Librería de etiquetas estándar de Java - JSTL

Conceptos básicos

La librería de etiquetas estándar de Java proporciona un conjunto de etiquetas JSP orientadas a realizar una serie de tareas que habitualmente se llevan a cabo en las páginas JSP, evitando en este caso tener que recurrir a scriptlets, es decir, a código Java.

Estas etiquetas pueden emplearse junto con marcos de trabajo tan populares como Struts o Spring. Están incorporadas en JSP 2.1 (J2EE 5), por lo que si se emplea un contenedor Servlet/JSP compatible con esta versión de J2EE o superior, se dispondrán de ellas por defecto. Sin embargo, si se posee la versión de JSP 1.2, será preciso descargar la librería si el contenedor no la incorpora[50]. Esta librería podrá incorporarse al contenedor Servlets/JSP o bien, incorporarla a la propia aplicación.

JSTL está organizada en cinco bibliotecas:

core
Funciones comunes de iteración sobre datos, operaciones condicionales, e importación de otras páginas.

fn
Funciones orientadas al procesamiento de cadenas.

fmt
Funciones para el formateo de textos e internacionalización.

sql
Funciones orientadas al acceso a bases de datos.

xml
Funciones para el procesamiento de documentos XML.

Estas etiquetas emplean el Lenguaje de Expresiones (EL) incorporado desde JSP 2.0 (J2EE 4). Se trata de un lenguaje que posibilita el acceso a los parámetros enviados en una petición, a las cookies o a los datos almacenados en cualquiera de los ámbitos de la aplicación. Este lenguaje soporta el uso de operadores y palabras reservadas.

El lenguaje de expresiones (EL)

Se basa en las siguientes normas:

- Las expresiones JSTL comienzan con ${ y terminan con }. Lo que hay en medio es tratado como una expresión.
- Las expresiones se componen de:
  - Identificadores. Hay once identificadores reservados que corresponden a once objetos implícitos (los veremos luego). El resto de identificadores sirven para crear variables.
  - Literales. Son números, cadenas delimitadas por comillas simples o dobles, y los valores *true, false* y *null*.
  - Operadores. Permiten comparar y operar con identificadores y literales.
  - Operadores de acceso. Se usan para referenciar propiedades de los objetos.
- Podemos usar expresiones en cualquier parte del documento, o como valores de los atributos de etiquetas JSTL, exceptuando los atributos var y scope, que no aceptan expresiones.
- En cualquier sitio donde sea valido colocar una expresión, también será válido colocar más de una. Por ejemplo: *value="Hola ${nombre} ${apellidos}"*.
- Las expresiones pueden contener operaciones aritméticas, comparaciones, operaciones booleanas, y agruparse mediante paréntesis. También pueden decidir cuando una variable existe o no (usando

50 Dependiendo de la versión que se requiera, puede descargarse de http://jstl.java.net/download.html (versión última) o http://java.sun.com/products/jsp/jstl/downloads/index.html (todas las versiones, 1.0, 1.1 y 1.2).

empty).

- Para acceder al campo de un bean java, o a un elemento de una colección (array, o Map), se usa el operador punto, o el operador corchete:

  ${bean.propiedad}
  ${map.elemento}
  ${header['User-Agent']}

  Si un campo de un bean es otro bean, podemos encadenar puntos para acceder a una propiedad del segundo bean:

  ${bean1.bean2.propiedad}
- Las expresiones pueden aparecer como parte del valor de un atributo de una etiqueta:

  <c:out value="${2+2}"/>
  <c:out value="${nombre}"/>
  <c:if test="${tabla.indice % 2 == 0}">es par</c:if>

  O independientemente junto a texto estático como el HTML:
  <input type="text" name="usuario" value="${requestScope.usuario.nombre}"/>

Los operadores de este lenguaje, así como el orden de precedencia, se corresponden con los de Java.

Acceso a datos

El operador punto (.) permite recuperar la propiedad de un objeto por su nombre. Ejemplo:

${libro.paginas}

Las propiedades se recuperan usando las convenciones de los JavaBeans. Por ejemplo, en el código anterior, se está invocando el método *getPaginas()* del objeto libro.

El operador corchete ([]) permite recuperar una propiedad con nombre o indexada por número. Ejemplo:

${libro["paginas"]}
${libro[0]}
${header['User-Agent']}

Dentro del corchete puede haber una cadena literal, una variable, o una expresión.

Operador empty

El operador empty comprueba si una colección o cadena es vacía o nula. Ejemplo:

${emptyparam.login}

Otra manera de hacerlo es usando la palabra clave *null*: *${param.login == null}*

Objetos implícitos

Ciertos objetos son automáticamente accesibles a cualquier etiqueta JSP. A través de ellos es posible a cualquier variable de los ámbitos *page*, *request*, *session*, *application*, a parámetros *HTTP*, *cookies*, valores de cabeceras, contexto de la página, y parámetros de inicialización del contexto. Todos, excepto *pageContext*, están implementados usando la clase *java.util.Map*.

Objetos implícitos	*contiene*
pageScope	Atributos de ámbito *page*.
requestScope	Atributos de ámbito *request*.
sessionScope	Atributos de ámbito *session*.
applicationScope	Atributos de ámbito *application*.
param	Parámetros del *request*.
paramValues	Parámetros del *request* como array de cadenas.
header	Cabeceras del *request* HTTP como cadenas.
headerValues	Cabeceras del *request* HTTP como array de cadenas.
cookie	Valores de las cookies recibidas en el *request*.
initParam	Parametros de inicialización de la aplicación Web.
pageContext	El objeto *PageContext* de la página actual.

Ejemplo 1

Este primer ejemplo muestra cómo puede visualizarse una cadena e imprimir el valor de un parámetro recibido en la petición. La página de la aplicación se denomina *holamundo.jsp*, y el contexto raíz de la misma será *ej1jstl*. Cuando se envíe una petición del estilo de http://localhost:9090/ej1jstl/holamundo.jsp?nombre=Pepe, la página mostrará un breve saludo seguido abajo por una frase que contiene el valor del parámetro nombre enviado junto a la petición.

El código de holamundo.jsp es el siguiente:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"    pageEncoding="UTF-8"%>
<%@include file="/WEB-INF/taglibs.jsp" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
        <title>Holamundo</title>
</head>
<body>
        <c:out value="Hola mundo"/><br>
        Su nombre es: ${param.nombre}
</body>
</html>
```

Esta página incluye en otro archivo jsp las directivas necesarias para añadir las bibliotecas de etiquetas *JSTL*. Aunque en este caso sólo se precisa la correspondiente a *core*, se ha creado el archivo *taglibs.jsp* con todas ellas con el propósito de servir para cualquier proyecto de aplicación web que vaya a hacer uso de estas etiquetas estándar. El código de taglibs.jsp se muestra a continuación:

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

Funciones en JSTL (fn)

La biblioteca **fn** proporciona 16 funciones orientadas a la manipulación de cadenas. Estas son:

- Alternar una cadena entre mayúsculas/minúsculas: *toLowerCase*, *toUpperCase*.

- Obtener una subcadena: *substring*, *substringAfter*, *substringBefore*.
- Eliminar el espacio en blanco en ambos extremos de una cadena: *trim*.
- Reemplazar caracteres en una cadena: *replace*.
- Comprobar si una cadena contiene a otra: *indexOf*, *startsWith*, *endsWith*, *contains*, *containsIgnoreCase*.
- Convertir una cadena en array (*split*), o un array en cadena (*join*).
- Codificar los caracteres especiales del XML: *escapeXml*.
- Obtener la longitud de una cadena: *length*.

La función *length* también puede aplicarse a instancias de *java.util.Collection*, caso en el cual, devolverá el número de elementos de una colección. Todas las funciones *EL* interpretan las variables no definidas o definidas con valor igual a *null*, como cadenas vacías.

fn:contains

Devuelve true si una cadena contiene a otra, false en caso contrario.

```
fn:contains(string, substring) # boolean
```

Si la cadena a buscar es vacía, siempre existirá coincidencia. Ejemplo:

```
<%-- Muestra un mensaje si la variable query contiene la cadena 'crack' --%>
<c:if test="${fn:contains(query,'crack')}">
Posiblemente está intentando piratear obras comerciales (Palabra: crack)
</c:if>
```

fn:containsIgnoreCase

Devuelve true si una cadena contiene a otra sin distinguir entre minúsculas/mayúsculas, false en caso contrario.

```
fn:containsIgnoreCase(string, substring) # boolean
```

Si la cadena a buscar es vacía, siempre existirá coincidencia. Ejemplo:

```
<%-- Muestra un mensaje si 'Alumno' es subcadena de username sin distinguir entre mayúsculas o
minúsculas. --%>
<c:set var="username" value="Alumno"/>
<c:if test="${fn:containsIgnoreCase(username, 'Alumno')}">
        ¡Hola Alumno!
</c:if>
```

fn:endsWith

Devuelve true si una cadena termina con el sufijo indicado, false en caso contrario.

```
fn:endsWith(string, suffix) # boolean
```

Si la cadena a buscar es vacía, siempre existirá coincidencia. Ejemplo:

```
<c:set var="username" value="James Stratchan"/>
<c:if test="${fn:endsWith(username, 'tratchan')}">
Hi James Stratchan!
</c:if>
```

fn:escapeXml

Codifica los caracteres que podrían ser interpretados como parte de las etiquetas XML.

fn:escapeXml(string) # String

Es equivalente a c:out con atributo xmlEscape="true".

fn:indexOf

Devuelve el índice (empezando desde cero) dentro de una cadena donde comienza otra cadena especificada.

fn:indexOf(string, substring) # int

Si la cadena a buscar es vacía devuelve cero (comienzo de la cadena). Si la cadena a buscar no se encuentra devuelve -1.

fn:join

Convierte en cadena los elementos de un array. El separador se usa para separar los elementos en la cadena resultante.

fn:join(array, separator) # String

Si el array es nulo devuelve la cadena vacía.

fn:length

Devuelve el número de caracteres en una cadena, o de elementos en una colección.

fn:length(input) # integer

Devuelve cero para cadenas vacías.

fn:replace

Recorre una cadena (inputString) reemplazando todas las ocurrencias de una cadena (beforeSubstring) por otra (afterSubstring).

fn:replace(inputString, beforeSubstring, afterSubstring) # String

El texto reemplazado no es vuelto a procesar para realizar más reemplazos. Si afterSubstring es vacía se eliminan todas las ocurrencias de beforeSubstring.

fn:split

Divide una cadena en subcadenas, usando como separador cualquier carácter de otra cadena. Los caracteres de la segunda cadena no forman parte del resultado.

fn:split(string, delimiters) # String[]

No se realiza reemplazo alguno cuando:
• La cadena a dividir es vacía (se devuelve la cadena vacía).
• La cadena de delimitadores es vacía (se devuelve la cadena original).

fn:startsWith

Comprueba si una cadena comienza por otra.

fn:startsWith(string, prefix) # boolean

Devuelve true si la cadena a buscar es vacía.

fn:substring

Devuelve un subconjunto de una cadena. Los índices comienzan en cero. La cadena resultante comienza en beginIndex y llega hasta el caracter en endIndex-1 (incluido). La longitud de la cadena es endIndex-beginIndex.

fn:substring(string, beginIndex, endIndex) # String

• Si beginIndex es menor que cero se ajusta su valor a cero.
• Si beginIndex es mayor que la longitud de la cadena se ajusta su valor a la longitud de la cadena.
• Si endIndex es menor que cero o mayor que la longitud de la cadena, se ajusta su valor a la longitud de la cadena.
• Si endIndex es menor que beginIndex se devuelve una cadena vacía.

fn:substringAfter

Devuelve la porción de una cadena que viene a continuación de cierta subcadena. La cadena devuelta comienza en el siguiente caracter tras la subcadena buscada.

fn:substringAfter(string, substring) # String

Si no se encuentra la subcadena o si esta es vacía se devuelve la cadena original.

fn:substringBefore

Devuelve la porción de una cadena que viene antes de cierta subcadena. La cadena devuelta va desde el primer carácter hasta el carácter anterior a la subcadena buscada.

fn:substringBefore(string, substring) # String

Si no se encuentra la subcadena o si esta es vacía se devuelve la cadena original.

fn:toLowerCase

Convierte todos los caracteres a minúsculas.

fn:toLowerCase(string) # String

fn:toUpperCase

Convierte todos los caracteres a mayúsculas.

fn:toUpperCase(string) # String

fn:trim

Elimina el espacio en blanco en los extremos de una cadena.

fn:trim(string) # String

Etiquetas comunes de iteración sobre datos, operaciones condicionales, e importación de otras páginas (core)

c:out

Muestra el resultado de una expresión. Su funcionalidad es equivalente a la de <%= %>. Ejemplos:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<c:out value="Hola mundo"/>
<c:out value="${variableInexistente}" default="la expresión falló"/>
<c:out value="${variableInexistente}">
la expresión falló
</c:out>
```

Para evitar que un nombre sea confundido con una expresión lo ponemos entre comillas:

```
<c:out value="${header['User-Agent']}" default="navegador no especificado"/>
```

Hemos usado comillas simples en *User-Agent* para que no se confundan con las comillas dobles que delimitan el valor del atributo, también podríamos haber usado \"User-Agent\":

c:set

Establece el valor de una variable. Ejemplos:

```
<%-- graba la variable cuatro=4 en el ámbito page --%>
<c:set var="cuatro" value="${2 + 2}"/>
<%-- equivalente al anterior pero con ámbito session --%>
<c:set var="cuatro" scope="session">
  4
</c:set>
<%-- equivalente al anterior --%>
<c:set var="cuatro" scope="session">
<c:out value="4"/>
</c:set>
```

c:remove

Elimina una variable. Cuando no se especifica ámbito, la etiqueta busca en todos los ámbitos por turno, yendo del más especifico al más general (*page*, *request*, *session*, *application*), hasta encontrar una variable con ese nombre. Si la variable no se encuentra, la etiqueta termina sin error.

c:if

Estructura condicional. En el cuerpo es posible colocar otras etiquetas, incluyendo otras <c:if>. Es útil guardar el resultado de evaluar la condición para evitar repetir los cálculos. Ejemplos:

```
<c:if test="${pageContext.request.remoteUser == 'hpetra'}">¡Hola Petra!</c:if>

<%-- establece una variable a null, que es lo mismo que no haberla definido --%>
<c:set var="nulo" value="${null}"/>

<%-- devuelve la cadena vacia --%>
<c:out value="${nulo}"/>

<%-- la condición es true en ambos casos --%>
<c:if test="${nulo == null}">la cadena es nula o no definida</c:if>

<c:if test="${nulo == variableNoDefinida}">la cadena es nula o no definida</c:if>
```

c:choose, c:when, c:otherwise

<c:choose> No tiene atributos. Acepta como hijos uno o más <c:when>

<*c:when*> tiene un atributo, la condición a evaluar.
<*c:otherwise*> no tiene atributos.

Ejecuta el cuerpo de la primera etiqueta when cuya condición evalúe a true, o el cuerpo de la etiqueta otherwise (si existe). Ejemplos:

```
<c:choose>
        <c:when test="${hour>=7 && hour<=12}"> morning </c:when>
        <c:when test="${hour>12 && hour<=17}"> afternoon </c:when>
        <c:when test="${hour>17 && hour<22}"> evening </c:when>
        <c:otherwise>night</c:otherwise>
</c:choose>
```

c:forEach

Permite iterar sobre los elementos siguientes:

- Arrays de objetos o tipos primitivos.
- Instancias de java.util.Collection, java.util.Map, java.util.Iterator, java.util.Enumeration.
- Cadenas delimitadas por comas.
- Instancias de javax.servlet.jsp.jstl.sql.Result (resultantes de una consulta SLQ con JSTL).

Es posible anidar varias etiquetas *c:forEach*. Ejemplos:

```
<c:forEach items="${informe.filas}" var="fila">
        <c:out value="${fila}"/>
</c:forEach>
Si se omite el atributo items, se itera sobre números:
<c:forEach begin="1" end="24" step="1" var="hora" varStatus="status">
        <c:out value="${hora}"/>
        <c:if test="${status.count == 19}">¡Ya han pasado 19 horas!</c:<br/>
</c:forEach>
```

c:forTokens

Permite descomponer una cadena en tokens. Analizar una cadena se llama *parsing*. Las partes en las que se descompone una cadena se llaman *tokens*.

Los delimitadores que aparecen uno tras otro son tratados como si solo existiera uno. Ejemplos:

```
<ul>
<c:forTokens items="Tinky Winki,Dipsy,Laa-Laa,Po" delims="," var="teletubbie">
        <li><c:out value="${teletubbie}"/></li>
</c:forTokens>
</ul>
```

La salida de lo anterior es:

```
<ul>
<li>Tinky Winki</li>
<li>Dipsy</li>
<li>Laa-Laa</li>
<li>Po</li>
</ul>
```

Esto devuelve *Dipsy*,*Laa-Laa* que son los tokens con índice 2, y 3:

```
<c:forTokens items="Tinky Winki,Dipsy,Laa-Laa,Po" delims="," var="teletubbie" begin="2" end="4">
        <c:out value="${teletubbie}"/>
</c:forTokens>
```

c:import, c:param

c:import
c:import proporciona toda la funcionalidad de *jsp:include* y añade otras:

- Permite incluir páginas de otros servidores.
- Permite guardar el texto leído en una variable.

c:param
c:import puede tener hijos *c:param*, que se usan para pasar parametros a la URL a recuperar. Si quieres que un bean sea accesible en otra página, seria engorroso convertirlo en una cadena de texto, es mejor usar *c:set* para guardar esta información.

Ejemplo:

Archivo: index.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<html>
<body>
<c:import url="weather.jsp">
        <c:param name="city" value="Madrid"/>
</c:import>
El tiempo meteorológico fue importado de la página weather.jsp.
</body>
</html>
```

Archivo: weather.jsp

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<c:import url="http://localhost:8080/weatherService.rest">
        <c:param name="id" value="${param.city}"/>
</c:import>
```

c:redirect

Ejecuta un *forward*. Es decir, se para la ejecución de la página actual (no se ejecuta nada de lo que este por debajo de esta instrucción) y el navegador es enviado a otra URL.

Ejemplos:

```
<c:redirect url="weatherPopUp.jsp"/>
        <c:param name="city" value="Madrid"/>
</c:redirect>
<c:redirect context="/myOtherApp" url="/somePage.jsp">
```

c:catch

Normalmente los errores en una página JSP interrumpen la ejecución de la página y se envían al contenedor, que a su vez muestra un mensaje de error. La etiqueta *c:catch* permite capturar y tratar los errores. La alternativa a *c:catch* es usar páginas de error con JSP convencional.

Si hay un error en alguna sentencia del cuerpo de la etiqueta se interrumpe la ejecución y se continua después del catch.

Aquí intentamos parsear el número FDJ34rm4 pero obviamente se produce un error:

```
<%@include file="taglibs.jsp" %>
<c:catch var="error">
<fmt:parseNumber var="total" value="FDJ34rm4"/>
```

```
<%-- nunca llega aquí --%>
...
</c:catch>
<c:if test="${not empty error}">
        No se puede parsear el dato proporcionado.
        Error details: <c:out value="${error}"/>
</c:if>
```

Para que una página redirija a otra en caso de fallar su procesamiento basta con colocar esta directiva en ella:

```
<%@ page errorPage="error.jsp" %>
Luego tendremos que crear la página de error, que deberá comenzar con esta línea:
<%@ page isErrorPage="true" %>
Un ejemplo de página de error con JSTL:
<%@ page isErrorPage="true" %>
<%@include file="taglibs.jsp" %>
</html>
<head>Incidencia en el sistema</head>
<body>
        El error fue..<br/>
<c:out value="${pageContext.exception.message}"/>
</body>
</html>
```

Internacionalización y formateo

La internacionalización (i18n) consiste en preparar una aplicación para que funcione en distintos idiomas (incluyendo formateo de fecha y moneda). Veremos aquí sólo la parte de formateo.

fmt:formatNumber

Funciona como *c:out* pero permitiendo más control sobre el formato. Su uso básico es

```
<fmt:formatNumber value="${variable}"/>
```

que hace lo mismo que c:out pero comprueba el idioma del navegador (lo lee a través de la cabecera *Accept-Language*) para mostrar correctamente la cantidad. Por ejemplo, este código muestra el idioma preferente del navegador y formatea un número:

```
<c:out value="${header['Accept-Language']}"/>
<fmt:formatNumber value="1234567.89"/>
```

el resultado para un navegador español será

es 1.234.567,89

mientras que para uno inglés será

en 1,234,567.89

En JDK 1.4, el atributo *currencyCode* tiene preferencia sobre el atributo *currencySymbol*. En versiones anteriores ocurre al contrario. Ejemplos para tipos de valores:

```
<fmt:formatNumber value="1234567.89" type="number"/>
1,234,567.89
<fmt:formatNumber value="1234567.89" type="currency"/>
$1,234,567.89
```

<fmt:formatNumber value="1234567.89" type="percent"/>
123,456,789%
Ejemplos para valores de tipo moneda:
<fmt:formatNumber value="1234567.89" type="currency"/>
$123,456,789
<fmt:setLocale value="es"/>
<fmt:formatNumber value="1234567.89" type="currency"/>
¤123,456,789
<fmt:formatNumber value="1234567.89" type="currency" currencySymbol="€"/>
€1,234,567.89
<fmt:formatNumber value="1234567.89" type="currency" currencyCode="EUR"/>
EUR1,234,567.89
<fmt:formatNumber value="1234567.89" type="currency" currencyCode="EUR" currencySymbol="€"/>
EUR1,234,567.89
Ejemplos para agrupación y número de dígitos:
<fmt:formatNumber value="1234567.89" groupingUsed="true"/>
1,234,567.89
<fmt:formatNumber value="1234567.89" groupingUsed="no"/>
1234567.89
Ejemplos para número de dígitos:
<fmt:formatNumber value="1234567.89" maxIntegerDigits="5"/>
34,567.89
<fmt:formatNumber value="1234567.89" minIntegerDigits="10"/>
0,001,234,567.89
<fmt:formatNumber value="1234567.89" maxFractionDigits="1"/>
1,234,567.9
<fmt:formatNumber value="1234567.89" minFractionDigits="5"/>
1,234,567.89000
Ejemplo de patrón:
1.2345679E6
<fmt:formatNumber value="1234567.89" pattern="##0.#####E0"/>
El formato de los patrones es el mismo que el empleado en la clase DecimalFormat [http://java.sun.com/j2se/1.4.2/docs/api/java/text/DecimalFormat.html].

fmt:formatDate

No acepta valores en el cuerpo. Requiere un atributo value con una variable que represente una fecha (conviene usar fmt:parseDate). Un modo rápido de crear una variable con una fecha es instanciar java.util.Date como bean:

<jsp:useBean id="ahora" class="java.util.Date"/>
<fmt:formatDate value="${ahora}"/>

fmt:parseNumber

Interpreta cadenas como números. Es necesario para pasar valores numéricos a algunas etiquetas como sql:param, fmt:formatNumber, y otras. Ejemplo:

java.util.Date.getTime() devuelve el número de milisegundos transcurridos desde 1 enero 1970 00:00:00 GMT, que son ahora:

<jsp:useBean id="date" class="java.util.Date" />
<fmt:formatNumber value="${date.time}" />