Tutorial de Introducción a JSP 2º ASIR

José Luis Pérez Morillas y Juan Antonio Martínez Sanchéz.

JavaServer Pages (JSP)

Las JavaServer Pages (JSP) nos permiten separar la parte dinámica de nuestras páginas Web del HTML estático. Simplmente escribimos el HTML regular de la forma normal, usando cualquier herramienta de construcción de páginas Web que usemos normalmente.

Se encierra el código de la partes dinámicas en unas etiquetas especiales, la mayoría de las cuales empiezan con “<%” y terminan con “%>”.

Por ejemplo:

```
<body>
    <%
        “Codigo Java”
    %>
</body>
```

Servlet

Los Servlet es una clase Java utilizada par extender las aplicaciones alojadas por servidores web, de tal manera que puedfen ser visto como applets de Java que se ejecutan en servidores en vez de navegadores web. Este tipo de servlets son la contraparte Java de otras tecnologías de contenido Web, como PHP o ASP.

La palabra servlet deriva de otra anterior, applet que se refiere a pequeños programas que se ejecutan en el contexto de un navegador web.

El uso más común es generar web de forma dinámica a partir de parámetros de la petición que envié el navegador.

El ciclo de vida de un servlet se divide en tres tiempos:

1 – Iniciar el servlet.

Cuando un servidor carga un servlet ejecuta el método del servlet. El servidor llama sólo una vez al método al crear la instancia del servlet.

2 – Interactuar con los clientes.

Después de la inicializaciónm el servlet puede dar servicio a las peticiones de los clientes. Estas peticiones serán atendidas por la misma instancia del servlet, por lo que hay que tener cuidado al acceder a variables compartidas ya que podrian darse problemas de sincornicación

3 – Destruir el servlet

Los servlets se ejecutan hasta que el servidor los destruye, por cierre del servidor o bien a petición del administrador del sistema. El servidor no ejecutara de nuevo el servlet hasta haberlo cargado e inicializado de nuevo.

Sintaxis JSP

Elemento JSP	Síntaxis	Interpretación
Expresión JSP	`<%= expresión %>;`	La expresión es evaluada y situada en la salida.
Scriptlet JSP	`<% código %>;`	El código se inserta en el método service.
Declaracón JSP	`<%! code %>`	El código se inserta en el cuerpo de la clase del servlet, fuera del método service.
Page JSP	`<%@ page att=”val” %>`	Dirige al motor servlet sobre la configuración
Directiva include JSP	`<%@ include file=”url” %>`	Un fichero del sistema local se incluirá traduzca a un Servlet
Comentario JSP	`<%-- comentario –%>`	Comentario ignorado cuando se traduce la página JSP en un servlet
Acción JSP:include	`<jsp:include page=”relative url flush=”true”/>`	Incluye un fichero en el momento en que la página es solicitada
Acción jsp:useBean	`<jsp:useBean %nbsp;att=val* /> <jsp:useBean$nbsp;att =val*> … </jsp:useBean>`	Encuentra o construye un Java Bean
Acción jsp:setProperty	`<jsp:setProperty att=val*/>`	Selecciona las propiedades del bean, bien directamente o designando el valor que viene desde un parámetro de la petición
Acción jsp:getProperty	`<jsp:getProperty name=”propertyName value=”valor” />`	Recupera y saca las propiedades del Bean
Acción jsp:forward	`<jsp:forward page=”relative URL” />`	Reenvía la petición a otra página.
Acción jsp:plugin	`<jsp:plugin attribute=”value”*>.. . </jsp:plugin>`	Genera etiquetas OBJECT o EMBED, apropiadas al tipo de navegado pidienteo que se ejecute un applet usando el Java Plugin.

Sesiones

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.

El tiempo de vida de una sesión comienza cuando un usuario se conecta por primera vez a un sitio web pero su finalización puede ser de tres maneras:

1 - Cuando se abandona el sitio web.

2 – Cuando se alcanza un tiempo de inactividad que es previaente establecido, en este caso la sesión es automáticamente eliminada. Si el usuario siguiera navegando se crearía una nueva sesión.

3 – Se ha cerrado a reiniciando el servidor.

Se utilizan para la identificación de usuarios principalmente, pero su aplicación puede ser infinita, como comercio electrónico donde estaría su “carrito”.

Con la identificación la pagina cambiara y podrá interactuar de una manera o otra según los permisos establecidos.

Para obtener la sesión de un usuario se utiliza el método getSession() que devuelve una interfaz de tipo HttpSession.

```
<% HttpSession sesion=request.getSession(); %>
```

Para guardar objetos en una sesión se utiliza el método setAttribute(), Este método utiliza dos argumetos:

```
SetAttribute(java.lang.String name, java.lan.Object value)
```

Ejemplo:

```
<%
        HttpSession sesion=request.getSession();
        String minombre = "Adrian";
        sesion.setAttribute("id", minombre);
   %>
```

Para obtener los objetos guardados en una sesión se utiliza el método getAttribute():

getAttribute(java.lang.String nombre)

Cuando este método devuelve el objeto, no establece en ningún momento de que tipo de objeto se trata. Por ello si se conoce previamente el tipo de objeto que puede devolver tras ser recuperado de la sesión es necesario realizar un casting, para convertirlo el objeto de tipo genérico al objeto exacto que se va a usar. Para realizar esta operación se añade el tipo de objeto al lado de tipo HttpSession que utiliza el método getAttribute() para obtener el objeto que devuelve:

```
<% HttpSession sesion=request.getSession();
```

```
String nombre=(String)sesion.getAttribute(“nombre”);
out.printIn(“Contenido de nombre:”+nombre);
%>
```

Si no existe ningún objeto almacenado en la sesión bajo el identificador que se utiliza en el método getAttribute(), el valor devuelto será null. Para ello se comprobara antes de usarlo para evitar errores:

```
<%
if (sesion.getAttribute(“nombre”)!=null)
{
String nombre = (String)sesion.getAttribute(“nombre”);
out.printIn(“Contenido de nombre: “+nombre);
}
%>
```

Y por último para cerrar la sesión usaremos el método invalidate():

```
<%@ page session=”true” %>
<%
HttpSession sesionOk = request.getSession();
sesionOk.invalidate();
%>
<jsp:forward page=”login.jsp”/>
```

Pero según la situación, como por ejemplo “deslogearnos” seria más recomendable en vez de quitar la sesión, simplemente poner “null” lo valores de nuestro usuario y pass.

Cookies

Una cookie no es más que un fichero de texto que algunos servidores piden a nuestro navegador que escriba en nuestro disco duro, con información acerca de lo que hemos estado haciendo por sus páginas.

Mediante el uso de cookies se permite al servidor Web recordar algunos datos concernientes al usuario, como por ejemplo si el usuario ha estado navegando anteriormente en la página web, que tipo de preferencias configuró el usuario, que clase de artículos lee para recomendarle otros similares, etc.

Las cookies representan una importante utilidad hoy en día en cuanto a información en la red.

Para crear un objeto Cookie, tendremos que utilizar el constructor de su clase correspondiente:

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

Con la línea anterior creamos la Cookie, pero necesitaremos configurarle una serie de parámetros:

```
if(rec!=null)
     {
             miCookie=new Cookie("login",login);
             miCookie.setMaxAge(60*60*24*31);
             miCookie.setPath("/");
             response.addCookie(miCookie);
     }
```

Estos atributos servirían para recordar el login de un usuario, con su correspondiente tiempo de vida.

Una función interesante que podemos añadir a nuestra cookie sería la de recordar datos que se encuentren en el ordenador del usuario y que provengan de otras páginas:

```
<%
    Cookie miCookie=null;
    String ciudad= request.getParameter(“formCiudad”);
    miCookie=new Cookie(“ciudadFavorita”,ciudad);
%>
```

Una vez tenemos la cookie creada con sus atributos, solo nos quedaría una pregunta, ¿como recuperamos esos datos que hemos recogido del usuario? Podremos hacerlo de esta manera:

```
<%
    Cookie [] todosLosCookies=request.getCookies();
    Cookie unCookie=null;
    /* El siguiente paso es crear un bucle que vaya leyendo todos
      los cookies. */
    for(int i=0;i<todosLosCookies.length;i++)
    {
    Cookie unCookie=todosLosCookies[i];
    if(unCookie.getName().equals(“nombre”))
    break;
    }
    out.println(“Nombre: “+unCookie.getName()+”<BR>”);
    out.println(“Valor: “+unCookie.getValue()+”<BR>”);
    out.println(“Path: “+unCookie.getPath()+”<BR>”);
    out.println(“Tiempo de vida:“+unCookie.getMaxAge()+”<BR>”);
    out.println(“Dominio: “+unCookie.getDomain()+”<BR>”);
%>
```

Formularios

Para que los usuarios introduzcan datos en un programa, se utilizan los formularios. Estos formularios se crean con la etiqueta <form>...</form>

```
<form action="destino" method="metodo"> ... </form>
```

Dentro del formulario podemos entontrar los parámetros action=”” y method=””. El parámetro action contiene el destino al que mandaremos los datos que se introduzcan en el formulario, mientras que el parámetro method indica el método que usaremos para enviar los datos. Este método puede ser GET o POST.

Con GET, el conjunto de datos recogido en el formulario se incluye en la URL y se envía al agente procesador; mientras que con POST agregamos los datos al cuerpo del formulario y lo mandamos al procesador.

En cuanto a las etiquetas de los formularios, usaremos la etiqueta <select>...</select> para hacer selecciones de un menú:

```
<select name="nombre" [size="filas" multiple]>
      <option value=“01”>valor1</option>
      <option deleted value=“02”>valor2</option>

</select>
```

Para introducir areas de texto, utilizaremos <textarea>...</textarea> :

```
<textarea name="nombre" rows="filas" cols="columnas">
            [texto predefinido]
</textarea>
```

Con <input> podemos agregar diferentes tipos de campos, dependiendo del parámetro que le añadamos. Los diferentes parámetros dentro de input pueden ser los siguientes:

- **text**: que sirve para introducir una caja de texto simple.
- **radio**: define un conjunto de elementos de formulario de tipo circular, en los que el usuario debe optar por uno de ellos, que se marca con el ratón o tabulador.
- **checkbox**: define una o más casillas de verificación, pudiendo marcar las que se desee del conjunto total. Si pinchamos una casilla con el ratón o la activamos con el tabulador y le damos a la barra espaciadora la casilla se marca; si volvemos a hacerlo, la casilla se desmarca.
- **button**: define un botón estándar. Que puede ser usado para diferentes acciones, normalmente mediante JavaScript, con el evento "OnClick".
- **password**: define una caja de texto para contener una password, por lo que el texto que introduzca el usuario aparecerá como asteriscos, por motivos de seguridad.
- **hidden**: define un campo invisible, que no se ve en pantalla. Sus usos son varios e importantes.
- **submit**: incorpora al formulario un botón de envío de datos. Cuando el usuario pulsa este botón los datos que se han introducido en los diferentes campos del formulario son enviados al programa del servidor o a la dirección de correo indicada en action.
- **reset**: define un botón que al ser pulsado por el usuario borra todos los datos que hubiera introducido en cualquiera de los campos del formulario.

Además de los formularios, mediante los caracteres de “?” y “&” colocándolos en un enlace también podemos enviar información

```
<a href=”recoge.jsp?valor=1&valor2=2”> Enviar </a>
<a href=”recoge.jsp?valor=<%=expresion_java%>+”&valor2=2”> Enviar </a>”);
```

De este modo nos aparece un enlace Enviar en el cual llama a recoge.jsp enviando dos datos (nombre/valor).

Ahora para recuperer estos datos para poder procesarlos en el programa jsp y generar un resultado:

Con `request.getParameter(“campo”)` Obtenemos el contenido de la variable campo y se lo asignamos a la variable aux.

Los datos vienen en modo String aun que sean números, es decir si queremos usar un integer,

deberemos transformarlos

Y en el caso de tener un campo/atributo que puede devolver más de un valor usaremos el:

`request.getParametersValues()`

Y lo meteremos en un `String[]`