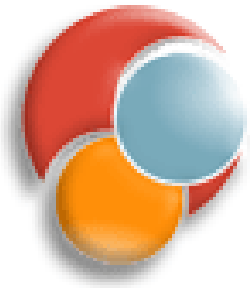


Componentes web

Sesión 7: JSTL

Puntos a tratar

- Introducción: taglibs y JSTL
- La librería Core
- La librería SQL
- La librería de internacionalización
- Otras librerías

Librerías de tags

- Una librería de tags (*taglib*) es un conjunto de etiquetas HTML personalizadas que permiten encapsular acciones mediante código Java
- Cada etiqueta define un nexo entre la página JSP donde se coloca y el código Java subyacente que se ejecutará cuando se procese dicha etiqueta

```
<%@taglib uri="http://ww.jtech.ua.es/ejemplo" prefix="ej" %>
<html>
<body>
<h1>Ejemplo de librerias de tags</h1>
<ej:mitag>Hola a todos</ej:mitag>
</body>
</html>
```

Qué es JSTL

- JSTL (*Java Server Pages Standard Tag Library*) es un grupo de librerías de tags estándar que encapsula varias funcionalidades: SQL, XML, internacionalización...
- Además, dispone de un lenguaje de expresiones que utiliza en sus etiquetas (el mismo que tiene JSP 2.0)
- Al ser estándar funciona igual en todos los servidores, y los contenedores pueden reconocerla y optimizar sus implementaciones
- Disponible a partir de servlets 2.3 y JSP 1.2
- Más información en:

http://java.sun.com/products/jsp/jstl

Librerías disponibles

Propósito general

Procesamiento XML

Multilenguaje / multiformato

Manipulación de bases de datos

Librería EL:

AREA	URI	PREFIJO
Core	http://java.sun.com/jsp/jstl/core	c
XML	http://java.sun.com/jsp/jstl/xml	x
Internacionalización (I18N)	http://java.sun.com/jsp/jstl/fmt	fmt
SQL	http://java.sun.com/jsp/jstl/sql	sql

Librería RT:

AREA	URI	PREFIJO
Core	http://java.sun.com/jsp/jstl/core_rt	c_rt
XML	http://java.sun.com/jsp/jstl/xml_rt	x_rt
Internacionalización (I18N)	http://java.sun.com/jsp/jstl/fmt_rt	fmt_rt
SQL	http://java.sun.com/jsp/jstl/sql_rt	sql_rt

Pueden utilizarse las dos versiones de cada librería en la misma página

Ejemplo de uso

- Se coloca al principio de cada página JSP la directiva *taglib* con cada librería a utilizar

```
<%@taglib uri="http://..." prefix="c" %>
```

- Dentro de la página, se pueden utilizar dichas librerías

```
<c:out ...>
```

- Se define el fichero TLD de la(s) librería(s) a utilizar en el `web.xml`:

```
<taglib>
   <taglib-uri>http://...</taglib-uri>
   <taglib-location>/WEB-INF/c.tld</taglib-location>
</taglib>
```

  - Este paso no es necesario si como *uri* en cada página ponemos la URI mostrada en la tabla anterior para cada librería

- También hay que copiar los ficheros JAR correspondientes en el directorio `WEB-INF/lib` de nuestra aplicación

Introducción

- La librería *Core* incluye tags de propósito general para:
  - Evaluación de expresiones
  - Establecimiento/obtención de valores de parámetros
  - Sentencias de control de flujo: condiciones, iteradores...
  - Funciones de acceso a URLs
- Normalmente, los tags de esta librería se utilizan con el prefijo "`c`" (o "`c_rt`" para la versión RT)

El tag *out*

- Se utiliza para evaluar el resultado de una expresión y colocarlo en la salida JSP

```
<c:out value="valor" [escapeXML="true|false"] default="valor"/>

<c:out value="valor" [escapeXML="true|false"]>
   Valor por defecto
</c:out>
```

- Ejemplo:

```
<c:out value="${datos.ciudad}" default="desconocida"/>
```

El tag *set*

- Establece el valor de un atributo en cualquier sección (*param, header, cookie...*)

```
<c:set value="valor" var="variable" [scope="..."]/>
<c:set var="variable" [scope="..."]> Valor </c:set>

<c:set value="valor" target="objeto" property="propiedad"/>
<c:set target="objeto" property="propiedad"> Valor </c:set>
```

- Ejemplo:

```
<c:set var="foo" value="2"/>
<c:set value="19" target="${persona}" property="edad"/>
<c:out value="foo vale ${foo} y edad vale ${persona.edad}"/>
```

El tag *if*

- Ejecuta su código si se cumple una condición, o guarda el valor de la comparación en una variable

```
<c:if test="condicion" var="variable" [scope="..."]/>

<c:if test="condicion" [var="variable"] [scope="..."]>
   Cuerpo
</c:if>
```

- Ejemplo:

```
<c:if test="${visitas > 1000}">
   <h1>¡Mas de 1000 visitas!</h1>
</c:if>
```

El tag *choose*

- Ejecuta una de las opciones *when* que tiene (la que cumpla la condición), o la *otherwise* si ninguna la cumple. Similar al *switch* de C o Java

```
<c:choose>
   <c:when test="condicion1"> codigo </c:when>
   <c:when test="condicion2"> codigo </c:when>
   <c:when test="condicionN"> codigo </c:when>
   <c:otherwise> codigo </c:otherwise>
</c:choose>
```

- Ejemplo:

```
<c:choose>
   <c:when test="${a > 10}"><h1>a mayor que 10</h1></c:when>
   <c:when test="${a < 0}"><h1>a menor que 0</h1></c:when>
   <c:otherwise><h1>a entre 0 y 10</h1></c:otherwise>
</c:choose>
```

El tag *forEach*

- Repite su código recorriendo un conjunto de objetos o un número de iteraciones

```
<c:forEach [var="variable"] items="conjunto"
   [varStatus="varEstado"] [begin="comienzo"] [end="final"]
   [step="incremento"]>
        codigo
</c:forEach>
<c:forEach [var="variable"] [varStatus="varEstado"]
   begin="comienzo" end="final" [step="incremento"]>
        codigo
</c:forEach>
```

- Ejemplo:

```
<c:forEach var="item" items="${cart.items}">
   <c:out value="${item.valor}"/>
</c:forEach>
```

El tag *forTokens*

- Similar a *forEach* pero recorre una cadena, separando por los delimitadores indicados
- La sintaxis es la misma que *forEach*, pero con un atributo *delim* que es obligatorio, e indica los delimitadores de la cadena
- Ejemplo:

```
<c:forTokens var="item" items="un#token otro#otromas" delim="# ">
   <c:out value="${item}"/>
</c:forTokens>
```

  - Sacaría 4 tokens: “un”, “token”, “otro” y “otromas”

Los tags *import* y *param*

- `import` se utiliza para importar el contenido de una URL
- Internamente, puede utilizar tags `param` (otros tags también pueden utilizarlo) para especificar parámetros de la URL
- Ejemplo:

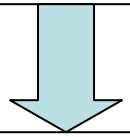
```
<c:import url="http://unapagina.com" var="varurl">
   <c:param name="id" value="12"/>
</c:import>

<c:out value="${varurl}"/>
```

  - Equivaldría a importar el contenido de `http://unapagina.com?id=12`

Ejemplo de uso

```
<html><body>
   <form action="request.jsp">
        <input type="text" name="nombre">
        <input type="text" name="descripcion">
        <input type="submit" value="Enviar">
   </form>
</body></html>
```



```
<%@taglib uri="http..." prefix="c" %>
<html><body>
   Nombre:<c:out value="${param.nombre}"/>
   <c:if test="${not empty param.descripcion}">
        Descripcion:<c:out value="${param.descripcion}"/>
   </c:if>
</body></html>
```

Introducción

- La librería *SQL* permite acceder y manipular información de bases de datos. Podremos:
  - Establecer la base de datos a la que acceder
  - Realizar consultas a bases de datos (`SELECT`)
  - Acceder a los datos de las consultas realizadas
  - Actualizar la BD (`INSERT`, `UPDATE`, `DELETE`)
  - Agrupar operaciones en una sola transacción
- Normalmente, los tags de esta librería se utilizan con el prefijo "`sql`" (o "`sql_rt`" para la versión RT)
- Las operaciones se realizan sobre un objeto `javax.sql.DataSource`, accediendo a su objeto `Connection`

El tag *setDataSource*

- Permite definir el objeto `DataSource` con el que trabajar, y dejarlo asignado a una variable

```
<sql:setDataSource {dataSource="DataSource" | url="url"
   [driver="driver"] [user="usuario"] [password="password"]}
   [var="variable"] [scope="..."]/>
```

- Podemos obtener el `DataSource` directamente de una variable precalculada, o indicando *url*, *driver*, *login* y *password*
- Ejemplo:

```
<sql:setDataSource url="jdbc:mysql//localhost/miBD"
   driver="org.gjt.mm.mysql.Driver" user="usuario1"
   password="password1" var="varBD1" />
```

El tag *query*

- Permite definir una consulta SELECT

```
<sql:query sql="consulta" var="variable"
   [dataSource="varDataSource"][maxRows="max"][startRow="inicio"]
   [scope="..."]/>

<sql:query sql="consulta" var="variable"
   [dataSource="varDataSource"][maxRows="max"][startRow="inicio"]
   [scope="..."]>
   <sql:param ...> ...
</sql:query>

<sql:query var="variable" [dataSource="varDataSource"]
   [maxRows="max"][startRow="inicio"][scope="..."]>
   Consulta
   <sql:param ...> ...
</sql:query>
```

El tag *update*

- Permite definir `INSERT`, `UPDATE` o `DELETE`

```
<sql:update sql="operacion" [var="variable"]
   [dataSource="varDataSource"][scope="..."]/>

<sql:update sql="operacion" [var="variable"]
   [dataSource="varDataSource"][scope="..."]/>
   <sql:param ...> ...
</sql:query>

<sql:update [var="variable"] [dataSource="varDataSource"]
   [scope="..."]/>
   Operacion
   <sql:param ...> ...
</sql:query>
```

El tag *transaction*

- Permite agrupar dentro varios `query` y/o `update` que forman una transacción unitaria

```
<sql:transaction dataSource="${varBD1}">
   <sql:query.../>
   <sql:update.../>
   <sql:query.../>
   <sql:query.../>
   <sql:update.../>
</sql:transaction>
```

El tag *param*

- Se utiliza internamente en `query` y `update` si la consulta o actualización necesita parámetros
- Se colocan ? en las zonas de la *query* que necesiten parámetros, y luego dentro del tag tantas etiquetas `param` como parámetros haya
- Cada etiqueta sustituye a un ?, por orden

```
<sql:query sql="SELECT * FROM datos WHERE nombre=? AND
   descripcion=?" var="var1" dataSource="${varBD1}">
   <sql:param value="pepe"/>
   <sql:param value="mi descripcion"/>
</sql:query>
```

Ejemplo de uso

```
<!-- OBTENEMOS LA CONEXIÓN -->
<sql:setDataSource url="jdbc:mysql//localhost/miBD"
   driver="org.gjt.mm.mysql.Driver" user="usuario1"
   password="password1" var="varBD1" />

<!-- REALIZAMOS LA CONSULTA/ACTUALIZACION -->
<sql:query sql="SELECT * FROM datos WHERE nombre=? AND
   descripcion=?" var="var1" dataSource="${varBD1}">
   <sql:param value="pepe"/>
   <sql:param value="mi descripcion"/>
</sql:query>

<!-- RECORREMOS LOS DATOS -->
<c:forEach var="item" items="${var1.rows}">
   <c:out value="${item.nombre}"/>
</c:forEach>
```

La librería de internacionalización

- Permite adaptar aplicaciones Web a convenciones de idioma, formato, moneda o fecha de una determinada región
- Se tienen etiquetas para la internacionalización propiamente dicha, y para dar formatos determinados a monedas, fechas, números... etc

El tag *formatNumber*

- Permite dar formato a un número e indicar con cuántos decimales queremos que se muestre

```
<fmt:formatNumber [value="numero"]
   [type="number|currency|percentage"]
   [pattern="patron"] [groupingUsed="true|false"]
   [maxIntegerDigits="cantidad"] [minIntegerDigits="cantidad"]
   [maxFractionDigits="cantidad"] [minFracionDigits="cantidad"]
   [var="nombreVariable"] [scope="ambito"]... />

<fmt:formatNumber value="${num}" maxFractionDigits="3"
   minFractionDigits="3"/>
```

El tag *formatDate*

- Permite dar formato a una fecha e indicar con qué formato queremos que se muestre

```
<fmt:formatDate [value="fecha"] [type="time|date|both"]
    [dateStyle="default|short|medium|long|full"]
    [timeStyle="default|short|medium|long|full"]
    [pattern="patron"] [timeZone="zona"]
    [var="nombreVariable"] [scope="ambito"]... />

<fmt:formatDate value="${miFecha}" pattern="dd/MM/yyyy-
   hh:mm:ss"/>
```

XML y funciones

- La librería *XML* se emplea para acceder y procesar el contenido de ficheros XML. Normalmente suele utilizarse con el prefijo “x”. En ella existen
  - *Tags principales*: para explorar el fichero XML
  - *Tags de control de flujo*
  - *Tags de transformación*: que aplican hojas XSLT
- La librería de *funciones* (`fn`) recopila una serie de funciones que introducir dentro del lenguaje de expresiones (sustituir en cadenas, concatenar, etc).

```
<%@ taglib uri="http://java.sun.com/jstl/functions" prefix="fn" %>
La cadena tiene <c:out value="${fn:length(miCadena)}"/> caracteres
```