JSTL

Índice

Copyright © 2008-2009 Depto. CCIA All rights reserved.

1. Introducción a las librerías de tags

Las **librerías de tags** (*taglibs*) son conjuntos de etiquetas HTML personalizadas que permiten encapsular determinadas acciones, mediante un código Java subyacente. Es decir, se define lo que va a ejecutar la etiqueta mediante código Java, y luego se le da un nombre a la etiqueta para llamarla desde las páginas JSP, estableciendo la relación entre el nombre de la etiqueta y el código Java que la implementa.

Por ejemplo, una página JSP que hace uso de librerías de tags podría tener este aspecto:

```
<%@ taglib uri="ejemplo" prefix="ej" %>
<html>
<body>
<h1>Ejemplo de librerias de tags</h1>
<ej:mitag>Hola a todos</ej:mitag>
<br>
<ej:otrotag/>
</body>
</html>
```

donde se utiliza una librería llamada `ejemplo`, que se simplifica con el prefijo `ej`, de forma que todos los tags de dicha librería se referencian con dicho prefijo y dos puntos, teniendo la forma `ej:tag`. Se utilizan así los tags `mitag` y `otrotag`.

JSTL (`JavaServer Pages Standard Tag Library`) es una librería de tags estándar que encapsula, en forma de tags, muchas funcionalidades comunes en aplicaciones JSP, de forma que, en lugar que tener que recurrir a varias librerías de tags de distintos distribuidores, sólo necesitaremos tener presente esta librería que, además, por el hecho de ser estándar, funciona de la misma forma en cualquier parte, y los contenedores pueden reconocerla y optimizar sus implementaciones.

JSTL permite realizar tareas como iteraciones, estructuras condicionales, tags de manipulación de documentos XML, tags SQL, etc. También introduce un lenguaje de expresiones que simplifica el desarrollo de las páginas, y proporciona un API para simplificar la configuración de los tags JSTL y el desarrollo de tags personalizados que sean conformes a las convenciones de JSTL.

Se puede emplear JSTL a partir de la versión 2.3 de servlets, y 1.2 de JSP. Podéis encontrar más información sobre JSTL en:

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

2. Uso de JSTL

JSTL contiene una gran variedad de tags que permiten hacer distintos tipos de tareas, subdivididas en áreas. Así, JSTL proporciona varios ficheros TLD, para describir cada una de las áreas que abarca, y dar así a cada área su propio espacio de nombres.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

En la siguientes tablas se muestran las áreas cubiertas por JSTL (cada una con una librería):

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

Se tienen dos versiones de JSTL, aunque la utilidad de las librerías es la misma para las dos:

- **Core** se utiliza para funciones de propósito general (manejo de expresiones, sentencias de control de flujo, etc).
- **XML** se emplea para procesamiento de ficheros XML.
- La librería de **internationalización** se usa para dar soporte a páginas multilenguaje, y a multiformatos de números, monedas, etc, en función de la región en que se tenga la aplicación.
- **SQL** sirve para acceder y manipular bases de datos relacionales.

El motivo de esta duplicación de librerías es que las librerías EL son compatibles con el lenguaje de expresiones de JSP (incluído desde JSTL 1.0 en JSTL, y desde JSP 2.0 en todas las páginas JSP), y las librerías RT son compatibles con los clásicos scriptlets <%=. . .%>. Esto quiere decir que las librerías EL admitirán lenguaje de expresiones en sus atributos, y las RT admitirán scriptlets. Se mantiene este segundo grupo de librerías por compatibilidad con versiones antiguas.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

Las URIs y prefijos que se indican en la tabla pueden emplearse (aunque no es obligatorio) para utilizar las librerías en nuestras aplicaciones.

EJEMPLO

Para utilizar, por ejemplo, los tags del área `core` en una página JSP, seguimos pasos similares a los realizados para utilizar las librerías de tags vistas anteriormente:

- Colocar al principio de cada página JSP donde vayamos a utilizar la librería `core` la línea:

```
<%@ taglib uri="/jstl-core" prefix="c" %>
```

  donde la `uri` y el `prefix` tendrán los valores que queramos darles, si bien la URI se recomienda que sea la indicada en las tablas superiores, para no tener que utilizar fichero TLD en nuestra aplicación

- Las librerías JSTL vienen en dos versiones: los ficheros TLD para la librería JSTL-EL vienen nombrados como `prefix.tld` (siendo `prefix` el prefijo para la librería), y los de la librería JSTL-RT vienen nombrados como `prefix-rt.tld`. Así, por ejemplo, para indicar en el descriptor de despliegue (`web.xml`) que vamos a utilizar la librería de `core`, pondríamos algo como:

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

  Si utilizamos las `uris` absolutas que se han indicado anteriormente, no tenemos que añadir este elemento `<taglib>`en el descriptor de despliegue. El contenedor JSP localiza automáticamente el TLD.

- Finalmente, tenemos que tener accesible una implementación de la librería que vamos a utilizar. Hay que copiar los ficheros JAR correspondientes en el directorio `WEB-INF/lib` de nuestra aplicación, si el servidor web no las tiene directamente disponibles
- Podemos utilizar las dos librerías conjuntamente, como en este ejemplo, que utiliza las dos librerías de internacionalización (prefijo `fmt`):

```
<fmt:message key="clave">
        <fmt:param value="${miParametro}"/>
        <fmt_rt:param value="<%= miParametro %>"/>
</fmt:message>
```

3. La librería Core

Los tags `core` incluyen tags de propósito general. En esta librería se tienen etiquetas para:

Copyright © 2008-2009 Depto. CCIA All rights reserved.

- Funciones de propósito general: evaluar expresiones, establecer valores de parámetros, etc.
- Funciones de control de flujo: condiciones para ejecutar unos bloques de código u otro, iteradores, etc.
- Funciones de acceso a URLs: para importar URLS en la página actual, etc.

Los tags de esta librería se presentan con el prefijo "**c**".

3.1. Tags de propósito general

out

El tag **out** evalúa el resultado de una expresión y lo pone en el objeto `JspWriter` actual. Es equivalente a la sintaxis `<%= ... %>` de JSP, y también a poner directamente una expresión **${...}** en su lugar, aunque admite algunos atributos adicionales.

SINTAXIS:

Dar el valor por defecto mediante un atributo `default`:

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

Dar el valor por defecto mediante el cuerpo del tag:

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

ATRIBUTOS:

- **value**: expresión que se tiene que evaluar.
- **escapeXML**: a `true` (valor por defecto) indica que los caracteres <, >, &, ', " que haya en la cadena resultado se deben convertir a sus códigos correspondientes (<, >, &, ', ", respectivamente).
- **default**: valor por defecto si el resultado es `null`. Se puede indicar por el atributo o por el cuerpo del tag.

EJEMPLO:

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

Sacaría el valor del campo `ciudad` del objeto `datos`, o mostraría "`desconocida`" si dicho valor es nulo.

set

El tag **set** establece el valor de un atributo en cualquier campo JSP (`page, request,`

Copyright © 2008-2009 Depto. CCIA All rights reserved.

`session, application`). Si el atributo no existe, se crea.

SINTAXIS:

Dar valor a una variable utilizando el atributo `value`:

```
<c:set value="valor" var="variable"
 [scope="page|request|session|application"]/>
```

Dar valor a una variable utilizando el cuerpo del tag:

```
<c:set var="variable"
 [scope="page|request|session|application"]>
        Valor
</c:set>
```

Dar valor a una propiedad de un objeto utilizando el atributo `value`:

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

Dar valor a una propiedad de un objeto utilizando el cuerpo del tag:

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

ATRIBUTOS:

- **value**: valor que se asigna. Podemos dar el valor con este atributo o con el cuerpo del tag.
- **var**: variable a la que se asigna el valor.
- **scope**: ámbito de la variable a la que se asigna el valor.
- **target**: objeto al que se le modifica una propiedad. Debe ser un objeto JavaBeans con una propiedad `propiedad` que pueda establecerse, o un objeto `java.util.Map`.
- **property**: propiedad a la que se le asigna valor en el objeto `target`.

EJEMPLO:

```
<c:set var="foo" value="2"/>
...
<c:out value="${foo}"/>
```

Asignaría a la variable `foo` el valor "2", y luego mostraría el valor por pantalla.

Otras Etiquetas

Existen otras etiquetas, como **remove** o **catch**, que no se comentan aquí.

3.2. Tags de control de flujo

if

Copyright © 2008-2009 Depto. CCIA All rights reserved.

El tag **if** permite ejecutar su código si se cumple la condición que contiene su atributo `test`.

SINTAXIS:

Sin cuerpo:

```
<c:if test="condicion" var="variable"
 [scope="page|request|session|application"]/>
```

Con cuerpo:

```
<c:if test="condicion" [var="variable"]
 [scope="page|request|session|application"]>
        Cuerpo
</c:if>
```

ATRIBUTOS:

- **test**: condicion que debe cumplirse para ejecutar el `if`.
- **var**: variable donde se guarda el resultado de evaluar la expresión. El tipo de esta variable debe ser `Boolean`.
- **scope**: ámbito de la variable a la que se asigna el valor de la condición.

EJEMPLO:

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

Sacaría el mensaje "`¡Mas de 1000 visitas!`" si el contador `visitas` fuese mayor que 1000.

choose

El tag **choose** permite definir varios bloques de código y ejecutar uno de ellos en función de una condición. Dentro del `choose` puede haber espacios en blanco, una o varias etiquetas **when** y cero o una etiquetas **otherwise**.

El funcionamiento es el siguiente: se ejecutará el código de la primera etiqueta **when** que cumpla la condición de su atributo **test**. Si ninguna etiqueta `when` cumple su condición, se ejecutará el código de la etiqueta **otherwise** (esta etiqueta, si aparece, debe ser la última hija de `choose`).

SINTAXIS:

```
<c:choose>
        <c:when test="condicion1">
                codigo1
        </c:when>
        <c:when test="condicion2">
                codigo2
        </c:when>
        ...
```


Copyright © 2008-2009 Depto. CCIA All rights reserved.

```
        <c:when test="condicionN">
                codigoN
        </c:when>
        <c:otherwise>
                codigo
        </c:otherwhise>
</c:choose>
```

EJEMPLO:

```
<c:choose>
        <c:when test="${a < 0}">
        <h1>a menor que 0</h1>
        </c:when>
        <c:when test="${a > 10}">
        <h1>a mayor que 10</h1>
        </c:when>
        <c:otherwise>
        <h1>a entre 1 y 10</h1>
        </c:otherwhise>
</c:choose>
```

Sacaría el mensaje "`a es menor que 0`" si la variable a es menor que 0, el mensaje "`a es mayor que 10`" si es mayor que 10, y el mensaje "`a esta entre 1 y 10`" si no se cumple ninguna de las dos anteriores.

forEach

El tag **forEach** permite repetir su código recorriendo un conjunto de objetos, o durante un número determinado de iteraciones.

SINTAXIS:

Para iterar sobre un conjunto de objetos:

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

Para iterar un determinado número de veces:

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

ATRIBUTOS:

- **var**: variable donde guardar el elemento actual que se está explorando en la iteración. El tipo de este objeto depende del tipo de conjunto que se esté recorriendo.
- **items**: conjunto de elementos que recorre la iteración. Pueden recorrerse varios tipos:
  - **Array**: tanto de tipos primitivos como de tipos complejos. Para los tipos

Copyright © 2008-2009 Depto. CCIA All rights reserved.

primitivos, cada dato se convierte en su correspondiente `wrapper` (`Integer` para `int`, `Float` para `float`, etc)
    - **java.util.Collection**: mediante el método `iterator()` se obtiene el conjunto, que se procesa en el orden que devuelve dicho método.
    - **java.util.Iterator**
    - **java.util.Enumeration**
    - **java.util.Map**:el objeto del atributo `var` es entonces de tipo `Map.Entry`, y se obtiene un `Set` con los mapeos. Llamando al método `iterator()` del mismo se obtiene el conjunto a recorrer.
    - **String**: la cadena representa un conjunto de valores separados por comas, que se van recorriendo en el orden en que están.
- **varStatus**: variable donde guardar el estado actual de la iteración. Es del tipo `javax.servlet.jsp.jstl.core.LoopTagStatus`.
- **begin**: indica el valor a partir del cual comenzar la iteración. Si se está recorriendo un conjunto de objetos, indica el índice del primer objeto a explorar (el primero es el 0), y si no, indica el valor inicial del contador. Si se indica este atributo, debe ser mayor o igual que 0.
- **end**: indica el valor donde terminar la iteración. Si se está recorriendo un conjunto de objetos, indica el índice del último objeto a explorar (inclusive), y si no, indica el valor final del contador. Si se indica este atributo, debe ser mayor o igual que `begin`.
- **step**: indica cuántas unidades incrementar el contador cada iteración, para ir de `begin` a `end`. Por defecto es 1 unidad. Si se indica este atributo, debe ser mayor o igual que 1.

EJEMPLO:

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

Muestra el valor de todos los `items`.

forTokens

El tag **forTokens** es similar al tag **foreach**, pero permite recorrer una serie de `tokens` (cadenas de caracteres), separadas por el/los delimitador(es) que se indique(n).

SINTAXIS

La sintaxis es la misma que `foreach`, salvo que se tiene un atributo **delims**, obligatorio.

ATRIBUTOS

- **var**: igual que para `foreach`

Copyright © 2008-2009 Depto. CCIA All rights reserved.

- **items**: cadena que contiene los tokens a recorrer
- **delims**:conjunto de delimitadores que se utilizan para separar los tokens de la cadena de entrada (colocados igual que los utiliza un `StringTokenizer`).
- **varStatus**: igual que para `foreach`
- **begin**: indica el índice del token a partir del cual comenzar la iteración.
- **end**: indica el índice del token donde terminar la iteración.
- **step**: igual que para `foreach`.

EJEMPLO:

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

Definimos dos separadores: el '#' y el espacio ' '. Así habrá 4 iteraciones, recorriendo los tokens "un", "token", "otro" y "otromas".

3.3. Tags de manejo de URLs

import

El tag **import** permite importar el contenido de una URL.

SINTAXIS:

Para copiar el contenido de la URL en una cadena:

```
<c:import url="url" [context="contexto"]
 [var="variable"]
 [scope="page|request|session|application"]
 [charEncoding="codificacion"]>
        cuerpo para tags "param" opcionales
</c:import>
```

Para copiar el contenido de la URL en un `Reader`:

```
<c:import url="url" [context="contexto"]
 varReader="variableReader"
 [charEncoding="codificacion"]>
        codigo para leer del Reader
</c:import>
```

ATRIBUTOS:

- **url**: URL de la que importar datos
- **context**: contexto para URLs que pertenecen a contextos distintos al actual.
- **var**: variable (`String`) donde guardar el contenido de la URL
- **varReader**: variable (`Reader`) donde guardar el contenido de la URL

Copyright © 2008-2009 Depto. CCIA All rights reserved.

- **scope**: ámbito para la variable `var`
- **charEncoding**: codificación de caracteres de la URL

EJEMPLO:

```
<c:import url="http://www.ua.es"
 var="universidad">
        <c:out value="${universidad}"/>
</c:import>
```

Obtiene y muestra el contenido de la URL indicada.

param

El tag **param** se utiliza dentro del tag **import** y de otros tags (`redirect, url`) para indicar parámetros de la URL solicitada. Dentro del tag `import` sólo se utiliza si la URL se guarda en una cadena. Para los `Readers` no se emplean parámetros.

SINTAXIS:

Sin cuerpo:

```
<c:param name="nombre" value="valor"/>
```

Con cuerpo:

```
<c:param name="nombre">
        Valor
</c:param>
```

ATRIBUTOS:

- **name**: nombre del parámetro
- **value**: valor del parámetro. Puede indicarse bien mediante este atributo, bien en el cuerpo del tag.

EJEMPLO:

```
<c:import url="http://localhost/mipagina.jsp"
 var="universidad">
        <c:param name="id" value="12"/>
</c:import>
```

Obtiene la página `mipagina.jsp?id=12` (le pasa como parámetro `id` el valor 12).

Otras Etiquetas

Existen otras etiquetas, como **url** o **redirect**, que no se comentan aquí.

3.4. Ejemplo

Vemos cómo quedaría el ejemplo visto en la sesión anterior para la librería `request` adaptado a la librería `core`. Partiendo del mismo formulario inicial:

Copyright © 2008-2009 Depto. CCIA All rights reserved.

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

Para obtener los parámetros podríamos tener una página como esta:

```
<%@ taglib uri="core" prefix="c" %>

<html>
<body>
        Nombre: <c:out value="${param.nombre}"/>
        <br>
        <c:if test="${not empty param.descripcion}">
                Descripcion: <c:out value="${param.descripcion}"/>
        </c:if>
</body>
</html>
```

Hemos utilizado en este caso, como ejemplo, los tags `out` e `if` (para comprobar si hay parámetro `descripcion`). En este último caso, utilizamos el operador `empty` en el lenguaje de expresiones, para ver si hay o no valor.

4. La librería SQL

Los tags de la librería SQL permiten acceder y manipular información de bases de datos relacionales. Vienen definidos con el prefijo "**sql**".

Con esta librería podremos:

- Establecer la base de datos a la que acceder
- Realizar consultas a bases de datos (`select`)
- Acceder a los resultados de las consultas realizadas
- Realizar actualizaciones sobre la base de datos (`insert, update, delete`)
- Agrupar operaciones en una sola transacción

Estas acciones se realizan sobre objetos de tipo `javax.sql.DataSource`, que proporciona conexiones a la fuente de datos que representa. Así, se obtiene un objeto `Connection` de dicho `DataSource`, y con él podremos ejecutar sentencias y obtener resultados. Podemos definir el `DataSource` mediante la etiqueta `setDataSource` y luego acceder a ese `DataSource` con los atributos `dataSource` de las etiquetas de la librería.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

4.1. Etiquetas de la librería

setDataSource

El tag **setDataSource** permite definir el objeto `DataSource` con el que trabajar, y dejarlo asignado en una variable.

SINTAXIS:

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

ATRIBUTOS:

- **dataSource**: objeto `DataSource` al que queremos enlazar (en caso de que esté creado de antemano).
- **url**: URL de la base de datos a acceder
- **driver**: driver con que conectar con la base de datos
- **user**: nombre de usuario con que conectar a la base de datos
- **password**: password con que conectar a la base de datos
- **var**: variable donde guardar el `DataSource` que se obtenga
- **scope**: ámbito de la variable `var`

Notar que se puede obtener el `DataSource` tanto indicándolo directamente en el atributo `dataSource` como indicando `url`, `driver`, `user` y `password` (los tres últimos opcionales).

query

El tag **query** permite definir una consulta (`select`) en una base de datos.

SINTAXIS:

Sin cuerpo:

```
<sql:query sql="consulta" var="variable"
 [scope="page|request|session|application"]
 [dataSource="DataSource"] [maxRows="max"]
 [startRow="inicio"]/>
```

Con cuerpo donde indicar parámetros de la consulta:

```
<sql:query sql="consulta" var="variable"
 [scope="page|request|session|application"]
 [dataSource="DataSource"] [maxRows="max"]
 [startRow="inicio"]>
        Campos <sql:param>
</sql:query>
```

Con cuerpo donde indicar la propia consulta y los parámetros de la consulta:

Copyright © 2008-2009 Depto. CCIA All rights reserved.

```
<sql:query var="variable"
 [scope="page|request|session|application"]
 [dataSource="DataSource"] [maxRows="max"]
 [startRow="inicio"]>
        Consulta
        Campos <sql:param>
</sql:query>
```

ATRIBUTOS:

- **sql**: consulta a realizar (en formato SQL). Puede indicarse la consulta tanto en este atributo como en el cuerpo de la etiqueta.
- **dataSource**: objeto `DataSource` asociado a la base de datos a la que se accede. Si especificamos este campo, no podemos incluir esta etiqueta dentro de una transacción (etiqueta `transaction`).
- **maxRows**: máximo número de filas que se devuelven como resultado
- **startRow**: fila a partir de la cual devolver resultados. Por defecto es la 0.
- **var**: variable donde guardar el resultado. Es de tipo `javax.servlet.jsp.jstl.sql.Result`.
- **scope**: ámbito de la variable `var`.

update

El tag **update** permite definir una actualización (`insert, update, delete`) en una base de datos.

SINTAXIS:

Sin cuerpo:

```
<sql:update sql="actualizacion"
 [dataSource="DataSource"] [var="variable"]
 [scope="page|request|session|application"]/>
```

Con cuerpo donde indicar parámetros de la actualización:

```
<sql:update sql="actualizacion"
 [dataSource="DataSource"] [var="variable"]
 [scope="page|request|session|application"]>
        Campos <sql:param>
</sql:update>
```

Con cuerpo donde indicar la propia actualización y los parámetros de la misma:

```
<sql:update [dataSource="DataSource"] [var="variable"]
 [scope="page|request|session|application"]>
        Actualización
        Campos <sql:param>
</sql:update>
```

ATRIBUTOS:

- **sql**: actualización a realizar (en formato SQL). Puede indicarse tanto en este atributo como en el cuerpo de la etiqueta.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

- **dataSource**: objeto `DataSource` asociado a la base de datos a la que se accede. Si especificamos este campo, no podemos incluir esta etiqueta dentro de una transacción (etiqueta `transaction`).
- **var**: variable donde guardar el resultado. Es de tipo `Integer` (se devuelve el número de filas afectadas por la actualización).
- **scope**: ámbito de la variable `var`.

transaction

El tag **transaction** permite agrupar dentro de él varios tags **query** y/o **update**, de forma que se define una transacción con ellos.

SINTAXIS:

```
<sql:transaction [dataSource="DataSource"]
 [isolation="nivel"]>
        Conjunto de etiquetas query y/o update
</sql:transaction>
```

ATRIBUTOS:

- **dataSource**: objeto `DataSource` asociado a la base de datos a la que se accede. Se utiliza este `DataSource` por todas las operaciones `query` y `update` que se definan dentro.
- **isolation**: nivel de aislamiento de la transacción, que puede ser `"read_committed"`, `"read_uncommitted"`, `"repeatable_read"` o `"serializable"`.

param

El tag **param** permite definir parámetros a la hora de ejecutar consultas (`query`) o actualizaciones (`update`). Así, se sustituyen las marcas "?" que pueda haber en consultas o actualizaciones, por los valores de los parámetros que se indiquen.

SINTAXIS:

Sin cuerpo:

```
<sql:param value="valor"/>
```

Con cuerpo:

```
<sql:param>
        Valor
</sql:param>
```

ATRIBUTOS:

- **value**: valor del parámetro. Puede indicarse mediante este atributo o como cuerpo del tag.

Los valores de los parámetros se sustituyen en las etiquetas en el orden en que se van definiendo. Veremos un ejemplo más adelante.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

Otras Etiquetas

Se tienen otras etiquetas, como **dateParam**, que es igual que `param` pero para parámetros de tipo `Date` (fechas). No la veremos con más detalle aquí.

EJEMPLO

Vemos un ejemplo de uso de las etiquetas explicadas.

- Creamos un `DataSource` mediante una etiqueta `setDataSource`, que se conecta a la base de datos `miBD`, de MySQL. Guardamos el `DataSource` en la variable `miDataSource`:

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

- Ejecutamos una consulta (`query`) que muestra todos los nombres de la tabla `nombres`. Guardamos el resultado en la variable `miConsulta`:

```
<sql:query var="miConsulta"
 dataSource="${miDataSource}">
        SELECT nombre FROM nombres
</sql:query>

<c:foreach var="fila" items="${miConsulta.rows}">
        <c:out value="${fila.nombre}"/><br>
</c:foreach>
```

- Ejecutamos una actualización que actualice el nombre que tenga id = 1 o id = 2:

```
<sql:update dataSource="${miDataSource}">
        UPDATE nombres SET nombre='pepe'
        WHERE id=1 OR id=2
</sql:update>
```

- Utilizamos dos etiquetas `param` para pasar como parámetros los `id` de los nombres que se tienen que actualizar:

```
<sql:update dataSource="${miDataSource}">
        UPDATE nombres SET nombre='pepe'
        WHERE id=? OR id=?
        <sql:param value="${param.id1}"/>
        <sql:param value="${param.id2}"/>
</sql:update>
```

  Se sustituiría la primera ? por el primer `param` y la segunda ? por el segundo `param`. Los dos parámetros se toman en este ejemplo de la petición `request`. Pueden tomarse de cualquier otro sitio (constantes, variables, etc).

- Vemos cómo quedarían las dos operaciones anteriores en una transacción (`transaction`):

Copyright © 2008-2009 Depto. CCIA All rights reserved.

```
<sql:transaction dataSource="${miDataSource}">

        <sql:query var="miConsulta">
                SELECT nombre FROM nombres
        </sql:query>

        <c:foreach var="fila"
         items="${miConsulta.rows}">
                <c:out value="${fila.nombre}"/><br>
        </c:foreach>

        <sql:update>
                UPDATE nombres SET nombre='pepe'
                WHERE id=? OR id=?
                <sql:param value="${param.id1}"/>
                <sql:param value="${param.id2}"/>
        </sql:update>

</sql:transaction>
```

4.2. Ejemplo

Vemos cómo quedaría el ejemplo visto en la sesión anterior para la librería `dbtags` adaptado a la librería `sql`.

```
<%@ taglib uri="sql" prefix="sql" %>
<%@ taglib uri="core" prefix="c" %>

<html>
<body>
        <sql:setDataSource url="jdbc:mysql://localhost/prueba"
         driver="org.gjt.mm.mysql.Driver"
         user="root" password="mysql" var="miDataSource"/>

        <sql:query var="miConsulta" dataSource="${miDataSource}">
                SELECT * FROM datos
        </sql:query>

        <c:forEach var="fila" items="${miConsulta.rows}">
        Nombre: <c:out value="${fila.nombre}"/><br>
        Descripcion: <c:out value="${fila.descripcion}"/><br>
        </c:forEach>
</body>
</html>
```

Utilizamos la librería `sql` y la `core` para recorrer todos los registros encontrados.

5. La librería de internacionalización

Los tags de la librería de **internacionalización** (o `I18n`) permiten adaptar aplicaciones Web a las convenciones de idioma y formato de los clientes para un determinado origen (`locale`). Los tags de esta librería se presentan con el prefijo "**fmt**".

Copyright © 2008-2009 Depto. CCIA All rights reserved.

En la librería tenemos tags para la **internacionalización** propiamente dicha, y para el **formato** de determinados elementos según la región (podremos formatear números, fechas, monedas, etc, dependiendo del idioma o región que se trate).

5.1. Etiquetas de la librería

Algunas etiquetas interesantes de esta librería son:

formatNumber

El tag **formatNumber** permite dar formato a un número, e indicar, por ejemplo, con cuántos decimales queremos que se muestre.

SINTAXIS:

```
<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"]... />
```

ATRIBUTOS:

- **value**: el valor del número que queremos formatear. Podrá ser un valor constante ("`2.34`"), o uno encapsulado en una variable en lenguaje de expresiones ("`${numero}`").
- **type**: indica el tipo de número que se quiere formatear: `number` (número genérico), `currency` (moneda) o `percentage` (para porcentajes).
- **pattern**: indica un patrón propio con el que formatear el número. Un ejemplo de uso sería "`#.###`", que dejaría 3 decimales, y al menos 1 entero.
- **groupingUsed**: a `true` indica que se separen las cifras de mil en mil, con puntos (2.032.135), y a `false` indica que no se utilicen los puntos para separar (2032135).
- **maxIntegerDigits** y **minIntegerDigits** establecen cuántas cifras tendrá como mucho y como poco (respectivamente) la parte entera.
- **maxFractionDigits** y **minFractionDigits** establecen el número de cifras como máximo y como mínimo (respectivamente) de la parte decimal.
- **var**: nombre de la variable donde guardar el número formateado (si se quiere almacenar temporalmente)
- **scope**: ámbito de la variable (como los campos `scope` vistos en otras etiquetas).
- Además, la etiqueta tiene otros atributos (para indicar monedas y otros elementos), que no se comentan aquí.

formatDate

El tag **formatDate** permite dar formato a una fecha y mostrar las partes de ella que queramos, y en el orden que queramos.

SINTAXIS:

Copyright © 2008-2009 Depto. CCIA All rights reserved.

```
<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"]... />
```

ATRIBUTOS:

- **value**: el valor de la fecha. Normalmente vendrá en una variable ("`${fecha}`").
- **type**: indica el tipo de fecha que se quiere formatear: fecha, hora o ambos.
- **dateStyle**: formatea la fecha con uno de los formatos predeterminados que tiene
- **timeStyle**: formatea la hora con uno de los formatos predeterminados que tiene
- **pattern**: indica un patrón propio con el que formatear la fecha.
- **timeZone**: cadena de texto descriptiva de la zona horaria para la que queremos formatear la fecha.
- **var**: nombre de la variable donde guardar el número formateado (si se quiere almacenar temporalmente)
- **scope**: ámbito de la variable (como los campos `scope` vistos en otras etiquetas).

5.2. Ejemplo

Vemos cómo formatear un número almacenado en la variable `num`, para que se muestre con 3 decimales siempre:

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

Formateamos ahora una fecha guardada en la variable `miFecha`, para mostrarla con:

```
día/mes/año - horas:minutos:segundos
```

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

6. La librería XML y la librería de funciones

Los tags de la librería de **XML** permiten tener acceso y procesar el contenido de documentos XML, utilizando la recomendación XPath del W3C como un lenguaje de expresiones local. Esta librería se utiliza con el prefijo "**x**".

Dentro de la librería, distinguimos entre los tags **principales** (que nos permiten explorar el contenido de un fichero XML), tags de **control de flujo** (para realizar código en función de dicho contenido), y tags de **transformación** (que permite transformar ficheros XML aplicando hojas de estilo XSLT).

Por otro lado, JSTL dispone también de un conjunto de **funciones** que pueden emplearse desde dentro del lenguaje de expresiones, y permiten sobre todo manipular cadenas, para sustituir caracteres, concatenar, etc.

Copyright © 2008-2009 Depto. CCIA All rights reserved.

Para utilizar estas funciones, cargaríamos la directiva `taglib` correspondiente:

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

Y luego utilizaríamos las funciones que haya disponibles, dentro de una expresión del lenguaje:

```
La cadena tiene <c:out value="${fn:length(miCadena)}"/> caracteres
```


Copyright © 2008-2009 Depto. CCIA All rights reserved.

Copyright © 2008-2009 Depto. CCIA All rights reserved.