para poder hacer uso de ellas.

El método por el cual se enviarán los datos del usuario al servidor (el atributo method).

Una codificación de caracteres que debe ser aceptada por el servidor para poder manejar este formulario (el atributo accept-charset). Los agentes de usuario pueden avisar al usuario del valor del atributo accept-charset y/o restringir al usuario la posibilidad de introducir caracteres no reconocidos.

Un formulario puede contener texto y código (párrafos, listas, etc.) además de controles de formulario.

El siguiente ejemplo muestra un formulario que va a ser procesado por el programa "usuarionuevo" cuando sea enviado. El formulario será enviado al programa usando el método HTTP "post".

```
<FORM action="http://algunsitio.com/prog/usuarionuevo" method="post">
...contenidos del formulario...
</FORM>
```

3.4. ELEMENTO LABEL

El elemento LABEL puede utilizarse para adjuntar información a los controles. Cada elemento LABEL se asocia exactamente con un control de formulario.

El atributo for asocia explícitamente un rótulo con otro control: el valor del atributo for debe ser el mismo que el valor del atributo id del elemento de control asociado. Se puede asociar más de un LABEL con el mismo control creando múltiples referencias a través del atributo for.

Este ejemplo crea una tabla que se utiliza para alinear dos controles de entrada de texto y sus rótulos asociados. Cada rótulo está explícitamente asociado a un control de entrada:

```
<FORM action="..." method="post">
<TABLE>
 <TR>
  <TD><LABEL for="nombre">Nombre</LABEL>
  <TD><INPUT type="text" name="nombre" id="nombre">
 <TR>
  <TD><LABEL for="apellido">Apellido</LABEL>
  <TD><INPUT type="text" name="apellido" id="apellido">
</TABLE>
</FORM>
```

Este ejemplo extiende el formulario de un ejemplo previo haciendo que incluya elementos LABEL.

```
<FORM action="http://algunsitio.com/prog/usuarionuevo" method="post">
  <P>
  <LABEL for="nombre">Nombre: </LABEL>
      <INPUT type="text" id="nombre"><BR>
  <LABEL for="apellido">Last name: </LABEL>
      <INPUT type="text" id="apellido"><BR>
```

```
<LABEL for="email">email: </LABEL>
        <INPUT type="text" id="email"><BR>
<INPUT type="radio" name="sexo" value="Varón"> Varón<BR>
<INPUT type="radio" name="sexo" value="Mujer"> Mujer<BR>
<INPUT type="submit" value="Enviar"> <INPUT type="reset">
</P>
</FORM>
```

4. MANEJO DE SESIONES CON JSP

Cuando se solicita una página independientemente del tipo que sea, el servidor abre una conexión por la que envía los datos y luego ésta es cerrada una vez que ha terminado.

Una sesión es una serie de comunicaciones entre un cliente y un servidor en la que se realiza un intercambio de información.
Por medio de una sesión se puede hacer un seguimiento de un usuario a través de la aplicación.

Hay que poner el atributo session de la directiva page a true, de esta forma se notifica al contenedor que la página interviene en un proceso que utiliza las sesiones del protocolo

HTTP:

```
<%@page session='true'%>
```

Guardar los datos de una sesión:

Para guardar un objeto en una sesión se utiliza el método setAttribute (), que ha sustituido al método putValue (). Este método utiliza dos argumentos:

-El primero es el nombre que identificará a esa variable.

-El segundo es el dato que se va a guardar.

```
<%@page import="java.util.*" session="true" %> <%
HttpSession sesion=request.getSession ();
sesion.setAttribute ("trabajo","Paginas de JSP");
%>
```

Recuperar los datos de una sesión:

Los datos que se guardan en la sesión permanecen ahí a la espera de ser utilizados. Para ello es necesario realizar el proceso contrario a cuando se graban, comenzando por la recuperación del objeto de la sesión para empezar a ser tratado.

Para poder realizar este paso se utiliza el método getAttribute (), utilizando como argumento el nombre que identifica al objeto que se quiere recuperar. GetAttribute (java.lang, String nombre)

<% HttpSession sesion=request.getSession ();

Sesion.getAttribute (“nombre”); %>

5. MANEJO DE COOKIES CON JSP

Los COOKIES son variables que se guardan en pequeños archivos de texto en la computadora del cliente y que permiten guardar ciertas informaciones del cliente. Eso permite por ejemplo guardar el nombre del cliente para recuperarlo la próxima vez que el cliente se conecte.

Para crear un objeto de tipo Cookie se utiliza el constructor de la clase Cookie que requiere su nombre y el valor a guardar. El siguiente ejemplo crearía un objeto Cookie que contiene el nombre “nombre” y el valor “objetos”.

<% Cookie miCookie=new Cookie (“nombre”,”objetos”); %>

El proceso de recuperar un cookie determinado puede parecer algo complejo, ya que no hay una forma de poder acceder a un cookie de forma directa. Por este motivo es necesario recoger todos los cookies que existen hasta ese momento e ir buscando aquél que se quiera, y que al menos, se conoce su nombre.

Para recoger todos los cookies que tenga el usuario guardado se crea un array de tipo Cookie, y se utiliza el método getCookies () de la interfaz HttpServletRequest para recuperarlos:

<% Cookie [] todosLosCookies=request.getCookies ();

/*El siguiente paso es crear un bucle que vaya leyendo todos los cookies.*/

For (int i=0; i Cookie unCookie=todosLosCookies[i];

Todas las cookies tienen un período de caducidad. Cuando este tiempo se cumple, la cookie desaparece. Teniendo en cuenta este comportamiento cabe distinguir cuatro tipos de cookies:

- **Cookies de sesión**: desaparecen cuando se acaba la sesión, entendiendo por sesión el período de tiempo que un usuario está en un sitio web de manera continuada con la misma ventana del navegador.

- **Cookies permanentes**: si la fecha de caducidad de la cookie se corresponde con un futuro lejano, se puede decir que la cookie que es permanente. Esto no es del todo cierto, como se ha observado, ya que los navegadores cuentan con un espacio limitado para almacenar cookies de manera que las más nuevas hacen desaparecer a las más viejas cuando dicho espacio está totalmente ocupado.

- **Cookies con fecha de caducidad**: a veces los sitios web establecen cookies con una fecha de caducidad concreta. Por ejemplo, se puede utilizar una cookie para recordar a un cliente que existe una oferta. Esta cookie tendrá necesariamente la fecha de caducidad de la propia oferta, más allá de la cual no tienen ningún sentido que siga existiendo.

- **Cookies para borrar**: borrar una cookie existente es como volver a escribirla con una fecha de caducidad anterior a la fecha actual.

6. CONCLUSIONES

A modo de conclusión, hay que decir que Java Server Pages (JSP), es una tecnología orientada a crear páginas web con programación java y junto con Netbeans, lo que permite este tipo de software es proporcionarle al usuario herramientas para que el uso de este sea lo menos complicado posible.

7. BIBLIOGRAFÍA

A continuación, podemos observar la bibliografía que hemos utilizado para poder llevar a cabo este tutorial sobre JSP:

- http://www.suarezdefigueroa.es/manuel/IAW/Java/teoriajsp.html
- http://dis.um.es/~lopezquesada/documentos/IES_1314/IAW/curso/UT5/UT5.pdf
- http://www.desarrolloweb.com/articulos/831.php
- https://tomcat.apache.org/tomcat-5.5-doc/servletapi/javax/servlet/Servlet.html
- http://html.conclase.net/w3c/html401-es/interact/forms.html
- http://www.galisteocantero.com/sesiones-en-servlets-y-jsp-ejemplo-login-logout/
- http://es.slideshare.net/Sysworkap/jsp-con-session
- http://dis.um.es/~lopezquesada/documentos/IES_1213/IAW/curso/UT5/ActividadesAlumnos/12/cookies.html
- https://jtagua.wordpress.com/2010/10/18/tutorial-de-jsp-15-gestion-de-cookies/
- http://javaserverpages99.blogspot.com.es/2015/05/conclusiones-acerca-del-uso-de-estas.html