Creación de tags propias

Índice

1 Tag files....2
1.1 Paso de parámetros 2
1.2 Etiquetas con contenido....3
2 Simple tags....4
2.1 Clase Java para implementar el tag....4
2.2 Descriptor de despliegue (.tld)....5
2.3 simple tags con atributos....6

Como hemos visto en JSTL, las librerías de tags encapsulan funcionalidad Java en los JSP sin necesidad de insertar código. Podría ser útil por tanto desarrollar nuestras propias tags para el uso en nuestras aplicaciones. JSP nos ofrece esta posibilidad desde las primeras versiones del API.

La forma "clásica" de definir tags propias es muy potente y flexible, pero también compleja. Por ello, en la versión 2.0 de JSP se introdujeron alternativas más simples, que son las que veremos aquí.

- **Tag files**: permiten definir tags como bloques de código JSP, es decir, son similares a los *includes* que ya hemos visto en sesiones anteriores, pero en forma de tags.
- **Simple tags**: usan código Java para definir las tags, pero el API es mucho más sencillo de usar que el API "clásico".

1. Tag files

Un "tag file" es simplemente un archivo que encapsula un fragmento de código JSP reutilizable. Por convenio, estos archivos tienen extensión `.tag`. El caso más simple sería un bloque de texto fijo. Por ejemplo, supongamos que la siguiente línea se guarda en un archivo llamado `copyright.tag`

```
Copyright JTECH 2008
```

Los archivos .tag deben colocarse en un directorio `tags` dentro de la carpeta `WEB-INF`. Ahora podemos usar el tag en nuestro JSP de la misma forma que lo hacemos con taglibs de terceros:

```
<%@ taglib prefix="jtech" tagdir="/WEB-INF/tags" %>
<jtech:copyright/>
```

Nótese que el nombre del tag es el nombre físico del archivo en que está definido.

En este ejemplo tan trivial, usar un tag file sería equivalente a hacer un include. No obstante,como veremos a continuación, los tag files permiten el paso de parámetros de manera más sencilla que con `<jsp:include>`, generando por tanto JSPs mucho más claros y concisos.

1.1. Paso de parámetros

Los archivos de tags se pueden parametrizar. Estos parámetros se denominan en el estándar JSP *atributos* y se definen con la directiva JSP del mismo nombre. Dicha directiva solo es válida en archivos de tag. Por ejemplo, podemos cambiar `copyright.tag` por el siguiente:

```
<%@ attribute name="anyo" required="true" rtexprvalue="false"%>
Copyright JTECH ${anyo}
```

El `required` especifica si el parámetro es o no obligatorio y `rtexprvalue` si puede usarse EL en el parámetro o tiene que ser un literal. En nuestro ejemplo debe ser esto último. Ahora para usarlo en un JSP, haríamos

```
<%@ taglib prefix="jtech" tagdir="/WEB-INF/tags" %>
<jtech:copyright anyo="2008"/>
```

Por supuesto, el parámetro será más útil si puede ser variable. El siguiente tag está preparado para aceptar una variable Java de tipo Date y formatearla en el formato día/mes/año:

```
<%@ tag import="java.util.Date"
import="java.text.SimpleDateFormat"%>
<%@ attribute name="fecha" required="true" type="java.util.Date" %>
<%
        SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
        out.print(sdf.format(fecha));
%>
```

Hay varios puntos interesantes en el archivo de tag anterior:

- En la directiva `attribute`, el `type` indica el tipo de dato que aceptará nuestro tag como parámetro
- La directiva `tag`, que hasta el momento no habíamos necesitado, nos permite importar los *packages* necesarios. Es similar a la directiva `page` usada en los JSP.
- Hemos introducido un par de líneas de código Java en forma de *scriptlet*. Hay que recalcar que un archivo de tag no es más que un fragmento de JSP, por lo que será válida cualquier construcción que se pueda usar en un JSP.

Ahora para demostrar el uso de nuestra recién creada etiqueta podemos hacer:

```
<%@ taglib prefix="jtech" tagdir="/WEB-INF/tags"%>
<jtech:fmtFecha fecha="<%= new java.util.Date() %>"/>
```

Obsérvese que como valor de fecha hemos usado un *scriptlet* en lugar de una constante, a diferencia del ejemplo anterior.

1.2. Etiquetas con contenido

En todos los ejemplos vistos hasta el momento, el tag creado no tenía "cuerpo", solo atributos. Podemos usar también el valor del cuerpo en nuestros tags. Por ejemplo, supongamos un tag propio que pase un texto a mayúsculas:

```
<%@ taglib prefix="test" tagdir="/WEB-INF/tags"%>
...
```

```
<test:capitalizar>Esto es un texto que la etiqueta
debería pasar a mayúsculas</test:capitalizar>
```

La acción `<jsp:body/>` obtiene el contenido del cuerpo de nuestro tag. El atributo `var` de esta acción nos permite guardarlo en un objeto en ámbito de página, petición, sesión o aplicación. El ámbito se controla con el atributo `scope`. Veamos cómo se podría implementar nuestro tag:

```
<jsp:doBody var="texto" scope="request"/>
<%
        String texto = (String) request.getAttribute("texto");
        out.print(texto.toUpperCase());
%>
```

2. Simple tags

En tags propios que requieran gran cantidad de código java, los *scriptlets* necesarios en el archivo de tag pueden volverse demasiado difíciles de mantener, dándose el mismo problema que tienen los JSP con gran cantidad de código Java. En ese caso, será mucho mejor encapsular el código del tag *en una clase Java aparte*. Esta idea está presente en JSP desde las primeras versiones. El problema es que cuando se diseñó la forma "clásica" de definir tags con código Java se pensó en la potencia y flexibilidad, pero no en la facilidad de desarrollo. Así, la complejidad de crear tags java de la forma clásica no está justificada en la gran mayoría de casos, excepto los más sofisticados. Para resolver este problema, en la versión 2.0 de JSP se introdujo una forma simplificada de programar tags que se llamó **simple tags**.

Crear un tag simple implica dos pasos:

- Escribir la clase Java que implementa el tag
- Crear un descriptor de despliegue (.tld)

2.1. Clase Java para implementar el tag

La clase Java que implemente el tag debe implementar el interfaz `SimpleTag`. Dicho interfaz tiene los métodos que se muestran a continuación:

```
package javax.servlet.jsp.tagext;
public interface SimpleTag extends JspTag {
        public void doTag() throws JspException, IOException;
        public JspTag getParent();
        public void setJspBody(JspFragment jspBody);
        public void setJspContext(JspContext jspContext);
        public void setParent(JspTag parent);
}
```

Cuando el contenedor web se encuentra con uno de nuestros tags, instancia un objeto de

la clase que lo implementa y va llamando a los métodos del interfaz en un cierto orden:

1. `setJspContext`: le pasa a nuestro tag la referencia al `JspContext`, que nos va a proporcionar, si lo necesitamos, acceso a los diferentes ámbitos (sesión, aplicación,...) y al `JspWriter` que representa la salida
2. `setParent`: nuestro tag recibirá información de quién es su tag "padre". Eso abre la posibilidad de anidar tags propios de manera que "colaboren" entre sí, modificando su comportamiento y pasándose información cuando están anidados.
3. `setJspBody`: nuestro tag recibe también el cuerpo que contiene, en forma de objeto de la clase `JspFragment`
4. `doTag`: es el método más importante. Indica que "es hora de hacer el trabajo".

Veamos un ejemplo sencillo, que se limita a mostrar la fecha del sistema en el formato día/mes/año

```
package jtech;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.SimpleTagSupport;

public class TestSimpleTag extends SimpleTagSupport {
        public void doTag() throws JspException, IOException {
                SimpleDateFormat sdf = new
SimpleDateFormat("dd/MM/yy");
                getJspContext().getOut().write(sdf.format(new
Date()));
        }
}
```

En el código, en lugar de implementar directamente el interfaz `SimpleTag`, que nos obligaría a definir todos sus métodos, los necesitemos o no, hemos heredado de la clase `SimpleTagSupport`, que ya incluye una definición por defecto para ellos.

El trabajo de nuestro tag es muy sencillo. A través del `getJspContext().getOut()` obtenemos el `JspWriter` de la salida y en él escribimos la fecha formateada.

Evidentemente este es un ejemplo con un código tan trivial que no justifica el uso de un tag simple en lugar de un archivo de tag. No obstante, conforme el número de líneas y la complejidad de código va creciendo, la balanza se inclina progresivamente más hacia este tipo de tag, sin olvidar las posibilidades que se nos dan de anidar tags propios comunicándolos entre sí.

2.2. Descriptor de despliegue (.tld)

Se debe crear un archivo .tld en el que, con etiquetas XML describimos las características de nuestros nuevos tags. Por convenio, se suele colocar en `WEB-INF/tlds`, aunque esto no es obligatorio. Aquí tenemos el .tld para nuestro ejemplo:

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd"
  version="2.1">
  <description>
    Ejemplo de simple tag
  </description>
  <jsp-version>2.1</jsp-version>
  <tlib-version>1.0</tlib-version>
  <short-name>test</short-name>
  <uri>http://www.jtech.ua.es/ayto/test</uri>
  <tag>
    <name>test</name>
    <tag-class>jtech.TestSimpleTag</tag-class>
    <body-content>empty</body-content>
    <description>
        Es una prueba tonta, en realidad
    </description>
  </tag>
</taglib>
```

Como vemos, en el descriptor, entre otros datos, decimos qué clase Java implementa el tag y cuál va a ser su nombre en JSP (test). Ahora ya podemos usarlo en nuestros JSPs:

```
<%@ taglib prefix="jtech" uri="WEB-INF/tlds/test.tld"%>
...
<jtech:test/>
```

2.3. simple tags con atributos

Un ejemplo más realista de tag usaría atributos. Afortunadamente su uso es muy sencillo. Por cada atributo que acepte nuestro tag simplemente necesitamos un *getter* y un *setter* en nuestro código Java. Modificando el ejemplo anterior:

```
public class TestSimpleTag extends SimpleTagSupport {
        private Date fecha;

        public void doTag() throws JspException, IOException {
                SimpleDateFormat sdf = new
SimpleDateFormat("dd/MM/yy");
                getJspContext().getOut().write(sdf.format(fecha));
        }
        public Date getFecha() {
                return fecha;
        }
        public void setFecha(Date fecha) {
                this.fecha = fecha;
        }
}
```

Ahora nuestro tag admitirá un atributo *fecha* en el que le pasamos la fecha a formatear.

Nos falta especificarlo en el .tld:

```
<?xml version="1.0" encoding="UTF-8" ?>
<taglib xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee/web-jsptaglibrary_2_1.xsd"
  version="2.1">
  <description>
    Ejemplo de simple tag
  </description>
  <jsp-version>2.1</jsp-version>
  <tlib-version>1.0</tlib-version>
  <short-name>test</short-name>
  <uri>http://www.jtech.ua.es/ayto/test</uri>
  <tag>
    <name>test</name>
    <tag-class>jtech.TestSimpleTag</tag-class>
    <body-content>empty</body-content>
    <description>
        Es una prueba tonta, en realidad
    </description>
    <attribute>
        <name>fecha</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
    </attribute>
  </tag>
</taglib>
```