Introducción a las librerías de tags

Índice

1. Conceptos básicos sobre 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`.

Para poder utilizar las taglibs necesitamos tener una versión de JSP 1.1 o superior. Veremos a continuación aspectos generales sobre las librerías de tags, para pasar después a ver dos ejemplos concretos de librerías ya hechas. Más adelante veremos cómo crear nuestras propias librerías de tags.

1.1. El gestor de tags

Cuando escribimos un tag (etiqueta) de una librería en una página JSP, un gestor de tags se pone en funcionamiento en el servidor para interactuar entre la página JSP y los objetos relacionados con dicho tag y su librería.

Existen dos interfaces principales que describen un gestor de tags, en el paquete **javax.servlet.jsp.tagext**. Cada tag que se cree implementará una de las dos:

- **Tag**: para tags simples, que no manejan el contenido de los mismos
- **BodyTag**: una extensión del anterior que permite manipular el contenido (cuerpo) de los tags

El gestor se encarga de llamar a su método **doStartTag()** cuando comienza el mismo. Este método puede devolver tres valores:

- `EVAL_BODY_INCLUDE`: explora el cuerpo del tag, para tags de tipo `Tag`
- `EVAL_BODY_TAG`: explora el cuerpo del tag, para tags del tipo `BodyTag.` Con esto podremos manipular el cuerpo del tag.
- `SKIP_BODY`: no explora el cuerpo del tag

Cuando se termina el tag, se llama al método **doEndTag()** que puede devolver:

- EVAL_PAGE: para continuar evaluando la página
- SKIP_PAGE: para no continuar evaluando la página

Estos valores devueltos permiten al contenedor JSP decidir cómo evaluar el resto de la página JSP.

Para los gestores de tipo `BodyTag`, se tienen los métodos **doInitBody()** y **doAfterBody()** para poder manipular el cuerpo del tag al procesarlo.

Los gestores de tags que se tengan deberán colocarse en el **directorio `WEB-INF/classes` o `WEB-INF/lib`** de la aplicación donde se utilicen (dependiendo de si son ficheros `.class` sueltos o están empaquetados en ficheros JAR).

1.2. El descriptor de la librería de tags

El **descriptor de la librería de tags (TLD)** es un fichero utilizado para interpretar páginas que incluyan dicha librería. Contiene directivas que describen la librería, para poderla utilizar. Es un fichero XML que mapea acciones con clases de tags.

1. Definición del fichero TLD en el descriptor de despliegue (web.xml)

Para encontrar y utilizar este fichero TLD, se utiliza una etiqueta de tipo `taglib` en el fichero descriptor de despliegue (`web.xml`) de nuestra aplicación. Así, para cada taglib que se emplee en la aplicación se tendrá un grupo de texto de este tipo en dicho fichero:

```
<taglib>
        <taglib-uri>identificador</taglib-uri>
        <taglib-location>fichero.tld</taglib-location>
</taglib>
```

- **`taglib-uri`** es un nombre identificativo de la librería
- **`taglib-location`** indica dónde se encuentra el fichero TLD (por ejemplo, en `/WEB-INF/fichero.tld`)

2. Contenido del fichero TLD

Por otra parte, el fichero TLD de la librería tiene un contenido como:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems,
 Inc.//DTD JSP Tag Library 1.1//EN"
```

```
 "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>

        <tlibversion>1.0</tlibversion>
        <jspversion>1.1</jspversion>
        <shortname>pt</shortname>
        <uri>pruebatags</uri>
        <info>Librería de prueba</info>

        <tag>
                <name>prueba</name>
                <tagclass>Prueba</tagclass>
                <bodycontent>empty</bodycontent>
                <info>Tag de prueba</info>
                <attribute>
                        <name>nombre</name>
                        <required>false</required>
                        <rtexprvalue>false</rtexprvalue>
                </attribute>
        </tag>
</taglib>
```

Se especifica al principio que es un fichero XML, y el DOCTYPE del mismo. La etiqueta raíz del documento es `taglib`, y puede tener como subetiquetas:

- **`tlibversion`**: con la versión de la librería
- **`jspversion`**: versión requerida de JSP para utilizar la librería
- **`shortname`**: nombre corto para referenciar a la librería desde JSP
- **`uri`**: identificador de la librería (el mismo que el indicado en el fichero `web.xml`)
- **`info`**: cadena descriptiva de la librería

Tras las etiquetas anteriores, tendremos una etiqueta de tipo **tag** por cada tag de la librería. Esta etiqueta puede tener como subetiquetas:

- **`name`**: nombre de la etiqueta, para llamarla desde JSP
- **`tagclass`**: clase Java que implementa la etiqueta
- **`teiclass`**: para etiquetas TEI (no las veremos) (opcional)
- **`bodycontent`**: tipo de contenido del cuerpo de la etiqueta: **empty** (sin contenido), **tagdependent** (cuerpo evaluado por la propia clase que implementa el tag), o **jsp** (cuerpo evaluado por el contenedor JSP que procesa la página)
- **`info`**: información sobre el tag (opcional)
- **`attribute`**: atributos o parámetros del tag. Puede tener como subetiquetas:
  - **`name`**: nombre del atributo
  - **`required`**: indica si el atributo es requerido (`true`) o no (`false`)
  - **`rtexprvalue`**: indica si se pueden asignar valores dinámicos, como expresiones JSP (`true`), o no (`false`). Por defecto es `false`.

1.3. Carga de taglibs en ficheros JSP

Para poder utilizar taglibs en ficheros JSP, se colocan al principio del fichero (antes de cualquier acción) directivas del tipo:

```
<%@ taglib uri="identificador" prefix="prefijo" %>
```

donde:

- **`uri`** es el identificador de la librería (indicado también en el fichero descriptor de despliegue (`web.xml`)). Normalmente cada librería tiene una URI que es la que se recomienda utilizar, y en algunos casos utilizar esa URI evita tener que disponer del fichero TLD en nuestra aplicación
- **`prefix`** es un prefijo para llamar a los tags de la librería

1.4. Ejemplo

Veremos ahora cómo utilizar una librería de tags. Supongamos que tenemos una librería llamada `prueba`, de forma que las clases Java que la implementan (los gestores de tags) están en un fichero `prueba.jar`, y el fichero descriptor es `prueba.tld`. Supongamos que el identificador (el `uri`) de la librería es `prueba`, y supongamos que la librería tiene un tag, llamado `hola`, sin cuerpo, que saca por pantalla el texto "hola a todos".

Para utilizar esta librería en una aplicación web, seguimos los pasos:

- Copiar el fichero JAR en el directorio `WEB-INF/lib` de la aplicación
- Copiar el fichero TLD en el directorio `WEB-INF` de la aplicación
- Añadir en el descriptor de despliegue (`web.xml`) el bloque `<taglib>` correspondiente para la librería:

  ```
  <taglib>
          <taglib-uri>prueba</taglib-uri>
          <taglib-location>
                  /WEB-INF/prueba.tld
          </taglib-location>
  </taglib>
  ```
- Añadir al principio de cada página JSP donde vayamos a utilizar la librería la línea:

  ```
  <%@ taglib uri="prueba" prefix="pr" %>
  ```

  donde el `prefix` puede ser el que queramos.

Con esto, por ejemplo, podemos utilizar la librería así:

```
<%@ taglib uri="prueba" prefix="pr" %>
<html>
<body>
```

```
<pr:hola/>
</body>
</html>
```

Vemos que el prefijo se utiliza para anteponerlo al nombre de cada tag de la librería, en la forma **prefijo:tag**.

2. Ejemplo de librería: request

La librería **request** es una librería desarrollada en el proyecto Jakarta (los autores de los servidores Apache y Tomcat), y que permite acceder a la información acerca de una petición HTTP realizada. De esta forma podremos acceder a los parámetros de entrada de una petición GET o POST, a las cabeceras HTTP, cookies, etc.

Se tiene información detallada sobre esta librería en:

http://jakarta.apache.org/taglibs/doc/request-doc/intro.html

2.1. Uso de la librería

Para utilizar la librería `request` en una aplicación web, seguimos los pasos:

- Copiar el fichero **TLD** (`request.tld`) en el directorio `WEB-INF` de la aplicación
- Copiar el fichero **JAR** con la implementación de los tags (`request.jar`) en el directorio `WEB-INF/lib` de la aplicación
- Añadir al descriptor de despliegue de la aplicación (`web.xml`) una marca `taglib` con información sobre la librería:

  ```
  <taglib>
          <taglib-uri>
          http://jakarta.apache.org/taglibs/request-1.0
          </taglib-uri>
          <taglib-location>
          /WEB-INF/request.tld
          </taglib-location>
  </taglib>
  ```

- Finalmente, en las páginas JSP donde vayamos a utilizar la librería, se añade al principio la línea:

  ```
  <%@ taglib
  uri="http://jakarta.apache.org/taglibs/request-1.0"
  prefix="req" %>
  ```

  donde el prefijo puede ser el que queramos

2.2. Algunos tags de la librería

Veremos ahora algunos de los tags con los que cuenta esta librería. Para los ejemplos que se verán, suponemos que se ha indicado un prefijo (`prefix`) "req":

1. Información de la petición

request

Para obtener información sobre la petición HTTP.

ATRIBUTOS

- **id**: permite luego obtener los datos de la petición con el mismo `id`. Los datos de la petición se obtienen mediante tags `jsp:getProperty` donde:
  - como parámetro `name` se pasa el `id` asignado al tag
  - como parámetro `property` se pasa la propiedad que se quiere obtener, que por ejemplo puede ser:
    - `remoteUser`: obtiene el login del usuario que realizó la petición
    - `contentLength`: obtiene la longitud en bytes de la petición
    - `remoteHost`: obtiene el nombre del cliente que realizó la petición, o la IP si no puede determinarse el nombre
    - ...etc.

EJEMPLO

```
<req:request id="req">
        Tamaño peticion:
        <jsp:getProperty name="req"
         property="contentLength"/>
        <br>
        Direccion cliente:
        <jsp:getProperty name="req"
         property="remoteHost"/>
</req:request>
```

2. Toma de parámetros

parameter

Obtiene el valor de un parámetro determinado. No tiene cuerpo.

ATRIBUTOS

- **name**: nombre del parámetro del que se quiere su valor.

EJEMPLO

```
<req:parameter name="param1"/>
```

parameters

Recorre toda la lista de parámetros

ATRIBUTOS

- **name**: para buscar los valores para un parámetro de nombre determinado. Este atributo es opcional.
- **id**: permite luego obtener los datos de la petición con el mismo `id`. Los datos de la petición se obtienen mediante tags `jsp:getProperty` donde:
  - como parámetro `name` se pasa el `id` asignado al tag
  - como parámetro `property` se pasa la propiedad que se quiere obtener, que pueden ser:
    - `name`: obtiene el nombre del parámetro
    - `value`: obtiene el valor del parámetro

EJEMPLO

```
<req:parameters id="id1">
        Nombre:
        <jsp:getProperty name="id1"
         property="name"/>
        <br>
        Valor:
        <jsp:getProperty name="id1"
         property="value"/>
        <br>
</req:parameters>
<req:parameters id="id1" name="param1">
        Valor:
        <jsp:getProperty name="id1"
         property="value"/>
        <br>
</req:parameters>
```

parameterValues

Recorre la lista de valores para un parámetro con múltiples valores. Debe estar incluido dentro de un tag `parameters`, recorriendo así los valores del parámetro que se esté explorando.

ATRIBUTOS

- **id**: permite luego obtener los datos de la petición con el mismo `id`. Los datos de la petición se obtienen mediante tags `jsp:getProperty` donde:
  - como parámetro `name` se pasa el `id` asignado al tag
  - como parámetro `property` se pasa `value`, que obtiene el valor del parámetro

EJEMPLO

```
<req parameters id="id1" name="param1">
        <req:parameterValues id="id2">
                Valor:
                <jsp:getProperty name="id2"
                 property="value"/>
                <br>
        </req:parameterValues>
</req:parameters>
```

equalsParameter

Comprueba si el valor del parámetro coincide con un determinado valor

ATRIBUTOS

- **name**: nombre del parámetro que se compara
- **match**: cadena con la que tiene que coincidir
- **value** indica si deben coincidir (`true`) o no (`false`) (por defecto es `true`). Este parámetro es opcional.
- **ignoreCase**: si deben coincidir mayúsculas y minúsculas (`false`, valor por defecto) o no (`true`). Este parámetro es opcional.

EJEMPLO

```
<req:equalsParameter name="param1"
 match="hola" value="false">
        El parametro no coincide
</req:equalsParameter>
<req:equalsParameter name="param1"
 match="hola">
        El parametro si coincide
</req:equalsParameter>
```

existsParameter

Comprueba si existe un determinado parámetro

ATRIBUTOS

- **name**: nombre del parámetro que se busca
- **value**: indica si debe existir (`true`) o no (`false`) (por defecto es `true`). Este parámetro es opcional

EJEMPLO

```
<req:existsParameter name="param1"
 value="false">
        El parametro no existe
</req:existsParameter>
```

3. Cookies

cookie

Obtiene el valor de una cookie determinada. Similar a `parameter.`

```
<req:cookie name="nombre"/>
```

cookies

Recorre toda la lista de cookies

ATRIBUTOS

- **name**: para buscar los valores para una cookie de nombre determinado. Este parámetro es opcional.
- **id**: permite luego obtener los datos de la petición con el mismo `id`. Los datos de la petición se obtienen mediante tags `jsp:getProperty` donde:
  - como parámetro `name` se pasa el `id` asignado al tag
  - como parámetro `property` se pasa la propiedad que se quiere obtener, que pueden ser, por ejemplo:
    - `name`: obtiene el nombre de la cookie
    - `value`: obtiene el valor de la cookie
    - `maxAge`: obtiene el tiempo de expiración

EJEMPLO

```
<req:cookies id="id1">
        Nombre:
        <jsp:getProperty name="id1"
         property="name"/>
        <br>
        Valor:
        <jsp:getProperty name="id1"
         property="value"/>
        <br>
        Max tiempo:
        <jsp:getProperty name="id1"
         property="maxAge"/>
        <br>
</req:cookies>
```

equalsCookie

Obtiene si el valor de una cookie coincide con una cadena dada (similar a `equalsParameter`)

existsCookie

Obtiene si existe una cookie dada (similar a `existsParameter`)

4. Cabeceras

header

Obtiene el valor de una cabecera determinada (similar a `cookie` o a `parameter`)

```
<req:header name="nombre"/>
```

headerValues

Obtiene todos los valores de una cabecera con múltiples valores (similar a `parameterValues`)

headers

Recorre toda la lista de cabeceras. Similar a `parameters`.

equalsHeader

Obtiene si el valor de una cabecera coincide con una cadena dada (similar a `equalsParameter`)

existsHeader

Obtiene si existe una cabecera dada (similar a `existsParameter`)

2.3. Ejemplo de uso

Dado el siguiente formulario `index.html`:

```
<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>
```

Al validarlo cargamos la página `request.jsp`, donde tenemos un ejemplo de cómo obtener los valores de los parámetros del formulario:

```
<%@ taglib
uri="http://jakarta.apache.org/taglibs/request-1.0"
```

```
prefix="req" %>

<html>
<body>
        Nombre:
        <req:parameter name="nombre"/>
        <br>
        <req:existsParameter name="descripcion" value="true">
                Descripcion:
                <req:parameter name="descripcion"/>
        </req:existsParameter>
        <br>
</body>
</html>
```

Para probarlo, deberíamos utilizar los ficheros TLD y JAR de la librería `request`, copiados ya en `WEB-INF` y `WEB-INF/lib`, respectivamente. Podemos probar el ejemplo con:

```
http://localhost:8080/apptaglibs/indexrequest.html
```

3. Ejemplo de librería: dbtags

La librería **dbtags** es otra librería desarrollada en el proyecto Jakarta que permite manipular bases de datos SQL. Requiere al menos la versión 1.2 de JSP (aunque en algunos servidores como Tomcat funciona con la 1.1). También necesita objetos `DataSource`, que no son parte de JDK estándar (SE). Para poderlos utilizar, tenemos que trabajar también con JDK Enterprise Edition, o bien utilizar la API para JDBC que corresponda.

Se tiene información detallada sobre esta librería en:

http://jakarta.apache.org/taglibs/doc/dbtags-doc/index.html

3.1. Uso de la librería

Para utilizar la librería `dbtags` en una aplicación web, seguimos los pasos:

- Copiar el fichero **TLD** (`dbtags.tld`) en el directorio `WEB-INF` de la aplicación
- Copiar el fichero **JAR** con la implementación de los tags (`dbtags.jar`) en el directorio `WEB-INF/lib` de la aplicación
- Añadir al descriptor de despliegue de la aplicación (`web.xml`) una marca `taglib` con información sobre la librería:

```
<taglib>
        <taglib-uri>
        http://jakarta.apache.org/taglibs/dbtags
        </taglib-uri>
        <taglib-location>
        /WEB-INF/dbtags.tld
        </taglib-location>
```

```
</taglib>
```

- Finalmente, en las páginas JSP donde vayamos a utilizar la librería, se añade al principio la línea:

```
<%@ taglib
uri="http://jakarta.apache.org/taglibs/dbtags"
prefix="sql" %>
```

  donde el prefijo puede ser el que queramos

3.2. Algunos tags de la librería

Veremos ahora algunos de los tags con los que cuenta esta librería. Para los ejemplos que se verán, suponemos que se ha indicado un prefijo (`prefix`) "sql":

1. Conexión / Desconexión a la base de datos

connection

Este tag permite conectar a una base de datos, bien a través de una URL, o un objeto `DataSource`. Veremos el primer caso, por ser más común.

ATRIBUTOS

- **id**: para identificar la conexión.

Luego, dentro, se tienen cuatro `subtags`:

- **url**: indica la URL de la base de datos a la que conectar. En este caso, una base de datos MySQL llamada `prueba`. Es el único `subtag` que hay que indicar obligatoriamente
- **driver**: driver para conectar con la base de datos (opcional)
- **userId**: login del usuario que conecta (opcional)
- **password**: password del usuario que conecta (opcional)

EJEMPLO

```
<sql:connection id="con1">
        <sql:url>
        jdbc:mysql://localhost/prueba
        </sql:url>
        <sql:driver>
        org.gjt.mm.mysql.Driver
        </sql:driver>
        <sql:userId>root</sql:userId>
        <sql:password>mysql</sql:password>
</sql:connection>
```

closeConnection

Desconecta de una base de datos

ATRIBUTOS

- **conn** el `id` de la conexión que se hubiera abierto y queremos cerrar.

EJEMPLO

```
<sql:closeConnection conn="con1"/>
```

2. Sentencias SQL

statement

Este tag permite crear una sentencia SQL para lanzarla a la base de datos.

ATRIBUTOS

- **id**: para identificar la sentencia SQL
- **conn**: el `id` de la conexión (abierta previamente) con que nos conectamos a la base de datos.

Luego, dentro, se tienen los `subtags`:

- **query**: para indicar la query SQL que se va a ejecutar
- **execute**: para ejecutar la sentencia (caso de sentencias INSERT, UPDATE, DELETE)
- **resultSet**: para obtener y explorar los resultados de una sentencia SELECT. El contenido del tag `resultSet` se ejecuta una vez para cada registro que se haya obtenido con el SELECT. Podemos utilizar en este bucle los tags:
  - **getColumn**: permite obtener los valores de cada columna de un registro, bien por el nombre de la columna (atributo `colName`), bien por la posición (atributo `position`). Existen tags similares, como `getNumber`, `getTime`, etc, para obtener columnas con un tipo de datos específico.
  - **wasEmpty**: acciones a realizar si la consulta SELECT no devuelve ningún registro. Es una etiqueta relacionada con `resultset`, aunque se utiliza fuera de esta etiqueta.
  - **wasNotEmpty**: acciones a realizar si la consulta SELECT devuelve algún registro. Es una etiqueta relacionada con `resultset`, aunque se utiliza fuera de esta etiqueta.

EJEMPLO

```
<sql:statement id="s1" conn="con1">
        <sql:query>
        INSERT INTO datos(nombre, desc)
        VALUES('nombre1','descripcion1')
        </sql:query>
        <sql:execute/>
</sql:statement>
<sql:statement id="s1" conn="con1">
        <sql:query>
```

```
                SELECT * FROM datos
        </sql:query>
        <sql:resultSet id="rs1">
                Nombre:
                <sql:getColumn colName="id"/>
                <br>
                Descripcion:
                <sql:getColumn position="2"/>
                <br>
        </sql:resultSet>
        <sql:wasEmpty>
                No hay resultados que mostrar
        </sql:wasEmpty>
        <sql:wasNotEmpty>
                Hay resultados que mostrar
        </sql:wasNotEmpty>
</sql:statement>
```

3.3. Ejemplo de uso

El siguiente ejemplo completo, tomando los fragmentos anteriores, se conecta a una base de datos MySQL para obtener todos los nombres y descripciones de su tabla `datos`:

```
<%@ taglib
uri="http://jakarta.apache.org/taglibs/dbtags"
prefix="sql" %>

<html>
<body>
        <sql:connection id="con1">
                <sql:url>
                jdbc:mysql://localhost/prueba
                </sql:url>
                <sql:driver>
                org.gjt.mm.mysql.Driver
                </sql:driver>
                <sql:userId>root</sql:userId>
                <sql:password>mysql</sql:password>
        </sql:connection>

        <sql:statement id="s1" conn="con1">
                <sql:query>
                        SELECT * FROM datos
                </sql:query>
                <sql:resultSet id="rs1">
                        Nombre:
                        <sql:getColumn colName="nombre"/>
                        <br>
                        Descripcion:
                        <sql:getColumn position="2"/>
                        <br>
                </sql:resultSet>
                <sql:wasEmpty>
```

```
                        No hay resultados que mostrar
                </sql:wasEmpty>
        </sql:statement>

        <sql:closeConnection conn="con1"/>
</body>
</html>
```

Para probar el ejemplo (una vez instalada la base de datos y establecida la contraseña de acceso al servidor), cargaríamos la página JSP con:

```
http://localhost:8080/apptaglibs/dbtags.jsp
```

4. Otras consideraciones acerca de las librerías de tags

4.1. Uso de los tags

Se pueden colocar tags en varios lugares de nuestra página JSP. Podemos por ejemplo utilizar un tag `request` para obtener un valor de parámetro que utilizar en la sentencia SQL de un `dbtag`:

```
<sql:statement id="s1" conn="con1">
        <sql:query>
                SELECT * FROM datos WHERE
                nombre = '<req:parameter name="nombre"/>'
        </sql:query>
        <sql:execute/>
</sql:statement>
```

O establecer un enlace en función de un parámetro:

```
<a href="http://<req:parameter name="url"/>">Enlace dinamico</a>
```

4.2. Comunicación entre JSP y las taglibs

En ocasiones nos puede interesar comunicar código JSP con código de taglibs: obtener dentro de JSP el valor de una taglib, o al revés, por ejemplo.

Si queremos utilizar código JSP en una taglib, podemos hacerlo sin problemas. Por ejemplo, el siguiente código obtiene las 3 primeras columnas de cada registro de una SELECT:

```
<sql:statement id="s1" conn="con1">
        <sql:query>
                SELECT * FROM datos
        </sql:query>
        <sql:resultSet id="rs1">
                <% for (int i = 1; i <= 3; i++) { %>
                Valor:<sql:getColumn position="<%=i%>"/>
                <br>
                <% } %>
```

```
        </sql:resultSet>
</sql:statement>
```

NOTA: algunos atributos de ciertas etiquetas no admiten que se utilicen `scriptlets` en ellos.

El caso contrario, obtener valores de taglibs dentro de código JSP, es más complejo de tratar. Algunas etiquetas instancian código Java que es directamente accesible desde JSP, con el nombre que se les da en la etiqueta. Por ejemplo, para saber si una conexión está cerrada, podemos hacerlo con:

```
<sql:connection id="con1">
        <sql:url>jdbc:mysql://localhost/prueba</sql:url>
        <sql:driver>org.gjt.mm.mysql.Driver</sql:driver>
</sql:connection>
...
<% if (con1.getClosed()) { %>
        La conexion esta cerrada
<% } %>
```

En otros casos la comunicación no es tan directa, y hay que ver en la documentación del tag en cuestión cómo podemos acceder a su valor. Por ejemplo, para la etiqueta `getColumn` , podemos utilizar un atributo `to` para indicar un nombre de variable donde guardar su valor. Luego, desde JSP, utilizando el método `getAttribute()` del objeto `pageContext` podemos acceder a dicho valor. Por ejemplo:

```
<sql:getColumn colName="nombre" to="nombrePersona"/>
...
<% if (pageContext.getAttribute("nombrePersona").equals("Pepe")) { %>
        Hola Pepe
<% } %>
```

4.3. Más información

Podemos encontrar información detallada sobre las librerías de tags del proyecto Jakarta en:

http://jakarta.apache.org/taglibs/index.html