Creación de librerías de tags

Índice

Veremos en esta sesión cómo podemos definir nuestros propios tags y formar con ellos una librería, tratando distintos tipos de tags (aunque no todos) los que se pueden hacer.

1. Consideraciones previas

Para ello iremos incorporando tags a una librería nueva, que llamaremos **pruebatags**. Antes de comenzar a ver cómo crear tags, tendremos en cuenta las siguientes consideraciones:

- Para definir los tags haremos clases Java, que contendrán el código de cada tag. Luego desde las páginas JSP llamaremos a los tags (con lo que se ejecutará el código de las clases Java que los implementan). Colocaremos todas las clases de nuestra librería de tags en el paquete **pruebatags**.
- Utilizaremos los paquetes **javax.servlet.jsp** y **javax.servlet.jsp.tagext** principalmente para definir las librerías de tags. Para poder acceder a ellos deberemos tener en el classpath el fichero JAR correspondiente (el fichero **j2ee.jar** que viene con J2EE, o algún fichero específico del servidor que incorpore estas librerías, como por ejemplo `servlet.jar` en Tomcat 4, o `jsp-api.jar` en Tomcat 5).
- Emplearemos el servidor **Apache Tomcat** para probar los tags. Crearemos un directorio **tags** a partir del raíz `webapps`, y en él definiremos el directorio `WEB-INF`, con el fichero **web.xml**:

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//
 DTD Web Application 2.2//EN"
 "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
        <display-name>Pruebas de Tags</display-name>
        <description>
                Ejemplos de Creacion de Tags
        </description>

        <taglib>
                <taglib-uri>pruebatags</taglib-uri>
                <taglib-location>
                /WEB-INF/pruebatags.tld
                </taglib-location>
        </taglib>
</web-app>
```

donde indicamos que se va a utilizar la librería `pruebatags`, cuyo fichero descriptor (`pruebatags.tld`) está en `WEB-INF` (lo añadiremos más adelante).

2. Tags simples

Como ejemplo de tag simple haremos un tag que muestre la fecha actual. Veremos los pasos que se siguen:

1. Creación de la clase que implementa el tag

Llamaremos **FechaActual** a la clase que implementa al tag.

```
package pruebatags;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

import java.util.Calendar;

public class FechaActual implements Tag
{
        private PageContext contexto; // Contexto del tag
        private Tag padre;                // Tag padre del actual

        // Metodo llamado cuando se comienza el tag
        public int doStartTag() throws JspException
        {
                return SKIP_BODY;
        }

        // Metodo llamado cuando se termina el tag
        public int doEndTag() throws JspException
        {
                try
                {
                        // Mostramos la fecha y hora actuales
                        Calendar c=Calendar.getInstance();
                        contexto.getOut().write(
                                c.get(Calendar.DAY_OF_MONTH) +
                                "/" +
                                (c.get(Calendar.MONTH)+1) +
                                "/" +
                                c.get(Calendar.YEAR) +
                                " - " +
                                c.get(Calendar.HOUR_OF_DAY) +
                                ":" +
                                c.get(Calendar.MINUTE) +
                                ":" +
                                c.get(Calendar.SECOND));
                } catch(java.io.IOException e) {
                        throw new JspException("Error");
                }
                return EVAL_PAGE;
        }

        // Metodo de limpieza
        public void release() {}
```

```
        // Metodo para asignar el contexto
        public void setPageContext(final PageContext contexto)
        {
                this.contexto = contexto;
        }

        // Metodo para asignar el tag padre
        public void setParent(final Tag padre)
        {
                this.padre = padre;
        }

        // Metodo para obtener el padre
        public Tag getParent()
        {
                return padre;
        }
}
```

- Primero colocamos el nombre del paquete, los paquetes que necesitamos y definimos la clase. El paquete **java.util.Calendar** lo utilizamos para obtener la fecha y hora actuales. Los otros dos paquetes son los que se han dicho que se emplean para crear librerías de tags. La clase implementa la interfaz **javax.servlet.jsp.tagext.Tag**, con lo que hay que definir todos sus métodos.
- Después creamos dos campos: uno para guardar el contexto del tag (campo `contexto`), y otro para guardar el tag padre del actual (campo `padre`).
- El método **doStartTag()** se llama cuando se inicia el tag. En este caso se devuelve **SKIP_BODY** para que no se evalúe el cuerpo del tag. En caso de que queramos evaluarlo, se debe devolver **EVAL_BODY_INCLUDE**.
- El método **doEndTag()** se llama cuando se termina el tag. Aquí se muestra por la salida del contexto la fecha y hora actuales (usando un objeto `java.util.Calendar`), y se devuelve **EVAL_PAGE** para que se siga evaluando el resto de la página. Si no queremos que se siga evaluando, se devuelve **SKIP_PAGE**.
- El resto de métodos es necesario ponerlos, al formar parte de la interfaz `Tag`. Con **release()** se pueden liberar recursos asociados al tag. Con **setPageContext()** se establece el contexto para el tag, y con **setParent()** y **getParent()** se establece y obtiene el tag padre del actual. Estos métodos se llaman cuando se utiliza el tag. No tenemos que preocuparnos de llamarlos nosotros, sólo dejarlos definidos para que hagan lo correcto (establezcan el contexto y el tag padre).

Una vez que tengamos la clase definida, la **compilamos**.

2. Crear el fichero TLD

Creamos un fichero de texto con extensión **.tld**, que llamaremos por ejemplo **pruebatags.tld**, y que contendrá la descripción de la librería de tags. Su contenido es:

```
<?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>fechaactual</name>
                <tagclass>pruebatags.FechaActual</tagclass>
                <bodycontent>empty</bodycontent>
                <info>Muestra la fecha y hora actual</info>
        </tag>

</taglib>
```

La sintaxis del documento TLD se explicó anteriormente, viendo otras librerías de tags. Hemos creado una librería llamada `pruebatags`, donde tenemos un tag, llamado `fechaactual`, implementado por la clase `pruebatags.FechaActual`.

La estructura de `tag` la iremos repitiendo luego para cada tag de la librería que vayamos creando. Nos queda llevar este fichero TLD al directorio `WEB-INF` del directorio `tags` donde estamos creando la librería.

3. Crear un JAR con la librería

Podemos crear un fichero JAR (**pruebatags.jar** por ejemplo), que contenga los ficheros `.class` con las clases que implementen los tags (en este caso sólo está la clase `FechaActual`). Luego copiamos dicho fichero JAR en el directorio `WEB-INF/lib` de nuestro directorio `tags`.

También podemos no crear el JAR, y colocar los ficheros `.class` en el directorio `WEB-INF/classes` (quedando así en la carpeta `WEB-INF/classes/pruebatags`). La primera opción es más recomendable si queremos llevar la librería a otras máquinas.

4. Probar el tag

Finalmente, probamos el funcionamiento del tag.

```
<%@ taglib uri="pruebatags" prefix="pt"%>

<HTML>
<BODY >
        La fecha actual es:
        <pt:fechaactual/>
</BODY>
```

```
</HTML>
```

3. Tags con parámetros

Como ejemplo de tag parametrizado vamos a crear un tag que salude al nombre que se le pase como parámetro:

1. Creación de la clase que implementa el tag

Llamaremos **Saludo** a la clase que implementa al tag.

```
package pruebatags;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class Saludo implements Tag
{
        private PageContext contexto; // Contexto del tag
        private Tag padre;                  // Tag padre del actual
        String nombre = "";         // Nombre a saludar

        // Metodo llamado cuando se comienza el tag
        public int doStartTag() throws JspException
        {
                return SKIP_BODY;
        }

        // Metodo llamado cuando se termina el tag
        public int doEndTag() throws JspException
        {
                try
                {
                        // Mostramos el saludo
                        contexto.getOut().write("¡Hola " +
                                                nombre + "!");
                } catch(java.io.IOException e) {
                        throw new JspException("Error");
                }
                return EVAL_PAGE;
        }

        // Metodo de limpieza
        public void release() {}

        // Metodo para asignar el contexto
        public void setPageContext(final PageContext contexto)
        {
                this.contexto = contexto;
        }

        // Metodo para asignar el tag padre
```

```
        public void setParent(final Tag padre)
        {
                this.padre = padre;
        }

        // Metodo para obtener el padre
        public Tag getParent()
        {
                return padre;
        }

        // Establece el nombre al que saludar
        public void setNombre(String nombre)
        {
                this.nombre = nombre;
        }
}
```

El código es muy similar al visto para **FechaActual**, con algún cambio: se añade el campo **nombre**, que contendrá el nombre al que se saluda. Luego, en **doEndTag()** se muestra el mensaje de saludo, en lugar de la fecha. Finalmente, se tiene el método **setNombre()** que establece el nombre al que saludar. Esto lo hará automáticamente JSP (de forma similar a los beans), tomando el nombre del parámetro que se le pase al tag.

2. Crear el fichero TLD

En este caso modificamos el fichero TLD anterior, añadiéndole una nueva marca `tag` con los datos del nuevo tag:

```
        <tag>
                <name>saludo</name>
                <tagclass>pruebatags.Saludo</tagclass>
                <bodycontent>empty</bodycontent>
                <info>Muestra un saludo</info>
                <attribute>
                        <name>nombre</name>
                        <required>false</required>
                        <rtexprvalue>false</rtexprvalue>
                </attribute>
        </tag>
```

Llamamamos al tag `saludo` (el nombre que usaremos desde JSP). Al igual que el anterior, no tiene cuerpo. Con la etiqueta `attribute` le indicamos que tiene un atributo `nombre`, que es el nombre al que saludar.

3. Crear un JAR con la librería

Actualizamos el fichero JAR anterior o el directorio `classes` añadiendo la nueva clase creada.

4. Probar el tag

Creamos una página similar a la del tag anterior:

```
<%@ taglib uri="pruebatags" prefix="pt"%>

<HTML>
<BODY >
        Saludo:
        <pt:saludo nombre="Pepe"/>
</BODY>
</HTML>
```

4. BodyTags

Con los BodyTags podemos reevaluar el contenido del cuerpo del tag tantas veces como se quiera. Haremos con ello un iterador, que repetirá un número determinado de veces (pasada como parámetro) el contenido del cuerpo del tag:

1. Creación de la clase que implementa el tag

Llamaremos **Iterador** a la clase que implementa al tag.

```
package pruebatags;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class Iterador extends BodyTagSupport
{
        private PageContext contexto; // Contexto del tag
        private Tag padre;                // Tag padre del actual
        int veces = 10;                // Num de iteraciones
        int valorActual = 0;         // Valor del contador

        // Metodo llamado cuando se comienza el tag
        public int doStartTag() throws JspException
        {
                if (valorActual >= veces)
                        return SKIP_BODY;
                else
                {
                        valorActual++;
                        return EVAL_BODY_BUFFERED;
                }
        }

        // Metodo llamado tras cada evaluacion del tag
        public int doAfterBody() throws JspException
        {
                if (valorActual >= veces)
                        return SKIP_BODY;
                else
```

```
                {
                        valorActual++;
                        return EVAL_BODY_BUFFERED;
                }
        }

        // Metodo llamado cuando se termina el tag
        public int doEndTag() throws JspException
        {
                try
                {
                        if(bodyContent != null)
                               bodyContent.writeOut(
                               bodyContent.getEnclosingWriter());
                } catch(java.io.IOException e) {
                        throw new JspException("IO Error");
                }
                return EVAL_PAGE;
        }

        // Establece el numero de iteraciones
        public void setVeces(int veces)
        {
                this.veces = veces;
        }
}
```

Se hereda de la clase **BodyTagSupport** que implementa la interfaz `BodyTag`, con lo que sólo tendremos que sobreescribir los métodos que necesitemos. En el campo `veces` guardamos el número de iteraciones a realizar. Luego con `valorActual` vamos incrementando el contador hasta llegar a `veces`. El método **doStartTag()** se ejecuta al empezar el tag, y luego es el método **doAfterBody()** quien permite repetir el contenido del cuerpo mientras se cumpla la condición que se quiera. Al terminar, en **doEndTag()** imprimimos el resultado de procesar el cuerpo tras el número de iteraciones establecido. Con el método **setVeces()** asignamos el número de iteraciones (como en el caso anterior, esto lo hará automáticamente JSP tomando el número de iteraciones del parámetro que se le pase al tag).

2. Crear el fichero TLD

Modificamos el fichero TLD anterior, añadiéndole una nueva marca `tag` con los datos del nuevo tag:

```
        <tag>
                <name>itera</name>
                <tagclass>pruebatags.Iterador</tagclass>
                <bodycontent>JSP</bodycontent>
                <info>Muestra un saludo</info>
                <attribute>
                        <name>veces</name>
                        <required>true</required>
```

```
                        <rtexprvalue>true</rtexprvalue>
                </attribute>
        </tag>
```

Llamamamos al tag `itera` (el nombre que usaremos desde JSP). En `bodycontent` ponemos `JSP` para que sea JSP quien se encargue de evaluar el cuerpo del tag. Como atributo le pasamos el número de iteraciones, con un parámetro llamado `veces`. Dicho parámetro es requerido, y admite expresiones JSP en su valor.

3. Crear un JAR con la librería

Actualizamos el fichero JAR anterior o el directorio `classes` añadiendo la nueva clase creada.

4. Probar el tag

Utilizamos una página como:

```
<%@ taglib uri="pruebatags" prefix="pt"%>

<HTML>
<BODY >
        <%int valor = 0;%>
        <pt:itera veces="5">
                Valor = <%=valor%>
                <%valor+=10;%>
        </pt:itera>
</BODY>
</HTML>
```

5. Tags anidados

Veremos por último cómo crear tags contenidos en otros tags. Como ejemplo haremos un tag `switch-case` donde el tag padre contendrá el `switch` con un valor global, y dentro podrá tener uno o varios tag `case` que se ejecutarán en caso de que el valor global coincida con el suyo.

1. Creación de la clase que implementa el Tag

En este caso tenemos 2 clases (porque tenemos 2 tags). El `switch` padre lo implementamos en la clase **TagSwitch**:

```
package pruebatags;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class TagSwitch implements Tag
{
        private PageContext contexto; // Contexto del tag
```

```
        private Tag padre;                  // Tag padre del actual
        String valor;                 // Valor para comparar

        // Metodo llamado cuando se comienza el tag
        public int doStartTag() throws JspException
        {
                return EVAL_BODY_INCLUDE;
        }

        // Metodo llamado cuando se termina el tag
        public int doEndTag() throws JspException
        {
                return EVAL_PAGE;
        }

        // Metodo de limpieza
        public void release() {}

        // Metodo para asignar el contexto
        public void setPageContext(final PageContext contexto)
        {
                this.contexto = contexto;
        }

        // Metodo para asignar el tag padre
        public void setParent(final Tag padre)
        {
                this.padre = padre;
        }

        // Metodo para obtener el padre
        public Tag getParent()
        {
                return padre;
        }

        // Obtiene el valor con que comparar con los case
        public String getValor()
        {
                return valor;
        }

        // Establece el valor con que comparar con los case
        public void setValor(String valor)
        {
                this.valor = valor;
        }
}
```

En el campo `valor` se guarda el valor con que se compararán luego los `case` para ver cuál encaja. Con los métodos **getValor()** y **setValor()** se puede obtener/establecer dicho valor. Observar también que en **doStartTag()** se devuelve **EVAL_BODY_INCLUDE** para que se evalúe el cuerpo del tag. El resto es muy parecido a los primeros tags vistos.

Definimos también el tag `case`, implementado en la clase **TagCase**:

```
package pruebatags;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class TagCase implements Tag
{
        private PageContext contexto; // Contexto del tag
        private Tag padre;                // Tag padre del actual
        String valor;                // Valor del case

        // Metodo llamado cuando se comienza el tag
        public int doStartTag() throws JspException
        {
                TagSwitch miPadre = (TagSwitch)getParent();

                if(miPadre==null)
                        throw new JspException("case sin switch");

                try
                {
                        if(miPadre.getValor().equals(getValor()))
                                return EVAL_BODY_INCLUDE;
                        else
                                return SKIP_BODY;
                } catch(NullPointerException e) {

                        if(miPadre.getValor()==null ||
                           getValor()==null)
                                return EVAL_BODY_INCLUDE;
                        else
                                return SKIP_BODY;
                }
        }

        // Metodo llamado cuando se termina el tag
        public int doEndTag() throws JspException
        {
                return EVAL_PAGE;
        }

        // Metodo de limpieza
        public void release() {}

        // Metodo para asignar el contexto
        public void setPageContext(final PageContext contexto)
        {
                this.contexto = contexto;
        }

        // Metodo para asignar el tag padre
        public void setParent(final Tag padre)
```

```
        {
                this.padre = padre;
        }

        // Metodo para obtener el padre
        public Tag getParent()
        {
                return padre;
        }

        // Obtiene el valor del case
        public String getValor()
        {
                return valor;
        }

        // Establece el valor del case
        public void setValor(String valor)
        {
                this.valor = valor;
        }
}
```

El campo `valor` aquí guarda el valor de cada `case` concreto. En el método **doStartTag()** se obtiene el tag padre, y se compara su valor con el del tag `case` actual. Si coinciden, se evalúa el cuerpo del tag, y si no no se evalúa.

2. Crear el fichero TLD

Modificamos el fichero TLD anterior, añadiéndole los dos tags:

```
        <tag>
                <name>switch</name>
                <tagclass>pruebatags.TagSwitch</tagclass>
                <bodycontent>JSP</bodycontent>
                <info>Tag switch</info>
                <attribute>
                        <name>valor</name>
                        <required>true</required>
                        <rtexprvalue>true</rtexprvalue>
                </attribute>
        </tag>

        <tag>
                <name>case</name>
                <tagclass>pruebatags.TagCase</tagclass>
                <bodycontent>JSP</bodycontent>
                <info>Tag case</info>
                <attribute>
                        <name>valor</name>
                        <required>true</required>
                        <rtexprvalue>true</rtexprvalue>
                </attribute>
        </tag>
```

Llamamamos a los tags `switch` y `case` respectivamente. Como atributo le pasamos el valor de cada uno, con un parámetro llamado `valor`.

3. Crear un JAR con la librería

Actualizamos el fichero JAR anterior o el directorio `classes` añadiendo las nuevas clases creadas.

4. Probar el tag

Utilizamos una página como:

```
<%@ taglib uri="pruebatags" prefix="pt"%>

<HTML>
<BODY >
        <pt:switch valor="a">
                <pt:case valor="b">
                        Si sale esto es que el valor es b
                </pt:case>

                <pt:case valor="a">
                        Si sale esto es que el valor es a
                </pt:case>

                <pt:case valor="c">
                        Si sale esto es que el valor es c
                </pt:case>
        </pt:switch>
</BODY>
</HTML>
```