J2EE: JSLT
Java Standard Template Library

Antonio Espín Herranz

JSLT

- La idea de utilizar las etiquetas dentro de las páginas es simplificar el código y que no parezca código java.

- Hay que tener en cuenta que las páginas JSP (la vista) puede ser realiza por diseñadores gráficos.

- Sirven para agilizar el desarrollo de páginas dinámicas.

Características de JSLT

- Las páginas JSTL son también páginas JSP. JSTL es un superconjunto de JSP.
- JSTL provee un conjunto de cinco librerías estándar:
  - Core
  - Internationalization/format
  - XML
  - SQL y
  - Funciones.
- Además JSTL define un nuevo **lenguaje de expresiones** llamado **EL**, que ha sido luego adoptado por JSP 2.0.
- Una etiqueta JSTL corresponde a una acción; llamándolas acción nos indica que añaden comportamiento dinámico.

Instalación

- En la carpeta **WEB-INF/lib** de nuestro proyecto se debe colocar los jar:
  - standar.jar
  - jslt.jar
- Los ficheros **tld** dentro de la carpeta WEB-INF.
- Y dentro del fichero WEB-INF/web.xml:

```
<jsp-config>
  <taglib>
    <taglib-uri>http://java.sun.com/jstl/core-rt</taglib-uri>
    <taglib-location>/WEB-INF/c-1.0-rt.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
    <taglib-location>/WEB-INF/c-1_0.tld</taglib-location>
  </taglib>

  <taglib>
    <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
    <taglib-location>/WEB-INF/c.tld</taglib-location>
  </taglib>
</jsp-config>
```

En Netbeans es suficiente
Con agregar los dos jar.

Soporte para EL

- El lenguaje de expresiones EL simplemente define un poderoso mecanismo para expresar expresiones simples en una sintaxis muy sencilla.
  - Es algo entre JavaScript y Perl.
  - Su combinación con las etiquetas de las 4 librerías antes mencionadas proveen mucha flexibilidad y poder para el desarrollo de páginas dinámicas.
- En EL las expresiones están delimitadas por ${ }.
- Algunos ejemplos del uso de EL son:
  - ${anExpression}
  - ${aList[4]}
  - ${aList[someVariable]} → acceso a un elemento de una colección
  - ${anObject.aProperty} → acceso a la propiedad de un objeto
  - ${anObject["aPropertyName"]} → entrada en un mapa con propiedad aPropertyName
  - ${anObject[aVariableContainingPropertyName]}

Soporte para EL

- Este lenguaje soporta operaciones aritméticas.
- Acceso a las propiedades de los beans. Estas clases java tendrían que tener métodos set / get.
- Soporta comparaciones.
- Dispone de un operador empty.

EL: Operaciones Aritméticas

EL operaciones aritméticas

Concept	EL Expression	Result
Literal	${10}	10
Addition	${10 + 10 }	20
Subtraction	${10 - 10 }	0
Multiplication	${10 * 10 }	100
Division /	${10 / 3 }	3.3333333333333335
Division DIV	${10 div 3 }	3.3333333333333335
Modulus	${10 % 10 }	1
Modulus	${10 mod 10 }	1
Division by Zero	${10 / 0 }	Infinity
Exponential	${2E2}	200.0
Unary Minus	${-10}	-10

Expresiones:

${1 + 2 * 4 - 6 / 2}

EL: Acceso a beans

```
<jsp:useBean id="person" class="modelo.Persona"></jsp:useBean>
 <table border="1">
        <tr>
          <td>${person.nombre}</td>
          <td>${person.apellidos}</td>
          <td>${person.edad}</td>
          <td>${person.dir.calle}</td>
          <td>${person.dir.numero}</td>
          <td>${person.dir.cp}</td>
          <td>${person.dir.ciudad}</td>
          <td>${person.telefonos[0]}</td>
          <td>${person.telefonos[1]}</td>
        </tr>
</table>
```

EL: operadores

- Dentro del lenguaje EL, disponemos del operador **empty**.

- Con este operador podemos preguntar si una variable está vacía o no.
  - empty "" ${empty ""}
  - empty "sometext" ${empty "sometext“}

JSLT Tag Libraries

Librería	URL	PREFIJO
core	http://java.sun.com/jsp/jstl/core	**c**
Formating i18n	http://java.sun.com/jsp/jstl/fmt	**fmt**
Funciones	http://java.sun.com/jsp/jstl/functions	**fn**
BD	http://java.sun.com/jsp/jstl/sql	**sql**

Utilizar las librerías

- La siguiente directiva ha de incluirse al comienzo de la página:
  <%@ taglib prefix="**c**" uri=http://java.sun.com/jsp/jstl/core %>

- Se podría utilizar otro prefijo pero es conveniente utilizar la c (por convención).

- El prefijo **c**, lo utilizamos siempre que queramos mostrar una etiqueta **core**.
  <**c**:out value="${anExpression}"/>

Librería de etiquetas: Functions
EL: funciones

- Se utilizan dentro de las expresiones del lenguaje EL.

- Definidas dentro de la directiva:
  - <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>

- Esta librería representa un conjunto de funciones para tratamiento de string.

Functions

- **fn:contains**(string, substring): devuelve true si la cadena contiene la subcadena.
- **fn:constainsIgnoreCase**(string, substring): idem de la anterior sin tener en cuenta las mayúsculas / minúsculas.
- **fn:endsWith**(string, sufijo): si la cadena termina en sufijo.
- **fn:escapeXml**(string): codifica los caracteres que podrían interpretados como etiquetas de xml.
- **fn:indexOf**(string, substring): Devuelve la posición que ocupa (empieza en 0) substring en string.
- **fn:join**(array, separator): Devuelve un string con los elementos del array concatenados con el separator.
- **fn:length**(string): La longitud de la cadena.

Functions

- **fn:replace**(string, before, after): Reemplaza una cadena por otra dentro del string.
- **fn:split**(string, delimitador): Devuelve un array con los elementos delimitados.
- **fn:startsWith**(string, prefijo): Devuelve true si la cadena empieza por prefijo.
- **fn:substring**(string, begin, end): Devuelve una subcadena comprendida entre begin y end.
- **fn:toLowerCase**(string): convierte a minúsculas.
- **fn:toUpperCase**(string): convierte a mayúsculas.
- **fn:trim**(string): recortar blancos.
- **fn:substringAfter**(string, substring): Devuelve la cadena que viene a continuación de la subcadena.
- **fn:substringBefore**(string, substring): Devuelve la cadena que viene antes de la subcadena.

Functions

- Ejemplo:

```
<c:set var="cadena" value="hola que tal"></c:set>
La longitud de la cadena es:
<c:out value="${fn:length(cadena)}"></c:out>
<c:set var="palabras" value="${fn:split(cadena, ' ')}"></c:set>

<c:forEach items="${palabras}" var="p">
      <p><c:out value="${p}"></c:out></p>
</c:forEach>
```

La librería de etiquetas: **Core**

- Permiten llevar a cabo las siguientes acciones:
  - Visualizar/asignar valores y manejar excepciones.
  - Control de flujo. Condiciones y Bucles.
  - Otras acciones de utilidad.
  - Tenemos que especificar la siguiente directiva.
    - **<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>**

<c:out>

- Se utiliza para visualizar el valor de una variable o de un parámetro.
  - <c:out value="${op}"></c:out>
  - <c:out value="${param.valor2}"></c:out>

  - param: representa los valores que hemos pasado por la URL.
  - param es un objeto implícito que nos sirve para recoger parámetros de un form o una Url.

<c:set>

- Esta etiqueta la utilizamos para asignar un valor a una variable.
- Por ejemplo, podemos recoger un parámetro de la URL y los asignamos a una variable.
- <c:set var="op" value="${param.valor1}"></c:set>
  - param: representa un parámetro.
  - Asignamos el valor de este a una variable llamada op.

- En la etiqueta también podemos indicar el ámbito, determinado por el atributo scope (page, session, application).
  - scope=”session”, indicaría que la definimos a nivel de sesión.

<c:remove>

- Se utiliza para eliminar una variable.
- Dentro de la etiqueta también se puede indicar el ámbito de donde queremos eliminar la variable.
- Suponiendo que tenemos definida una variable test:

```
<c:remove var="test" scope="page" />
```

<c:if>

- Para llevar a cabo condiciones simples.

- <c:if test="${ …}">
  - Dentro de la etiqueta if, el atributo test se utiliza para testear la condición.

- Ejemplo: imprime en negrita los valores pares de la colección:

```
<c:forTokens items="1,2,3,4,5,6,7,8" var="it" delims=",">
    <c:if test="${it mod 2 == 0}">
        <b><c:out value="${it}"></c:out></b><br />
    </c:if>
    <c:if test="${it mod 2 != 0}">
        <c:out value="${it}"></c:out><br />
    </c:if>
</c:forTokens>
```

<c:if>

- Dentro de if se pueden evaluar condiciones múltiples, para ello podemos utilizar **and** / **or**.
- También son válidos **&&** / **||**

```
<c:if test="${(guess < 10)  or (guess > 20)}">
    …
</c:if>
```

- Podemos usar el operador **not** o !.

<c:if>

```
<c:if test="${empty param.valor3}">
        <p>El valor del param.valor3 es vacío</p>
 </c:if>

<c:if test="${not empty param.valor3}">
        <p>El valor del param.valor3 es vacío</p>
 </c:if>
```

<c:choose>

- El switch de un lenguaje de programación se puede emular con `c:choose`:

```
<c:choose>
<c:when test="${item.type == 'book'}">
...
</c:when>

<c:when test="${item.type == 'electronics'}">
...
</c:when>

<c:when test="${item.type == 'toy'}">
...
</c:when>

<c:otherwise>
...
</c:otherwise>

</c:choose>
```

- otherwise: es equivalente al caso por defecto.

<c:forEach>

- Para iterar sobre una colección se define c:foreach.
- El atributos **items** representa la colección.
- **var**: representa la variable que va tomando los valores de los elementos de la colección.
- Se pueden especificar índice de comienzo, final e incremento con los atributos **begin**, **end** y **step**.

<c:forEach>

- **varStatus**: Es otro atributo de la etiqueta forEach y almacena información de cada iteración.
- Dispone de las siguientes **propiedades**:
  - begin: El inicio.
  - current: El elemento actual.
  - end: El útimo valor.
  - index: El elemento actual.
  - count: El número de elementos.
  - first: El primer elemento.
  - last: El último elemento.
  - step: El salto.

<c:forEach>

```
<p>Otro ejemplo de forEach</p>
 <ul>
 <c:forEach begin="1" end="20" step="5" var="i" varStatus="status">
        <li><c:out value="${i}"></c:out>
                | iteracion: <c:out value="${status.count}"></c:out>
         </li>
 </c:forEach>
 </ul>
```

- Si no se especifican items se itera de begin a end con el step que indiquemos.
- Con varStatus podemos obtener información de la iteración.

<c:forEach>

- **Ejemplo: Imprimir una colección de Personas que estaban en la session:**
- **personas**: será un parámetro que ya lo tenemos disponible.

```
<p>Tenemos <%=personas.size()%> personas</p>
<h4>ForEach</h4>
<table border='1'>
<c:forEach items="${personas}" var="persona">
    <tr>
        <td><c:out value="${persona.nombre}"></c:out></td>
        <td><c:out value="${persona.apellidos}"></c:out></td>
        <td><c:out value="${persona.edad}"></c:out></td>
    </tr>
</c:forEach>
</table>
```

<c:forTokens>

- Funcionalidad similar a StringTokenizer puede ser obtenida en JSTL con c:forTokens:

- Esta etiqueta viene bien para imprimir valores constantes, por ejemplo: cabeceras en una Tabla.

```
<table>
    <c:forTokens items="47,52,53,55,46,22,16,2" delims="," var="dailyPrice">
    <tr><td>
            <c:out value="${dailyPrice}"/>
    </td></tr>
    </c:forTokens>
</table>
```

<c:forTokens>

- Dentro de forTokens tenemos varios atributos:
  - **items**: Representan los elementos.
  - **delims**: El separador de los items.
  - **begin**, **end**: podemos indicar el índice inicial y final para recorrer una parte de los elementos. Si no se indican irán de 0 a n-1. Siendo n el número de elementos de la colección.
  - **step**: El incremento, por defecto será 1.
  - **var**: indicamos una variable que irá pasando por todos los elementos.

<c:forTokens>

Ejemplo:

```
<c:forTokens items="1,2,3,4,5,6,7,8,9,10" delims="," begin="2" end="5" step="1" var="i">
        <p><c:out value="${i}"/></p>
</c:forTokens>
```

- En este caso se itera desde índice 2 al 5, ambos inclusives.
- El paso será de 1 en 1.
- La variable i tomará los valores de los items.

Listar todos los parámetros pasados a una petición

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<body>
<head>
<title>Parameter Listing Example</title>
</head>

<br>

<b>Parameter values passed to this page for each parameter: </b>
<table border="2">
<c:forEach var="current" items="${param}">
    <tr>
    <td>
    <b><c:out value="${current.key}" /></b>
    </td>
        <c:forEach var="aVal" items="${paramValues[current.key]}">
            <td>
            <c:out value="${aVal}" />
            </td>
        </c:forEach>
    </tr>
</c:forEach>
</table>
</body>
</html>
```

Ejemplo

```
// Captura el parámetro que nos han pasado.
<c:set var="subvenciones" scope="page" value="${param.subvenciones}"></c:set>
// Recorrer una colección. Acceso a las propiedades mediante el punto.

<c:forEach var="item" items="${subvenciones}">
    <tr>
     <td><c:out value="${item.titulo}"></c:out></td>
     <td><c:out value="${item.categoriaSubvencion.categoriaSubvencion}"></c:out></td>
     <td><c:out value="${item.beneficiarios}"></c:out></td>
     <td><c:out value="${item.ambitoGeografico.ambitoGeografico}"></c:out></td>
     <td><c:out value="${item.fechaIni}"></c:out></td>
     <td><c:out value="${item.fechaFin}"></c:out></td>
     <td><c:out value="${item.provincia.provincia}"></c:out></td>
    </tr>
    </c:forEach>
```

forTokens → Se puede utilizar para pintar las cabeceras en la tabla.

<c:import>

- Está basada en la instrucción <jsp:include> de las JSPs.
- Permite incluir páginas de otros servidores y almacenar el valor leído dentro de una variable.
- Podemos importar páginas de nuestra aplicación o de fuera.

```
<c:import url="http://www.google.es"></c:import>
```

<c:param>

- Dentro de la sentencia **import** se podrían indicar parámetros a la página que queremos cargar.
- En la etiqueta param se deben indicar name y value.

```
<c:import url="miPagina.jsp">
    <c:param name="nombre" value="valor" />
</c:import>
```

<c:url>

- Con esta etiqueta podemos mostrar una URL o grabar en una etiqueta.

- Preserva la información de la sesión codificada.

```
<img src="<c:url value="/img/icono.jpg"/>"/>
```

<c:redirect>

- Ejecuta la instrucción forward. Salta a la página que indiquemos dentro del atributo url.
- <c:redirect url="otraPagina.jsp" />

<c:catch>

- Nos permite capturar excepciones dentro del JSP.
- Si se especifica la etiqueta catch, aunque hayamos definido una página de error no va a saltar.
- Dispone de un atributo var que se puede almacenar el error.
- <c:catch var="error"> … </c:catch>

<c:catch>

- Ejemplo: intentamos convertir un número que tiene letras → salta la exception.

```
<h3>Ejemplo de catch</h3>
<c:catch var="error">
    <fmt:formatNumber var="numero" value="44rr"></fmt:formatNumber>
 </c:catch>
<c:if test="${not empty error}">
    <p>Se ha producido el error: <c:out value="${error}"></c:out></p>
</c:if>
<c:if test="${empty error}">
    <p>El numero es: <c:out value="${numero}"></c:out></p>
</c:if>
```

Librería de etiquetas: **Formating**

- Nos permite opciones de formateo al mostrar resultados en el JSP.
  - Podemos formatear: números, fechas, %, etc.
  - Tenemos que añadir la siguiente directiva:
    - **<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>**

<fmt:formatNumber>

- Permite aplicar formatos a números.
- Los atributos más comunes:
  - value: El valor del número.
  - type: números, monedas o porcentajes.
  - var: Si queremos grabar el resultado en una variable.
  - scope: Ambito de la variable.
  - currencySymbol: Símbolo de la moneda.
  - groupingUsed: Si agrupa o no los dígitos antes del punto decimal.

<fmt:formatNumber>

```
<c:set var="numero" value="1200"></c:set>

 <p>Sin formatear: <c:out value="${numero}"></c:out></p>

 <p>Formateado: <fmt:formatNumber value="${numero}" type="currency"
     currencySymbol="€"></fmt:formatNumber></p>

  <p>Porcentaje: <fmt:formatNumber value="0.67"
     type="percent"></fmt:formatNumber></p>

 <p>Grouping: <fmt:formatNumber value="12345678.99"
     groupingUsed="true"></fmt:formatNumber></p>
```

Sin formatear: 1200
Formateado: €1.200,00
Porcentaje: 67%
Grouping: 12.345.678,99

<fmt:formatDate>

- Requiere una variable que represente una fecha, podemos utilizar java.util.Date.
- Atributos:
  - value: La fecha a mostrar.
  - type: horas o fechas, ambos: time, date, both.
  - dateStyle: estilo fecha: default, short, medium, long.
  - timeStyle: Idem del anterior.
  - var: Para grabar el resultado en una variable.
  - scope: Ámbito de la variable.
  - pattern: Podemos mostrar un patrón: yy, M, d, E, H, m → Clase de java SimpleDateFormat.

<fmt:formatDate>

```
<jsp:useBean id="fecha" class="java.util.Date"></jsp:useBean>

<p>Sin formatear: <c:out value="${fecha}"></c:out></p>

<p>Formateada: <fmt:formatDate value="${fecha}" type="date"
    dateStyle="medium"></fmt:formatDate>

<p>Formateada: <fmt:formatDate value="${fecha}" type="time"
    timeStyle="medium"></fmt:formatDate>

<p>Formateada: <fmt:formatDate value="${fecha}" type="both"
    pattern="dd/MM/yyyy hh:mm"></fmt:formatDate>
```

Sin formatear: Wed Dec 28 11:26:47 CET 2011
Formateada: 28-dic-2011
Formateada: 11:26:47
Formateada: 28/12/2011 11:26

<fmt:parseNumber>

- Convierte cadenas a números, si intentamos convertir cosas que no son números saltarán excepciones.

- Atributos:
  - value: El valor a convertir.
  - type: Interpretarlo como: number, currency, percent.
  - var: Almacenar el resultado en una var.
  - scope: Ámbito de la variable anterior.

<fmt:parseNumber>

- Parsea el número, lo almacena en una variable y luego lo muestra:

```
<p>Número parseado: <fmt:formatNumber var="numero"
    value="44"></fmt:formatNumber></p>
<p><c:out value="${numero}"></c:out></p>
```

Número parseado:
44

<fmt:parseDate>

- Permite parsear fechas:

- Atributos:
  - value: La fecha a parsear.
  - type: tipo: date, time, both
  - pattern: para indicar un patrón como SimpleDateFormat.
  - var: Almacenar el resultado en una variable.
  - scope: Ámbito de la variable anterior.

<fmt:parseDate>

- <fmt:parseDate var="miFecha" value="13/06/2009" type="date" pattern="dd/MM/yyyy"></fmt:parseDate>

- <c:out value="${miFecha}"></c:out>

Sat Jun 13 00:00:00 CEST 2009

Librería de etiquetas: **SQL**

- Permite el acceso a BD, para utilizar esta librería tenemos que añadir la siguiente directiva:

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

<sql:setDataSource>

- Es el equivalente a crear una conexión, tenemos que indicar los mismos parámetros que en JDBC.

```
<sql:setDataSource driver="com.mysql.jdbc.Driver"
                   url="jdbc:mysql://localhost:3306/empresa3"
                   user="root"
                   password="antonio"
                   var="bdEmpresa">
</sql:setDataSource>
```

  - En este caso creamos un datasource a la Bd de Empresa3, es necesario tener el driver de mysql.
  - En caso de requerir una conexión a otros tipos de BD, seguir los mismos criterios que en JDBC.
  - Este dataSource lo utilizaremos después para ejecutar consultas.

<sql:query>

- Con esta etiqueta podemos recuperar consultas de selección.
- Atributos:
  - sql: la consulta.
  - dataSource: la variable que representa el datasource.
  - var: La variable que almacenará el cursor (el resultado de la consulta).

<sql:query>

```
<sql:query var="clientes" dataSource="${bdEmpresa}"
sql="select nombre from clientes"></sql:query>

    <c:forEach items="${clientes.rows}" var="row">
       <p><c:out value="${row.nombre}"></c:out></p>
    </c:forEach>
```

- Imprime un listado de los nombres de los clientes.

<sql:update>

- Esta etiqueta nos permite ejecutar consultas de acción (delete, update, insert).
- Atributos:
  - sql: La consulta a ejecutar.
  - datasource: El datasource.
  - var: Una variable para almacenar las filas actualizadas.
  - scope: ámbito de la variable anterior.

<sql:update>

```
<sql:update var="n" dataSource="${bdEmpresa}"
   sql="delete from clientes where idcliente='A**'">
</sql:update>

<p>Filas borradas: <c:out
   value="${n}"></c:out></p>
```

- Elimina un cliente y después imprime el número de filas eliminadas.

<sql:param>

- Se utiliza para las consultas parametrizadas: preparedStatement.
- NO es válido para fechas, hay que utilizar <sql:paramDate>

```
<sql:update var="num">
    Insert into tabla(col1, col2) values (?,?)
    <sql:param value="${mi_var}" />
    <sql:param value="${mi_var2}" />
</sql:update>
```

<sql:paramDate>

- La misma idea que para <sql:param>.
- En este caso tenemos que indicar el **value** y **type** (time, date, timestamp).
- value, tiene que recibir una variable de tipo fecha.
- Si viene de un form habrá que pasearla antes.

<sql:transaction>

- <sql:transaction dataSource="{bdEmpresa3}>
  - Todas las sentencias insert, update, delete …
- </sql:transaction>

- Hay que indicar el datasource.

<sql:transaction>

```
<sql:transaction dataSource="${bdEmpresa}">
    <sql:update var="n" dataSource="${bdEmpresa}"
                sql="delete from clientes where
                idcliente='A**'">
    </sql:update>
    <p>Filas borradas: <c:out value="${n}"></c:out></p>
</sql:transaction>
```