Introducción a JSP

Introducción a JSP

JavaServer Pages (JSP) (http://java.sun.com/jsp) es una tecnología basada en el lenguaje Java que permite incorporar contenido dinámico a las páginas web. Los archivos JSP combinan HTML con etiquetas especiales y fragmentos de código Java.

El código fuente de una página JSP puede contener:

- Directivas: Indican información general de la página, como puede ser importación de clases, página a invocar ante errores, si la página forma parte de una sesión, etc.
- Declaraciones: Sirven para declarar métodos o variables.
- Scriptlets: Código Java embebido.
- Expresiones: Expresiones Java que se evalúan y se envían a la salida.
- Tags JSP: Etiquetas especiales que interpreta el servidor.

Un ejemplo simple de archivo JSP sería:

```
<%@page import="java.util.*"%>
<%! String cadena="Bienvenidos a JSP"; %>
<html>
      <body>
            <%= cadena %>
            <br>
            <% out.println(new Date()); %>
      </body>
</html>
```

JSP tiene bastantes ventajas frente a otras orientaciones, como ASP o PHP. Al ser JSP una especificación, podemos elegir entre diversas implementaciones, comerciales o gratuitas, sin tener que depender de un proveedor en particular. Quizá la ventaja fundamental es que tenemos toda la potencia del lenguaje Java a nuestro alcance, con sus ventajas como reusabilidad, robustez, multiplataforma, etc.

1. Elementos de JSP

1.1. Código Java

Podemos insertar código Java dentro de JSP de tres formas: Expresiones, *scriptlets* y declaraciones.

1.1.1. Expresiones: Son fragmentos de código Java, con la forma `<%= expresión %>` que se evalúan y se muestran en la salida del navegador. En general, dentro de una expresión podemos usar cualquier cosa que usaríamos dentro de un `System.out.print(expr);`

Ejemplos:

```
<%= "Tamaño de cadena: "+cadena.length() %>
<%= new java.util.Date() %>
<%= Math.PI*2 %>
```

1.1.2. *Scriptlets*: Son fragmentos de código Java con la forma `<% código %>`, en general, podemos insertar cualquier código que pudiéramos usar dentro de una función Java. Para acceder a la salida del navegador, usamos el objeto implícito `out`.

Ejemplos:

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

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

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.

```
<%
      String parametro1=request.getParameter("parametro1");
      String parametro2=request.getParameter("parametro2");
      MiClase miClase=new MiClase();
      String salida=miClase.procesa(parametro1, parametro2);
%>
<%= salida %>
```

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 */
%>
```

1.1.3. Declaraciones: Contienen declaraciones de variables o métodos, con la forma `<%! 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.

Ejemplos:

```
<%! int numeroAccesos=0; %>
<html>
      <body>
            <%=
                "La página ha sido accedida "+(++numeroAccesos)+
                " veces desde el arranque del servidor"
            %>
      </body>
</html>

<%! java.util.Date primerAcceso=new java.util.Date(); %>
<html>
      <body>
            El primer acceso a la página se realizo en:
            <%= primerAcceso %>
      </body>
</html>
```

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

1.2. Directivas

Las directivas son elementos que proporcionan información al motor JSP, e influirán en la estructura del *servlet* generado. Hay tres tipos de directivas: `page`, `taglib` e `include`.

1.2.1. page: Se indica con la forma `<%@ page atributo="valor">`. Tiene diversos usos, entre los cuales destacaremos:

- Importar clases. Importar código, de la misma forma que se realiza en un programa en Java, se indica con el atributo `import`.
  Ejemplo:
  ```
  <%@page import="java.io.*, miPackage.miClase"%>
  ```

- Indicar si la página tendrá acceso a la sesión. Se especifica con el atributo `session`. El uso de sesiones se verá con más detalle en el apartado de objetos implícitos.
  Ejemplo:
  ```
  <%@page session="true" import="java.util.ArrayList"%>
  ```

- Gestión de errores. Permite redireccionar a una página cuando se produzca un error, se indica con los atributos `errorPage` y `isErrorPage`.
  Ejemplos:
  ```
  <%@page errorPage="error.jsp">
  [...]

  <%@page isErrorPage="yes">
  <html>
  	<body>
  		Error, contacte con el administrador [...]
  	</body>
  </html>
  ```

1.2.2. include: Permite incluir un archivo en el lugar donde se especifique, al contrario que con la acción `<jsp:include>` que veremos más adelante, la directiva `include` simplemente copia el contenido del archivo byte a byte, siendo el resultado similar a si copiáramos el texto del archivo incluido y lo pegáramos en el JSP.
Ejemplo:

```
<html>
	<head>
		<%@ include file="titulo.txt"%>
	</head>
	<body>
		<%@ include file="cuerpoPagina.jsp"%>
	</body>
</html>
```

1.2.3. taglib: Se emplea para indicar que se van a emplear librerías de etiquetas. Se verá con más detalle en el apartado de JSTL.
Ejemplo:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

1.3. Acciones

Las acciones tienen la forma `<jsp:accion [parámetros]/>`, y tienen diversos usos, entre los que destacan la inclusión de páginas y transferencia de control.

1.3.1. Inclusión de páginas

Se realiza con la acción `<jsp:include page="pagina.jsp">`. Incluye la salida de otra página JSP en la actual, al contrario que con la directiva `<%@include file="fichero.ext"%>` la página incluida se *ejecuta* y su salida se inserta en la página que la incluye, con la directiva se incluye el contenido del archivo (no su salida) y se ejecuta conjuntamente con la página principal.

La página incluída tiene acceso a los parámetros enviados a la principal, y podemos enviarle nuevos parámetros con la subetiqueta `<jsp:param name="nombre" value="valor"/>`.
Ejemplo:

```
<html>
      <head>
            <jsp:include page="cabecera.jsp"/>
      </head>
      <body>
         <jsp:include page="cuerpo.jsp">
            <jsp:param name="tipo" value="paginaPrincipal"/>
         </jsp:include>
      </body>
</html>
```

1.3.2. Transferencia de control

Se realiza con la acción `<jsp:forward page="pagina.jsp"/>`. La petición es redirigida a otra página, y la salida de la actual se descarta. Al igual que con la inclusión, la página a la que se redirige tiene acceso a los parámetros pasados a la actual, y es posible el envío de nuevos parámetros.
Ejemplo:

```
<jsp:forward page="principal.jsp">
      <jsp:param name="titulo" value="Principal"/>
</jsp:forward>
```

2. Objetos implícitos

En JSP disponemos de algunos objetos implícitos, que nos permitirán acceder a diferente información y realizar diversas acciones. En JSP tenemos los siguientes objetos implícitos: `request`, `response`, `out`, `session`, `application`, `config`, `pageContext`, y `page`. Podemos acceder al *JavaDoc* de estas clases para ver los atributos y funciones disponibles en las direcciones:
http://java.sun.com/products/servlet/reference/api/index.html
http://java.sun.com/products/jsp/reference/api/index.html

2.1. `request`

Es un objetode la clase `HttpServletRequest`, su uso principal es el acceso a los parámetros de la petición. Destacaremos las siguientes funciones:

- `String getParameter(String name)`
  Devuelve el valor de un parámetro.

- `Enumeration getParameterNames()`
  Devuelve una enumeración con los nombres de todos los parámetros de la petición.
- `String[] getParameterValues(String name)`
  Los parámetros pueden tener valor múltiple, con esta función recuperamos un array con todos los valores para un nombre dado.
- `String getRemoteAddr()`
  Devuelve la IP del *host* desde el que se realiza la petición

- `String getRemoteHost()`
  Devuelve el nombre del *host* desde el que se realiza la petición.

Ejemplo:

```
<html>
	<body>
		<form>
			<input type="text" name="parametro"/>
			<input type="submit"/>
		</form>
		<br>
		<br>
		Su IP: <%=request.getRemoteAddr()%>
		<br>
		Su nombre de host: <%= request.getRemoteHost() %>
		<br>
		Valor del parámetro:
		<%= request.getParameter("parametro") %>
	</body>
</html>
```

2.2. response

Es un objeto de la clase `HttpServletResponse`, que asiste al *servlet* en su generación de la respuesta para el cliente, contiene funciones para manejo de cabeceras, códigos de estado, *cookies* y transferencia de control.

2.3. out

Es un objeto de la clase `JspWriter`, es el que nos permite acceder a la salida del navegador desde los *scriplet*.

Ejemplo:

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

2.4. session

Es un objeto de la clase `HttpSession`. Nos permite acceder a la sesión asociada a la petición. A través de este objeto podemos, entre otras cosas, guardar objetos que serán accesibles desde cualquier JSP de la sesión o invalidarla.

Para guardar y recuperar información usaremos:

```
Object session.getAttribute("clave");
void session.setAttribute("clave", Object objeto);
```

Y para invalidar la sesión:

```
void session.invalidate();
```

Ejemplo:

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

<%
      java.util.ArrayList accesos=
            (java.util.ArrayList)session.getAttribute("accesos");
      if (accesos==null)
            accesos=new java.util.ArrayList();
      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 sesión"/>
                  <input type="submit" value="Recargar página"/>

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

2.5. application

Es un objeto de la clase `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.
Para guardar y recuperar valores:

```
Object application.getAttribute("clave");
void application.setAttribute("clave", Object objeto);
```

Ejemplo:

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

<%
      java.util.Hashtable direcciones=
            (java.util.Hashtable)application.
                  getAttribute("direcciones");
      if (direcciones==null)
            direcciones=new java.util.Hashtable();
      direcciones.put(request.getRemoteAddr(),"");
      application.setAttribute("direcciones", direcciones);
%>

<html>
      <body>
            El servidor fue accedido desde las
            siguientes direcciones IP:
            <br>

            <%
                  java.util.Enumeration e=
                        direcciones.keys();
                  while (e.hasMoreElements())
                  {
            %>
                  <%= e.nextElement() %>
                  <br>
            <%
                  }
            %>
      </body>
</html>
```

2.6. config

Es un objeto de la clase `ServletConfig`. Permite acceder a parámetros de inicialización del *servlet* y a su contexto.

2.7. pageContext

Es un objeto de la clase `PageContext`. Entre otras cosas, nos permite almacenar información localmente a la página.
Para guardar y recuperar valores:

```
Object pageContext.getAttribute("clave");
void pageContext.setAttribute("clave", Object objeto);
```

También podemos usar PageContext para almacenar y recuperar información en sesión y en aplicación:

Almacenar en contexto de página:

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

Almacenar en contexto de sesión:

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

Almacenar en contexto de aplicación:

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

2.8. page

Es un sinónimo de `this`, no tiene utilidad en el estado actual de la especificación.

3. Comunicación entre formularios HTML y páginas JSP

En esta sección se hará un breve repaso a los formularios HTML, y veremos las distintas formas de enviar parámetros desde un formulario y de recibirlos desde un JSP. Dejaremos al margen algunos tipos como botones y envío de archivos, por tener relevancia sólo en lenguajes de *script* del lado del cliente o quedar fuera del alcance de este tutorial.

3.1. Conceptos básicos

Un formulario HTML tiene la forma:

```
<form action="destino" method="método">
      Elementos de formulario
</form>
```

En *destino* especificaremos la página que recibe los datos del formulario (p.e. *procesaformulario.jsp*), en el atributo `method` podemos indicar dos valores diferentes `GET` y `POST`. Si no se especifica el valor de los atributos, los valores por defecto son la página actual para `action` y `GET` para `method`

3.2. GET y POST

Cuando usamos `GET`, la información se codifica directamente en la URL, con la forma:

```
http://url?param1=valor1&param2=valor2...&paramN=valorN
```

Con `GET` no podemos manejar grandes cantidades de información, y existe la desventaja de que el servidor o el navegador guarden en caché la página llamada. Hay que tener en cuenta que los *logs* del servidor y el historial del navegador guardarán el acceso incluyendo los parámetros, lo cual hace desaconsejable `GET` para el envío de información privada. Por otro lado, al visualizarse en la URL los parámetros, facilita el desarrollo y depurado de la aplicación web, y en algunos casos, es imprescindible para realizar estadísticas basadas en los *logs* del servidor.

Con `POST` la información se envía directamente al servidor, no se codifica en la URL, y además permite el envío de grandes cantidades de información, como podrían ser archivos.

3.3. Elementos de formulario

Se indican con las etiquetas HTML:

```
<input type="tipo" name="nombre" value="valor"/>
<textarea name="nombre"/>Contenido por defecto</textarea>
<select name="nombre">
      <option value="valorOpcion">Texto opcion</option>
      [...]
</select>
```

Para enviar los datos usamos el tipo `submit`.

```
<input type="submit"/>
```

3.3.1. Campos de texto.

Los tipos que se envían como texto simple son `text` y `password` para `<input>`, y el elemento `<textarea>`.

Ejemplo:

```
<form action="pagina.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>
      <input type="submit"/>
</form>
```

Y en el archivo `pagina.jsp`:

```
Valor de parametro1: <%= request.getParameter("parametro1") %>
<br>
Valor de parametro2: <%= request.getParameter("parametro2") %>
<br>
Valor de parametro 'clave':
<%= request.getParameter("clave") %>
```

3.3.2. Selectores ON/OFF. `Checkbox`.

Se indica con el tipo `checkbox`.

```
<input type="checkbox" name="nombreCheckbox"/>
```

Si el `checkbox` está marcado, se envía un parámetro con el nombre especificado con el valor `on`. Si no está marcado, no se envía el parámetro. Así que podemos recibirlo en JSP de la siguiente forma:

```
<%
      String checkbox= request.getParameter("nombreCheckbox");

      if (checkbox!=null && checkbox.equalsIgnoreCase("on"))
      {
%>
      Checkbox seleccionado en el formulario origen.
<%
      }
      else
      {
%>
      Checkbox NO seleccionado en el formulario origen.
<%
      }
```

```
%>
```

3.3.3. Parámetros ocultos. Tipo `hidden`.

Se indican con el tipo `hidden`, los pares clave valor indicados se enviaran siempre junto con el resto de información del formulario. Se reciben en el JSP de la misma forma que los campos de texto.
Ejemplo:

```
<form action="pagina.jsp">
      <input type="text" name="variable" value="por defecto"/>
      <input type="hidden" name="fijo" value="valor fijo"/>
</form>
```

3.3.4. Botones `radio`.

Son grupos de valores ON/OFF, sólo puede haber uno seleccionado dentro del grupo con el mismo nombre en el atributo `name`. Sólo se envía un parámetro para el botón seleccionado, con el valor indicado en la etiqueta `value`. El valor se recibe en el JSP de forma similar a los campos de texto.
Ejemplo:

```
<form action="pagina.jsp">
      Opcion 1 <input type="radio" name="radio" value="uno"/>
      <br>
      Opcion 2 <input type="radio" name="radio" value="dos"/>
      <br>
      <input type="submit"/>
</form>
```

3.3.5. Selecciones. Etiquetas `<select>` y `<option>`.

Se usan para desplegables y listas. Se especifica un nombre para el parámetro y se envía como valor el contenido de la etiqueta `<option>`. Si especificamos el atributo `value` en la etiqueta `option`, Se mostrará en el desplegable el texto en el cuerpo de la etiqueta, pero se enviará el valor especificado en el atributo `value` en caso de ser la opción seleccionada.

Ejemplo:

```
<form action="pagina.jsp">
      <select name="selectSimple">
            <option value="1">Uno</option>
            <option>Dos</option>
            <option>Tres</option>
            <option>Cuatro</option>
      </select>
      <input type="submit"/>
</form>
```

También podemos crear listas que nos permitan la selección de múltiples valores, esto se indica añadiendo el atributo `MULTIPLE` al `<select>` en este caso se generan parámetros con el mismo nombre para cada opción seleccionada, lo que nos obliga a recuperar estos valores con la función:
`String[] getParameterValues(String name).`

Ejemplo:

```
<form action="pagina.jsp">
      <select name="selectMultiple" MULTIPLE>
            <option value="1">Uno</option>
            <option>Dos</option>
            <option>Tres</option>
            <option>Cuatro</option>
      </select>
      <input      type="submit"/>
</form>
```

Y en `pagina.jsp`:

```
Se marcaron las siguientes entradas:
<br>
<%
      String[] seleccion=
            request.getParameterValues("selectMultiple");
      for (int i=0;i<seleccion.length;i++)
      {
%>
      <%= seleccion[i] %>
      <br>
<%
      }
%>
```

3.3.6. Codificación directa en el `URL`.

En algunos casos resulta útil enviar directamente información a JSP, codificada en la URL.

Ejemplo:

```
<a href="noticias.jsp?param=nacional">Actualidad</a> <br>
<a href="noticias.jsp?param=deportes">Deportes</a> <br>
<a href="noticias.jsp?param=sociedad">Sociedad</a> <br>
```

Debemos tener cuidado cuando enviemos caracteres especiales, para codificar cadenas podemos usar la función estática `String java.net.URLEncoder.encode(String s);`
Ejemplo:

```
<%
String url="armas.jsp?tipo=cañón";
String urlCodificada=
      "armas.jsp?tipo="+java.net.URLEncoder.encode("cañón");
%>
Pulse un enlace y compruebe los parámetros en
la barra de direcciones.
<br>
<a href="<%=url%>"/>URL sin codificar</a>
<br>
<a href="<%=urlCodificada%>"/>URL codificada</a>
```

4. Introducción a JSTL

En JSP es posible definir librerías de etiquetas personalizadas, estas etiquetas no son más que clases Java que heredan de determinadas clases (p.e. `BodyTagSupport`). Estas clases se agrupan en librerías mediante un archivo descriptor TLD (*Taglib descriptor*). Queda fuera del ámbito de este tutorial la construcción de librerías de etiquetas. Existen innumerables librerías, comerciales y gratuitas, que implementan las más diversas funciones.

A partir de JSP 1.2. se introduce un conjunto de librerías en la especificación, pasando a ser estándar, es la librería JSTL (*JavaServer pages Standard Tag Library*).
JSTL consta de los siguientes grupos de etiquetas:

Tipo	URI (identificador)	PREFIJO
Core	http://java.sun.com/jsp/jstl/core	c
XML	http://java.sun.com/jsp/jstl/xml	x
Internacionalización y formato	http://java.sun.com/jsp/jstl/fmt	fmt
SQL	http://java.sun.com/jsp/jstl/sql	sql

En este tutorial sólo veremos algunas etiquetas de *core* y formato.

4.1. *Expression language* (EL).

Además de las librerías de etiquetas, JSTL define un lenguaje de expresiones (EL), que facilita enormemente el tratamiento de información. Las expresiones se indican de la forma `${expresion)`.

En los ejemplos usaremos la etiqueta `<c:out>`, que veremos en el apartado dedicado a la librería *core*. En las expresiones podemos usar los operadores típicos `+,-,*,/,mod, >,<,<=,>=,==,!=, &&, ||, !` además del operador `empty`, que nos servirá para comparar a la vez con `null` y con cadena vacía.

Con EL podemos además acceder a todos los objetos implícitos, y se añaden los objetos `param`, `paramValues` y `header`. En EL los objetos implícitos disponibles son: `pageContext`, `pageScope`, `requestScope`, `sessionScope`, `applicationScope`, `param`, `paramValues` y `header`.

Para acceder a un atributo dentro de un objeto, podemos usar los operadores '.' y '[]', de la forma `objeto.atributo` ó `objeto["atributo"]`.

Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:out value="${1+2+3}"/>
<br>
<%= request.getParameter("parametro")%>
<br>
<c:out value="${param.parametro}"/>
<br><br>
<%
      application.setAttribute("atributo", "valor");
%>
<%= application.getAttribute("atributo")  %>
<br>
<c:out value="${applicationScope.atributo}"/>
```

En el ejemplo vemos que con `<%= request.getParameter("parametro")%>` se muestra null si el parámetro no está definido, mientras que con `<c:out value="${param.parametro]"/>` se muestra cadena vacía.

En las últimas versiones de la especificación, se pueden embeber directamente expresiones EL, de forma que se hace innecesario el uso de `<c:out value="`*`expresion`*`"/>` para mostrarlas.

Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
Valor del parámetro 'parametro': ${param.parametro}
```

4.2. La librería *core.*

En las páginas que la usen deberemos incluir la siguiente directiva:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

Esta librería implementa acciones de propósito general, como mostrar información, crear y modificar variables de distinto ámbito y tratar excepciones. Veremos algunas de las etiquetas más comunes.

4.2.2. `out`

Muestra información en la página, se muestra la expresión contenida en el atributo `value`.

Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<br> <c:out value="1+2+3"/>
<br> <c:out value="${1+2+3}"/>
<br> <c:out value="${param.nombreParametro}"/>
<br> <c:out value="${sessionScope.variableDeSesion}"/>
```

4.2.3. `set`

Guarda información en una variable, tiene los siguientes atributos:

- `var`. Nombre de la variable
- `scope`. Ámbito de la variable, admite los valores `page`, `session` y `application`.

Para eliminar una variable podemos usar:

```
<c:remove var="nombreVariable" scope="ambito"/>
```

Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="variableDePagina" scope="page">
      Esta información se guarda en la página
</c:set>
<c:set var="variableDeSesion" scope="session">
      Esta información se guarda en la sesión
</c:set>
<c:set var="variableDeAplicacion" scope="application">
      Esta información se guarda en la aplicación
</c:set>

<br>${variableDePagina}
<br>${variableDeSesion}
<br>${variableDeAplicacion}
```

4.2.4. `if`

Procesa el cuerpo de la etiqueta si la condición se evalua a cierto. La condición se indica en el atributo `test`.

Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:if test="${empty param.nombre}">
      Parámetro 'nombre' no definido.
</c:if>
```

4.2.5. choose, when y otherwise

Procesa condiciones múltiples, se procesa el cuerpo del primer `when` cuya condición especificada en el atributo `test` se evalúe a cierto. Si ninguna de las condiciones se cumple, se procesa el cuerpo de `otherwise` en caso de que aparezca.
Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:choose>
      <c:when test="${empty param.nombre}">
            Parámetro 'nombre' no definido.
      </c:when>
      <c:otherwise>
            Valor del parámetro 'nombre':
            ${param.nombre}
      </c:otherwise>
</c:choose>
```

4.2.6. forEach y forTokens .

`forEach` consta de los siguientes atributos:

- `items`. Indica la colección sobre la que iterar
- `var`. Indica el nombre de la variable donde se guardará el elemento en curso.
- `varStatus`. Indica el nombre de la variable donde se guardará el estado de la iteración.

En cada iteración, la variable indicada en `var` irá tomando el valor del elemento en curso. A través de la variable indicada en `varStatus` podemos acceder a las siguientes propiedades:

- `index`. Posición del elemento en curso (comienza con 0).
- `count`. Número de iteraciones (comienza con 1).
- `first`. Valor booleano que indica si es la primera iteración.
- `last`. Valor booleano que indica si es la última iteración.

forTokens permite partir una cadena en fragmentos y recorrer cada uno de éstos, consta de los siguientes atributos:

- `items`. Cadena que se quiere *tokenizar*.
- `var`. Indica el nombre de la variable donde se guardará el fragmento en curso.
- `delims`. Cadena con todos los caracteres que actúan como delimitador.
- `varStatus`. Indica el nombre de la variable donde se guardará el estado de la iteración.

Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:forEach items="${header}" var="cabecera">
      ${cabecera} <br>
</c:forEach>
<br><hr>
<%
      String []cadenas={"uno","dos","tres"};
      pageContext.setAttribute("cadenas",cadenas);
%>
<c:forEach items="${pageScope.cadenas}" var="cadena">
      ${cadena}
</c:forEach>
<br><hr>
<c:forEach items="${param}" var="parametro">
      Nombre: ${parametro.key}
      Valor: ${parametro.value}
      <br>
</c:forEach>
```

```
<br>
<c:forTokens items="cero, uno, dos, tres, cuatro, cinco"
      var="token" varStatus="status" delims=",">
      <br> ${status.index}.- ${token}
</c:forTokens>
```

4.2.7. redirect

Redirige a la dirección especificada en el atributo `url`, y aborta el procesamiento de la página actual. Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:if test="${param.clave!='secreto'}">
      <c:redirect url="login.jsp"/>
</c:if>
```

4.2.8. Control de errores con catch

Con `<c:catch>` podemos capturar excepciones, sin que se aborte la ejecución de la página al producirse un error. En el atributo `var` indicamos el nombre de la variable donde debe guardarse la información de la excepción, podremos saber que se ha producido un error comprobando que el valor de esa variable no es nulo.

Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:catch var="error01">
      <%=Integer.parseInt(request.getParameter("parametro"))%>
</c:catch>

<c:if test="${not empty error01}">
      Se produjo un error: ${error01}
</c:if>

<br/>

<form>
      <input type="hidden" name="parametro" value="prueba"/>
      <input type="submit" value="Enviar 'prueba'"/>
</form>

<form>
      <input type="hidden" name="parametro" value="1234"/>
      <input type="submit" value="Enviar '1234'"/>
</form>

<form>
      <input type="submit" value="No enviar el parámetro"/>
</form>
```

4.3. La librería `formatting`

Contiene etiquetas que realizan diversas funciones relacionadas con formato y localización, comentaremos únicamente `formatDate` y `formatNumber`. La directiva para poder utilizarla es:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
```

4.3.1. `formatDate`

Permite formatear fechas. Tiene, entre otros, los siguientes atributos:

- `type`. Permite los valores `time` (sólo hora), `date` (sólo fecha) y `both` (ambos). El valor por defecto es `date`.
- `pattern`. Permite controlar el formato, podemos ver como estructurarlo en la clase `java.text.SimpleDateFormat` del API de Java.
- `value`. La fecha en sí o la variable que la contiene.

Ejemplo

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@  taglib  uri="http://java.sun.com/jsp/jstl/fmt"  prefix="fmt"
%>

<%
      pageContext.setAttribute("fecha",new java.util.Date());
%>

<br>

Hoy es:
<fmt:formatDate value="${pageScope.fecha}"
      pattern="dd/MM/yyyy"/>
<br>
Son las:
<fmt:formatDate value="${pageScope.fecha}" pattern="HH:mm:ss"/>
<br>
Es el <fmt:formatDate value="${pageScope.fecha}" pattern="DD"/>°
día del año
```

4.3.2 `formatNumber`.

Permite dar formato a valores numéricos. Entre otros, tiene los siguientes atributos:

- `value`. Valor numérico a formatear.
- `pattern`. Permite controlar el formato, podemos ver como estructurarlo en la clase `java.text.DecimalFormat` del API de Java.

Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@  taglib  uri="http://java.sun.com/jsp/jstl/fmt"  prefix="fmt"
%>

<c:set var="numero">123</c:set>

<br> <fmt:formatNumber value="${numero}" pattern="#"/>
<br> <fmt:formatNumber value="${numero}" pattern="00000000"/>
<br> <fmt:formatNumber value="${numero}" pattern="#.00"/>
<br> <fmt:formatNumber
            value="${numero/5}" pattern="0000000.000"/>
```

4.4. Funciones EL.

Además de las librerías, JSTL define funciones, que se insertan en las expresiones de EL. Para poder utilizarlas usaremos la siguiente directiva:

```
<%@ taglib
      uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
```

Mostraremos una tabla con algunas de ellas:

·`fn:join(String[], separador)`	Convierte todos los elementos de un *array* en una cadena, separados por la cadena especificada.
·`fn:lenght(colección o cadena):int`	Devuelve el número de elementos de una colección o el número de caracteres de una cadena.
·`fn:replace(cadena, cadAntes, cadDespues):string`	Reemplaza todas las ocurrencias de `cadAntes` por `cadDespues` en la cadena `cadena`.
·`fn:startsWith(cadena, prefijo):booleano` ·`fn:endsWith(cadena, sufijo):booleano` ·`fn:contains (cadena, subcadena):booleano` ·`fn:containsIgnoreCase (cadena, subcadena):booleano`	Funciones que nos permiten evaluar si una subcadena contiene a otra, o comienza o termina con ella.

Ejemplo:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib
      uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

<%
      String[] cadenas={"uno","dos","tres"};
      pageContext.setAttribute("cadenas",cadenas);
%>

<c:set var="cadena">
      ${fn:join(cadenas,'-')}
</c:set>
${cadena}
<br>
${fn:contains(cadena,"uno")}
<br>
${fn:replace(cadena,"uno","1")}
<br>
${fn:endsWith(cadena,"tres")}
```

5. Introducción a Tomcat

Tomcat (http://jakarta.apache.org/tomcat/) es la implementación de referencia de las tecnologías *Java Servlet* (http://java.sun.com/products/servlets) y *JavaServer Pages* (http://java.sun.com/products/jsp).
Existen varias versiones, según la versión de las especificaciones que implementan.

Servlet/JSP Spec	Tomcat version
2.4/2.0	5.x
2.3/1.2	4.x
2.2/1.1	3.x

Los ejemplos de esta documentación están probados sobre la versión 5.5.

5.1. Instalación

Para que Tomcat funcione es necesario que se encuentre instalado el JDK de Java y que exista la variable de entorno JAVA_HOME que apunte al directorio de instalación del JDK, si no se encuentra podemos definirla con `export JAVA_HOME=[ruta jdk]` en Linux y con `set JAVA_HOME=[ruta jdk]` en Windows. Para instalar Tomcat simplemente debemos descomprimir el contenido del archivo en un directorio, también podemos usar la versión con instalador para plataformas Windows.

5.2. Ejecución

Una vez instalado Tomcat, en la subcarpeta *bin* se encuentran los dos scripts para arrancar y detener el servidor: `startup.sh` y `shutdown.sh` (`startup.bat` y `shutdown.bat` en Windows).

Una vez arrancado el servidor, podemos acceder a la dirección raíz: http://localhost:8080

5.3. Directorios

En el directorio de instalación de Tomcat, encontramos entre otros:

- **bin**: Aquí encontramos los ejecutables y scripts para lanzar y detener el servidor, así como para instalar como servicio.
- **common**: Clases y ficheros jar comunes al servidor (globales para todas las aplicaciones web). En el directorio /lib encontramos ficheros jar necesarios para la compilación de servlets y JSP.
- **conf**: Archivos de configuración.
- **logs**: Ficheros log del servidor.
- **webapps**: Aquí colocaremos nuestras aplicaciones web, cada una en un directorio. Si no queremos crear una nueva, podemos crear páginas sueltas en el webapp por defecto ROOT.
- **work:** Páginas JSP compiladas y caché. Se puede eliminar el contenido si tenemos problemas para ver reflejados los cambios que realicemos a las páginas JSP.

Dentro de cada directorio dentro de **webapps**, tenemos:

- **WEB-INF**: Fichero de configuración web.xml
- **WEB-INF/classes**: Aquí colocaremos las clases Java que usemos en nuestras aplicaciones web, es importante que las clases estén contenidas en paquetes (packages).
- **WEB-INF/lib**: Ficheros jar comunes a la aplicación web.

5.4. Arranque paso a paso

- Descomprimir Tomcat en un directorio.
- Arrancar el servidor con bin/startup (.sh en Linux y .bat en Windows). Si da error, asegurar que estén definidas JAVA_HOME y CATALINA_HOME.
- Crear un archivo holamundo.jsp en webapps/ROOT/ con el contenido:

```
<html>
	<body>
		Hola, mundo
		<br>
		<% out.println("Hola, mundo"); %>
		<br>
		<%= new String("Hola, mundo") %>
		<br>
		<%
			String s="Hola, mundo";
			out.println(s);
		%>
	</body>
</html>
```

- Acceder a la página en el explorador, en la dirección http://localhost:8080/holamundo.jsp

5.5. Acceder a una clase Java desde un JSP

- Crear en webapps/ROOT/WEB-INF/classes/pruebas/ un archivo Prueba.java con el contenido:

```
package pruebas;
public class Prueba
{
	public Prueba() {}

	static public int suma(int a, int b)
	{
		return a+b;
	}
}
```

- Nos situamos en /webapps/ROOT/WEB-INF/classes y compilamos:

```
javac pruebas/Prueba.java
```

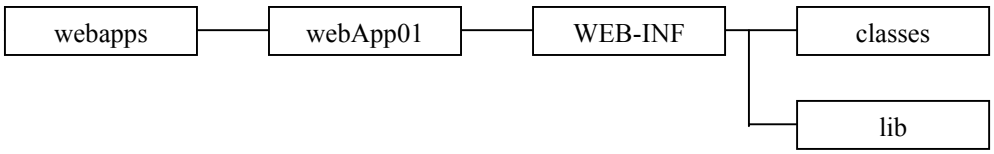
- Crear un archivo prueba.jsp en webapps/ROOT, con el contenido:

```
<%@page import="pruebas.Prueba"%>
<html>
	<body>
		5 + 6 =
		<%=Prueba.suma(5,6)%>
	</body>
</html>
```

- Acceder a la página en el explorador, en la dirección http://localhost:8080/prueba.jsp

5.6. Crear un *webapp*

La estructura típica de directorios de un webapp es la siguiente:



En el directorio WEB-INF debe aparecer el archivo web.xml, la forma de construir este archivo y sus posibilidades queda fuera del alcance de este tutorial.

En Tomcat, la forma más simple de crear un *webapp* es crear una carpeta dentro de `webapps` y una subcarpeta `WEB-INF` dentro de ésta, se reinicia Tomcat y ya podemos trabajar con la aplicación *web*.

En caso que necesitemos utilizar la librería estándar JSTL, necesitamos que los archivos `standard.jar` y `jstl.jar` se encuentren en el directorio `WEB-INF/lib` de la aplicación *web* (podremos usar JSTL en la *webapp*) o en `tomcat/shared/lib` (JSTL estará disponible para todas las *webapps)*. Estos archivos forman parte de la distribución oficial de JSTL, que podemos encontrar en http://jakarta.apache.org/taglibs/doc/standard-doc/intro.html, también podemos simplemente cogerlos del directorio `/webapps/jsp-examples/WEB-INF/lib`.

Apéndice.- Material de referencia en Internet

- Referencia rápida de JSP (2.0):
  http://java.sun.com/products/jsp/syntax/2.0/card20.pdf
- Guía de sintaxis JSP (1.2):
  http://java.sun.com/products/jsp/syntax/1.2/syntaxref12.pdf
- Referencia JSTL en formato JavaDoc
  http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html
- Referencia rápida de JSTL (Copyright 2003 Bill Siggelkow):
  http://www.jadecove.com/jstl-quick-reference.pdf
- Especificación oficial JSP (2.0):
  http://jcp.org/aboutJava/communityprocess/final/jsr152/
- API JSP 1.2 (Apartado *specifications*, 1.2 *Final release*, *Download JavaDoc*)
  http://java.sun.com/products/jsp/reference/api/index.html
- API Servlet 2.3 (Apartado *specifications, 2.3 Final release, Download JavaDoc*)
  http://java.sun.com/products/servlet/reference/api/index.html