JavaBeans y lenguaje de expresiones

Índice

1. JavaBeans

Un **JavaBean** (o, para abreviar, un *bean*) es un componente software reutilizable escrito en Java. En realidad un *bean* no es más que una clase Java escrita siguiendo unas ciertas convenciones. Estas convenciones hacen posible que herramientas automáticas puedan acceder a sus propiedades y manipularlas sin necesidad de modificar el código. Esto puede servir en el caso de un IDE, por ejemplo, para realizar "programación visual". En JSP el uso principal de los *beans* es manipular componentes Java sin necesidad de incluir código en la página, accediendo a sus propiedades mediante etiquetas.

El uso de *beans* en páginas JSP ofrece diversas ventajas con respecto al uso directo de código Java:

- Se evita el uso de sintaxis Java, en su lugar se emplean etiquetas con sintaxis XML. Esto permite separar más fácilmente el trabajo de programadores y diseñadores web.
- Se simplifica la creación y uso de objetos compartidos entre varias páginas.
- Se simplifica la creación de objetos a partir de los parámetros de la petición

1.1. Características de un bean

Como se ha comentado, un *bean* no es más que una clase Java en la que se observan ciertas convenciones. En lo que respecta a su uso con JSP, estas convenciones afectan al modo de definir constructores, métodos y variables miembro:

1. Un *bean* debe tener al menos un constructor sin argumentos. Este constructor será llamado cuando una página JSP cree una instancia del *bean*.
2. Un *bean* no debe tener variables miembro de acceso público. El acceso a las variables y su modificación se debe hacer a través de métodos.
3. El nombre de los métodos de acceso y modificación de variables miembro debe seguir una norma: si la variable tiene el nombre `nombreVar`, entonces el método de acceso debe llamarse `getNombreVar` (obsérvese el cambio a mayúsculas de la "N", siguiendo las convenciones habituales de Java), y el método de cambio de valor (en caso de que exista) debe llamarse `setNombreVar`. En el caso especial de variables booleanas, el método de acceso se debe denominar `isNombreVar`.

Por ejemplo, supongamos que se desea definir un *bean* para almacenar información relativa a un usuario (nombre, número de visitas al sitio, fecha de la última visita y sexo), y compartirla entre varias páginas, una vez que se ha autentificado en la aplicación y sabemos quién es. Para ello podríamos utilizar un código similar al siguiente:

```
package beans;
```

```
import java.util.Date;

public class UsuarioBean
{
  //variables miembro, privadas
  private String nombre;
  private int visitas;
  private Date ultimaVisita;
  private boolean varon;

  //constructor sin argumentos
  public UsuarioBean() {
     nombre = null;
     visitas = 0;
     ultimaVisita = null;
     varon = false;
  }

  //métodos de acceso: getXXX, isXXX
  public String getNombre() {
     return nombre;
  }

  public int getVisitas() {
     return visitas;
  }

  public Date getUltimaVisita() {
     return ultimaVisita;
  }

  public boolean isVaron() {
     return varon;
  }

  //métodos de cambio de valor: setXXX
  public void setNombre(String nom) {
     nombre = nom;
  }

  public void setVisitas(int v) {
     visitas = v;
  }

  public void setUltimaVisita(Date fecha) {
     ultimaVisita = fecha;
  }

  public void setVaron(boolean valor) {
     varon = valor;
  }
}
```

1.2. Uso de beans desde páginas JSP

Para interactuar con un *bean* desde una página JSP es necesario primero asignarle un nombre y especificar qué clase Java lo define. Una vez hecho esto, se puede acceder a sus propiedades y darles nuevos valores.

1. Acceso al *bean*

Para hacer accesible un *bean* a una página JSP se emplea la etiqueta **jsp:useBean**. En su forma más simple la sintaxis es:

```
<jsp:useBean id="nombreBean" class="paquete.Clase"/>
```

En caso de que el *bean* referenciado no existiera previamente, esta etiqueta se puede ver como la creación de una variable en Java de nombre `nombreBean` y de tipo `paquete.Clase`. Así, para crear un *bean* de tipo `UsuarioBean` sería equivalente utilizar las siguientes expresiones:

```
<jsp:useBean id="usuario" class="beans.UsuarioBean" />
<% beans.UsuarioBean usuario = new beans.UsuarioBean() %>
```

La clase a la que pertenece el *bean* debería colocarse donde están habitualmente las clases Java que pertenecen a una aplicación web, es decir, en `WEB-INF/classes`. Para que el contenedor JSP pueda encontrar la clase del *bean*, es conveniente que éste pertenezca a un *package* (como en el ejemplo anterior). En caso contrario se asumiría que pertenece al mismo *package* que el servlet generado a partir del JSP, con el problema de que el nombre de este *package* es desconocido.

2. Acceso a las propiedades del *bean*

El acceso a una propiedad se realiza mediante la etiqueta **jsp:getProperty**. Su sintaxis es:

```
<jsp:getProperty name="nombreBean" property="nombrePropiedad"/>
```

donde `nombreBean` debe corresponderse con el atributo `id` definido mediante alguna etiqueta anterior `jsp:useBean`. El acceso a la propiedad también se podría hacer llamando al método Java correspondiente. De este modo, el acceso a la propiedad `visitas` del *bean* `usuario` se puede hacer mediante las dos formas alternativas:

```
<jsp:getProperty name="usuario" property="visitas"/>
<%= usuario.getVisitas() %>
```

aunque se considera preferible la primera forma, ya que la sintaxis es más accesible a diseñadores web que no programen en Java.

3. Asignación de valores a las propiedades del *bean*

La asignación de valores se realiza mediante la etiqueta **jsp:setProperty**. Esta etiqueta requiere tres parámetros: `name` (el `id` del *bean*, definido anteriormente mediante alguna

etiqueta `jsp:useBean`), `property` (el nombre de la propiedad) y `value` (el valor que se desea dar a la propiedad). Por ejemplo, para darle a la propiedad `visitas` del *bean* `usuario` el valor 1 se haría:

```
<jsp:setProperty name="usuario" property="visitas" value="1"/>
```

Una forma alternativa en código Java sería llamar directamente al método, aunque normalmente es preferible el uso de la sintaxis anterior:

```
<% usuario.setVisitas(1) %>
```

Además de poder asignar a una propiedad un valor fijo, se pueden usar expresiones JSP:

```
<jsp:setProperty name="usuario" property="ultimaVisita"
                 value="<%= new java.util.Date() %>"/>
```

4. Inicialización de un *bean*

En algunos casos, puede ser necesario inicializar un bean antes de empezar a usarlo. Esto no se puede hacer directamente con la etiqueta `jsp:useBean`, ya que no admite parámetros. Para solucionar el problema, en el cuerpo de la etiqueta `jsp:useBean` (siguiendo sintaxis XML) se pueden introducir etiquetas `jsp:setProperty` que inicialicen las propiedades. Además se pueden colocar *scriptlets* y código HTML.

```
<jsp:useBean id="usuario" class="beans.usuarioBean">
   <b> inicializando datos de usuario </b>
   <jsp:setProperty name="usuario" property="ultimaVisita"
                    value="<%= new java.util.Date() %>"/>
</jsp:useBean>
```

Es importante destacar que el código de inicialización solo se ejecutará en caso de que el *bean* no existiera previamente (no sea un *bean* compartido con otras páginas o creado por ejemplo por un servlet).

5. Utilizar los parámetros de la petición HTTP

JSP incluye un mecanismo para asignar los valores de los parámetros de la petición a las propiedades de un *bean*. Para ello hay que utilizar el parámetro `param` de la etiqueta `jsp:setProperty`. Por ejemplo, supongamos que se ha definido el siguiente formulario, que toma el nombre del usuario y llama a la página **main.jsp** con los datos introducidos por el usuario (nombre y sexo):

```
<html>
<body>
<form action="main.jsp" method="get">
        Nombre <input type="text" name="nombre">
        <br>
        Sexo: varon <input type="radio" name="varon" value="true">
                mujer: <input type="radio" name="varon" value="false"> <br>
        <input type="submit" value="entrar">
</form>
</body>
</html>
```

En la página **main.jsp** se puede hacer uso de los parámetros para instanciar algunas propiedades del bean:

```
<html>
<body>
   <jsp:useBean id="usuario" class="beans.UsuarioBean"/>
   <jsp:setProperty name="usuario" property="nombre" param="nombre"/>
   <jsp:setProperty name="usuario" property="varon" param="varon"/>
   Buenos días, <jsp:getProperty name="usuario" property="nombre"/>
</body>
</html>
```

Nótese que, aunque los parámetros HTTP son en realidad cadenas, el contenedor JSP es capaz de efectuar la conversión al tipo correspondiente, al menos para tipos primitivos (por ejemplo, se ha convertido de la cadena "true" al valor `true` que requiere el método `setVaron`. Esta conversión de tipos no funciona en caso de tipos no primitivos. Por ejemplo, no se puede aplicar a la propiedad `ultimaVisita`, de tipo `java.util.Date`, ya que no se puede convertir automáticamente de cadena a `Date` (al menos el estándar JSP no lo exige).

En caso de que las propiedades del *bean* tengan el mismo nombre que los parámetros HTTP, (como en el caso anterior) la asignación de todos los parámetros se puede hacer mediante una única etiqueta `setProperty`, con el parámetro `property="*"`.

```
<html>
<body>
   <jsp:useBean id="usuario" class="beans.UsuarioBean"/>
   <jsp:setProperty name="usuario" property="*" />
   Buenos días, <jsp:getProperty name="usuario" property="nombre"/>
</body>
</html>
```

1.3. Compartir beans

Hasta el momento, se han tratado los *beans* como si fueran objetos propios de la página en la que se definen, y exclusivos de ella. Este es el comportamiento por defecto, pero podemos cambiar el **ámbito** de un *bean* para definir desde dónde será accesible, lo que nos permite compartirlo entre varias páginas. El ámbito se controla con el atributo `scope` de la etiqueta `jsp:useBean`, y puede tomar cuatro valores distintos:

- **`page`**: es el valor por defecto
- **`application`**: el bean será accesible a todas las páginas JSP de la aplicación web, y compartido entre todos los usuarios. Los servlets pueden acceder a él a través del objeto `ServletContext`.
- **`session`**: el bean será accesible a todas las páginas JSP, pero cada usuario tendrá su propio objeto. Los servlets pueden acceder a él a través del objeto `HttpSession`, obteniendo su valor con el método `getAttribute`.

- **`request`**: el bean será accesible durante la petición actual, lo que significa que podrán acceder a él las páginas a las que se desvíe la petición con un `<jsp:include>` o un `<jsp:forward>`. Los servlets pueden acceder a él a través del objeto `ServletRequest`, de donde se puede obtener su valor con `getAttribute`.

2. Lenguaje de expresiones en JSP 2.0

2.1. Introducción al lenguaje de expresiones

Desde las versión 2.0 de JSP se ha introducido en las páginas un lenguaje de expresiones que es equivalente en gran medida a los scriptlets <%= ... %> que venimos utilizando hasta ahora. En realidad, dicho lenguaje se implantó con las primeras versiones de JSTL, una librería de etiquetas JSP que se considera estándar, y que permitía utilizar este lenguaje de expresiones en los atributos de dichas etiquetas, constituyendo una característica muy importante de JSTL.

Cualquier elemento que pertenezca al lenguaje de expresiones irá englobado dentro de la marca **${...}**. En ella podremos colocar nombres de variables que hayamos definido en la página, parámetros de petición HTTP, elementos de una sesión... etc.

Por ejemplo, si previamente hemos creado una variable *miVar*, podríamos acceder a ella desde el propio contenido JSP con algo como:

```
<h3>La variable miVar vale ${miVar}</h3>
```

Si lo que queremos es mostrar el parámetro *password* que hemos tomado de un formulario, para sacarlo por pantalla o guardarlo en alguna base de datos, podríamos acceder a él con algo como:

```
Accediendo al parámetro ${param.password}
```

También se pueden utilizar, como veremos, expresiones más complejas, que se evalúan desde el contenedor JSP. Por ejemplo, si tenemos una variable *edad* para una persona y queremos comprobar si dicha persona es mayor de edad, podríamos poner:

```
${edad > 18}
```

Y luego utilizar el resultado de esta expresión en otras zonas (por ejemplo, etiquetas condicionales de JSTL) para realizar la acción correspondiente.

Se describe a continuación, y a grandes rasgos, el lenguaje de expresiones incluido en JSTL 1.0, y en general a partir de JSP 2.0. El lenguaje está inspirado en los lenguajes ECMAScript y XPath, y está basado en espacios de nombres (atributos *PageContext*), propiedades de elementos, operadores relacionales, lógicos y aritméticos, y un conjunto de objetos implícitos.

2.2. Atributos y expresiones

Como hemos comentado anteriormente, podremos invocar a este lenguaje desde cualquier lugar de nuestra página (en JSP 2.0), o dentro de un atributo de una etiqueta JSTL (desde JSTL 1.0), mediante el elemento **${...}**:

```
${expresion}
```

Esta expresión podrá estar:

- Por sí sola dentro de un atributo JSTL: :

  ```
  <c:set var="miVariable" value="${expresion}"/>
  ```

  En este caso, se evalúa la expresión y el resultado se convierte al tipo de dato del atributo, siguiendo las reglas de conversión internas del lenguaje.
- Combinada con texto dentro de un atributo JSTL:

  ```
  <c:set var="miVariable" value="texto${e1}y ${e2}texto"/>
  ```

  Aquí, las expresiones se evalúan de izquierda a derecha, y se intercalan entre el texto, convirtiéndolas a *String* (siguiendo reglas de conversión internas). Luego, la cadena resultante se convierte al tipo del atributo en el que esté (si está dentro de algún atributo).
- Fuera de atributos, dentro del contenido HTML de la página JSP:

  ```
  <h3>Hola, esto es una página</h3>
  <p>Y aquí ponemos una expresión ${expresion}, para mostrar su valor</p>
  ```

  Para cadenas que contengan la secuencia '${' sin que sea propiamente una expresión, se encapsula esa secuencia así: **${'${'}**. Por ejemplo:

  ```
  Cadena con ${'${'}expr}"/>
  ```

  Mostraría: "*Cadena con ${expr}*"

2.3. Operadores

- **Operadores '[ ]' y '.'**: se unifican los operadores [ ] y . de forma que son equivalentes:

  ```
  ${expr.campo}
  ${expr["campo"]}
  ```
- **Operadores aritméticos**:
  - **+, -, *,** /: suma, resta, multiplicación y división
  - **div**: división entera
  - **%, mod**: resto (se mantienen los dos por compatibilidad con XPath y ECMAScript)
  - **-**: cambio de signo
- **Operadores relacionales**:
  - **>, gt**: mayor que

  - **<, lt**: menor que
  - **>=, ge**: mayor o igual que
  - **<=, le**: menor o igual que
  - **==, eq**: igual que
  - **!=, ne**: distinto que
- **Operadores lógicos**:
  - **&&, and**: Y lógica
  - **||, or**: O lógica
  - **!, not**: NO lógica
- **Operador empty**: utilizado delante de un elemento, para indicar si el elemento es nulo o vacío (devolvería *true*) o no (devolvería *false*). Por ejemplo:

```
${empty A}
```

PRECEDENCIA

- **[ ], .**
- **( )**
- **-** (cambio de signo), **not, !, empty**
- ***, /, div, %, mod**
- **+, -**
- **<, >, <=, >=, lt, gt, le, ge**
- **==, !=, eq, ne**
- **&&, and**
- **||, or**

EJEMPLOS

```
${empty param.nombre} // Daría true si el parametro nombre no se ha enviado
${num1 + num2}         // Devolvería el resultado de la suma de ambas variables
${valor1 >= valor2} // Devolvería true si valor1 es mayor o igual que valor2
${v1 ne v2 and v3 < v4} // Daría true si v1 fuese distinto a v2, y v3 menor que v4
```

2.4. Nombres de variables

El lenguaje de expresiones evalúa un identificador o nombre de elemento mirando su valor como un atributo, según el comportamiento del método *PageContext.findAttribute(String)*. Por ejemplo, si ponemos:

```
${valor}
```

Se buscará el atributo *valor* en los ámbitos de página (*page*), petición (*request*), sesión

session) y aplicación (*application*), y si lo encuentra devuelve su valor. Si no, se devuelve *null*.

Objetos implícitos

Cuando como nombre de atributo se utiliza alguno de los que el lenguaje de expresiones considera como implícitos, se devolverá el objeto asociado. Dichos objetos implícitos son:

- **pageContext**: el objeto *PageContext* actual
- **pageScope, requestScope, sessionScope, applicationScope**: para obtener valores de atributos de página / petición / sesión / aplicación, respectivamente.
- **param**: para obtener el valor de un parámetro de petición. Se obtiene un tipo *String* (utilizando el método *ServletRequest.getParameter(String)*)
- **paramValues**: para obtener los valores de un parámetro de petición. Se obtiene un tipo *String[ ]* (utilizando el método *ServletRequest.getParameterValues(String)*).
- **header**: para obtener el valor de un parámetro de cabecera. Se obtiene un tipo *String* (utilizando el método *ServletRequest.getHeader(String)*)
- **headerValues**: para obtener los valores de un parámetro de cabecera. Se obtiene un tipo *String[ ]* (utilizando el método *ServletRequest.getHeaderValues(String)*).
- **cookie**: para obtener el valor de una cookie. Se obtiene un objeto *Cookie*. Las cookies se buscan con *HttpServletRequest.getCookies()*
- **initParam**: para obtener el valor de un parámetro de inicialización. Se obtiene un tipo *String* (utilizando el método *ServletContext.getInitParameter(String)*)

EJEMPLOS

```
${sessionScope.profile}
```

Se obtiene el atributo *profile* de la sesión

```
${param.id}
```

Se obtiene el valor del parámetro *id* de la petición, o *null* si no se encuentra.

Palabras reservadas

Se tienen algunos identificadores que no podemos utilizar como nombres de atributos, como son *and, eq, gt, true, instanceof, or, ne, le, false, empty, not, lt, ge, null, div* y *mod*.