Servlets y JSP

Sesión 10: Creación de *taglibs*

Puntos a tratar

- Introducción a la creación de librerías de tags
- Creación de un tag simple
- Creación de un tag con parámetros
- Creación de un *BodyTag*
- Creación de tags anidados

Introducción

- Veremos a continuación cómo crear algunos de los tags más comunes
- Necesitaremos los paquetes `javax.servlet.jsp` y `javax.servlet.jsp.tagext`
- Utilizando estos paquetes, haremos clases Java para cada tag. El contenedor JSP llamará a estas clases cuando encuentre los tags correspondientes

Ejemplo concreto

- Iremos incorporando ejemplos concretos de tags a una librería propia, que llamamos `pruebatags`
- Las clases Java para cada tag estarán en un paquete `pruebatags`
- Probaremos la librería en una aplicación *tags* cuyo fichero descriptor (`web.xml`) tendrá:

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

- El fichero TLD lo iremos modificando a medida que vayamos añadiendo tags

Pasos a seguir

- Para cada tag que se cree, seguiremos los mismos pasos:
  - Creación de la clase que implementa el tag
  - Añadir el tag al fichero TLD
  - Regenerar el fichero JAR con la librería, y colocarlo en `WEB-INF/lib` en la aplicación Web
  - Probar el tag

Tag que muestra la fecha actual

```
package pruebatags;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class FechaActual implements Tag {
   private PageContext contexto;
   private Tag padre;

   public int doStartTag() throws JspException { return SKIP_BODY; }

   public int doEndTag() throws JspException {
        contexto.getOut().write("... imprimir fecha actual");
        return EVAL_PAGE;
   }

   public void release() {}
   public void setPageContext(final PageContext c){ contexto = c; }
   public void setParent(final Tag p) { padre = p; }
   public Tag getParent() { return padre; }
}
```

Aclaraciones del código

- `doStartTag( )` se llama cuando se empieza a procesar el tag. Puede devolver:
  - `SKIP_BODY`: para no procesar el cuerpo del tag
  - `EVAL_BODY_INCLUDE`: para explorar el cuerpo del tag
- `doEndTag( )` se llama cuando se termina de procesar el tag. Aquí se emite la respuesta (se muestra la fecha por la salida JSP). Puede devolver:
  - `EVAL_PAGE`: para seguir evaluando la página
  - `SKIP_PAGE`: para dejar de evaluar la página
- El resto de métodos obtienen/establecen campos, o liberan recursos si es necesario (`release( )`)

Modificación del fichero TLD

- Como es nuestro primer tag, el fichero TLD está vacío, con lo que creamos la cabecera, la etiqueta padre `taglib`, y el tag que acabamos de hacer

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems ..." ... >
<taglib>
   <tlibversion>1.0</tlibversion>
   <jspversion>1.1</jspversion>
   <shortname>pt</shortname>
   <uri>pruebatags</uri>

   <tag>
        <name>fechaactual</name>
        <tagclass>pruebatags.FechaActual</tagclass>
        <bodycontent>empty</bodycontent>
   </tag>
</taglib>
```

Regenerar el JAR con la librería

- Cada nuevo tag que añadamos implicará actualizar el fichero JAR con la librería completa. Podemos optar por 2 opciones:
  - Tener las clases de la librería en un fichero JAR, que vayamos actualizando con cada cambio. Dicho fichero JAR lo copiaremos en el directorio `WEB-INF/lib` de nuestra aplicación
  - Añadir las clases sueltas (ficheros `.class`) al directorio `WEB-INF/classes` de nuestra aplicación
- La primera opción es más recomendable para poder llevar la librería a otras aplicaciones o máquinas
- Por comodidad, en este ejemplo utilizaremos la segunda opción, para no ir regenerando el JAR en cada cambio
- Así, para cada tag, añadiremos directamente su clase a `WEB-INF/classes`

Probar el tag

- Creamos una pequeña página JSP para probar el tag en nuestra aplicación Web:

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

<html>
<body>
   La fecha actual es:
   <pt:fechaactual/>
</body>
</html>
```

Tag que saluda al nombre que se le pase

```
package pruebatags;
...
public class Saludo implements Tag {
   private PageContext contexto;
   private Tag padre;
   String nombre = “”;

   public int doStartTag() throws JspException { return SKIP_BODY; }

   public int doEndTag() throws JspException {
        contexto.getOut().write(“Hola ” + nombre);
        return EVAL_PAGE;
   }

   public void release() {}
   public void setPageContext(final PageContext c){ contexto = c; }
   public void setParent(final Tag p) { padre = p; }
   public Tag getParent() { return padre; }
   public void setNombre(String nombre) { this.nombre = nombre; }
}
```

Aclaraciones del código

- El código es muy parecido al del tag anterior
- Se tiene un campo `nombre` que guarda el nombre al que saludar (que lo pasaremos como atributo del tag)
- En `doEndTag( )` ahora simplemente se muestra el mensaje de saludo
- Se añade el método `setNombre(...)` para que el contenedor lo utilice para asignar valor al atributo del tag

Modificación del fichero TLD

- Añadimos un nuevo bloque `tag` con el nuevo tag creado:

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

Probar el tag

- Creamos una pequeña página JSP para probar el tag en nuestra aplicación Web:

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

<html>
<body>
   Saludo:
   <pt:saludo nombre="Pepe"/>
</body>
</html>
```

Tag que itera su cuerpo varias veces

```
...
public class Iterador extends BodyTagSupport {
   private PageContext contexto;
   private Tag padre;
   int veces = 10, valorActual = 0;

   public int doStartTag() throws JspException {
        if (valorActual >= veces) return SKIP_BODY;
        else { valorActual++; return EVAL_BODY_BUFFERED; }
   }
   public int doAfterBody() throws JspException {
        if (valorActual >= veces) return SKIP_BODY;
        else { valorActual++; return EVAL_BODY_BUFFERED; }
   }
   public int doEndTag() throws JspException {
        bodyContent.writeOut(bodyContent.getEnclosingWriter());
        return EVAL_PAGE;
   }
   public void setVeces(int veces) { this.veces = veces; }
}
```

Aclaraciones del código

- Ahora se hereda de `BodyTagSupport`
- *veces* almacena el número de iteraciones, y `valorActual` la iteración actual
- `doStartTag( )` comprueba si se han pasado las iteraciones, y si no, se evalúa el cuerpo del tag
- `doAfterBody( )` se llama tras cada evaluación del cuerpo. Comprueba si se han pasado las iteraciones para salir del tag, o si por el contrario se debe seguir evaluando
- `doEndTag( )` vuelca a la salida el resultado de haber explorado el cuerpo del tag
- El método `setVeces(...)` lo utiliza el contenedor JSP para establecer desde un atributo el número de iteraciones

Modificación del fichero TLD

- Añadimos un nuevo bloque `tag` con el nuevo tag creado:

```
<tag>
   <name>itera</name>
   <tagclass>pruebatags.Iterador</tagclass>
   <bodycontent>JSP</bodycontent>
   <attribute>
        <name>veces</name>
        <required>true</required>
        <rtexprvalue>true</rtexprvalue>
   </attribute>
</tag>
```

Probar el tag

- Creamos una pequeña página JSP para probar el tag en nuestra aplicación Web:

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

<html>
<body>
   <% int valor = 0; %>
   <pt:itera veces="10">
        Valor = <%= valor %>
        <% valor += 10; %>
   </pt:itera>
</body>
</html>
```

Tag *switch*

```
...
public class TagSwitch implements Tag {
   private PageContext contexto;
   private Tag padre;
   String valor;

   public int doStartTag() throws JspException { return EVAL_BODY_INCLUDE; }

   public int doEndTag() throws JspException { return EVAL_PAGE; }

   public String getValor() { return valor; }
   public void setValor(String valor) { this.valor = valor; }

   public void release() {}
   ... // setParent(), setPageContext()... etc
}
```

Aclaraciones del código

- El campo `valor` contiene el valor que se va a comparar en el `switch` para ver en qué *case* entrar
- Los métodos `setValor(...)` y `getValor( )` manipulan el valor de dicho atributo
- Observar que `doEndTag( )` devuelve `EVAL_BODY_INCLUDE` para que se examine el contenido del tag
- La diferencia entre implementar `Tag` y `BodyTag` NO ES no poder explorar el contenido, sino no poder iterarlo o procesarlo (no hay `doAfterBody( )` en la interfaz `Tag`)

Tag *case* interno al *switch*

```
...
public class TagCase implements Tag {
   private PageContext contexto;
   private Tag padre;
   String valor;

   public int doStartTag() throws JspException {
        TagSwitch miPadre = (TagSwitch)getParent();
        if (miPadre.getValor().equals(valor))      return EVAL_BODY_INCLUDE;
        else                                       return SKIP_BODY;
   }

   public int doEndTag() throws JspException { return EVAL_PAGE; }

   public String getValor() { return valor; }
   public void setValor(String valor) { this.valor = valor; }

   ... // release(), setParent(), setPageContext()... etc
}
```

Aclaraciones del código

- El campo `valor` aquí guarda el valor concreto de cada *case* con el que se compara el valor del `switch` general
- `doStartTag( )` compara el valor del tag padre (`switch`) con el del tag `case` actual, y si coinciden, evalúa el cuerpo del `case` y si no, no lo evalúa
- `doEndTag( )` permite seguir evaluando la página
- `setValor(...)` y `getValor( )` se emplean para manipular el atributo `valor`

Modificación del fichero TLD

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

Probar el tag

- Creamos una pequeña página JSP para probar los dos tags anidados:

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

<html>
<body>
   <pt:switch valor="a">
        <pt:case valor="b"> Si sale esto es que vale b </pt:case>
        <pt:case valor="a"> Si sale esto es que vale a </pt:case>
        <pt:case valor="c"> Si sale esto es que vale c </pt:case>
   </pt:switch>
</body>
</html>
```