JavaBeans

Conceptos básicos

Al diseñar aplicaciones web se plantea como objeto primordial crear un código bien estructurado, flexible y reutilizable. Que suponga un mínimo esfuerzo hacer evolucionar una aplicación y posibilitar el desarrollo de otras partiendo de código ya diseñado previamente. A esto se une el hecho de que el código debe ser fácilmente legible, lo que supone elaborar páginas JSP con el mínimo código Java posible. Además se añade la necesidad de diseñar separadamente todo lo que es código asociado a la presentación de datos enviada al usuario de lo que es puramente procesamiento o lógica de negocio.

Los JavaBeans[41] proporcionan ocultación de información accesible mediante etiquetas JSP, siguiendo un estándar prefijado de nomenclatura.

Definimos un JavaBean como una clase Java formada por un conjunto de propiedades y métodos, donde estos últimos están orientados a dar acceso a las primeras. Implementan la interfaz *java.io.Serializable*, con el propósito de que el contenedor web pueda almacenarlos en disco si así lo necesitara. Y posee un constructor sin argumentos. Todo el acceso a las propiedades[42] de un *bean* se lleva a cabo mediante sus correspondientes métodos de acceso para escritura y para lectura.

JSP proporciona las etiquetas necesarias para poder hacer uso de los JavaBeans. En el patrón *MVC* suelen emplearse para almacenar el modelo, o estructura de datos con la que trabaja la aplicación.

Los JavaBeans ofrecen básicamente dos grupos de métodos: los métodos *getter* y los métodos *setter*[43]. El método de lectura para una propiedad del bean debe seguir la siguiente nomenclatura: *get<NombrePropiedad>()* Por ejemplo, para la propiedad *usuario*, el método de acceso al valor de la propiedad debe denominarse *getUsuario()*. Sin argumentos, y con la primera letra del nombre de la propiedad en mayúsculas.

De la misma manera, el método que permite modificar el estado de una propiedad debe seguir la nomenclatura: *set<NombrePropiedad>()*. Por ejemplo, y siguiendo el caso anterior, el método *setter* correspondiente será *setUsuario(<argumentos>)*, donde *<argumentos>* representa el conjunto de argumentos necesario para este método, y que dependerá de la implementación de la propiedad *usuario*.

Cómo usar un JavaBean en páginas JSP

A los JavaBeans se puede acceder tanto desde un scriptlet como desde las etiquetas JSP orientadas a su uso. Independientemente del mecanismo empleado para acceder a los JavaBeans es necesario declararlo e inicializarlo en las páginas JSP, al principio de ellas. Para ello se emplea la etiqueta *<jsp:useBean>*.

La sintaxis es:

```
<jsp:useBean id="<nombreBean>" scope="<ámbito>"  <EspecificaciónDeTipo> />

o bien

<jsp:useBean id="<nombreBean>" scope="<ámbito>"  <EspecificaciónDeTipo> >
      <cuerpo>
</jsp:useBean>
```

41 Para abreviar, también se los denomina simplemente bean.
42 Las propiedades deben ser atributos privados.
43 Efectivamente, ya llevamos un tiempo haciendo uso de ellos, pero ahora veremos realmente la utilidad "real" que poseen y el papel clave que jugarán como métodos pertenecientes a objetos JavaBeans.

donde:

- *<nombreBean>*: Es el nombre que recibirá el bean y con el que lo identificará cualquier código posterior en la página JSP. Se distingue entre mayúsculas y minúsculas.
- *<ámbito>*: Es el ámbito en el que se enmarca el bean. Las opciones válidas son *page*[44], *session*, *request* y [45]. Los conceptos de estos ámbitos ya los hemos abordado previamente. El bean, en función del ámbito, se almacenará como atributo dentro del objeto *session*, *request* o *application* correspondiente. Por tanto, puede accederse a un bean en cualquiera de estos ámbitos con los métodos *getAttribute()* y *setAttribute()*.
- *<EspecificaciónDetipo>*: Consiste en una combinación de los parámetros *type*, *class* y *beanName*. Al menos debe usarse *class* o *type*, y *beanName* es incompatible con el uso de *class*. La especificación de tipo permite crear nuevos beans, reutilizar beans serializados o incorporar beans ya existentes al esquema de beans de JSP. Por ahora nos basta con saber cómo usar *class*. Si se emplea este parámetro, entonces se crea e inicializa un bean perteneciente a la clase especificada si no existía previamente, o se recupera si ya había sido creado con anterioridad.
- *<cuerpo>:* Puede contener etiquetas *<jsp:setProperty>* y scriptlets, y sólo se procesa si el bean acaba de crearse en el ámbito de la página actual.

Acceso a las propiedades de un Bean

Se emplean dos etiquetas JSP para acceder a las propiedades de un bean. Con *<jsp:setProperty>* se asigna valor a una propiedad y con *<jsp:getProperty>* se accede al valor de ella. La sintaxis de la primera puede ser una de las siguientes:

```
<jsp:setProperty name="<nombBean>" property="<nombPropiedad>" value="<valor>"/>
<jsp:setProperty name="<nombBean>" property="<nombPropiedad>" param="<param>"/>
<jsp:setProperty name="<nombBean>" property="<nombPropiedad>"/>
<jsp:setProperty name="<nombBean>" property="*"/>
```

donde

- *<nombBean>*: Es el nombre del Bean. Se debe haber especificado previamente en una etiqueta *<jsp:useBean>* en su *id*.
- *<nombPropiedad>*: Nombre de la propiedad. Si se emplea *"*"* se compara cada propiedad del bean con los parámetros de la petición, y por cada coincidencia se produce una asignación através del método *set* correspondiente. Cada propiedad debe tener su correspondiente método *setter*. Si el atributo *property* es *"*"* o no se especifica los parámetros *value* ni *param*, y el parámetro de la petición es *null* o la cadena está vacía, la propiedad no se verá modificada.
- *<valor>*: Especifica el valor a asignar a la propiedad. Si no se emplea, se asignará el valor del parámetro de la petición especificado. El valor puede ser una expresión o una cadena, en cuyo caso la propiedad del bean puede ser un String, boolean, byte, char, double, int, float, long, Boolean, Byte, Char, Double, Integer, Float o Long.
- *<param>*: Se asigna a la propiedad del bean el valor del nombre del parámetro de la petición especificado en este atributo.

La sintaxis de la etiqueta *<jsp:getProperty>* es la siguiente:

```
<jsp:getProperty name="<nombBean>" property="<nombPropiedad>" />
```

El nombre de la propiedad debe ser una cadena, nunca una expresión que se evalúe en tiempo de ejecución. Esta etiqueta pasa a cadena el valor de la propiedad y lo envía al flujo de salida.

---

44 Ámbito por defecto.

45 Ámbito de nivel aplicación (instancia del servlet) lo que supone que los beans que pertenezcan a este ámbito estarán accesibles para todas las peticiones que se efectúen sobre la aplicación web, independientemente de la sesión a la que pertenezcan estas peticiones.

Serialización y persistencia
Las clases Java que se manipularán como JavaBeans requieren ser implementadas mediante la interfaz java.io.Serializable. Esto permitirá que el contenedor web almacene el estado del bean con el propósito de optimizar su uso dentro del servidor de aplicaciones. Este mecanismo permite hacer persistente los objetos, almacenándolos en archivos o transmitiéndolos por la red. El proceso contrario recibe el nombre de *deserialización*.

Otros tipos de métodos
Aunque normalmente se emplean métodos *setter* y *getter* al usar los beans, pueden utilizarse métodos de otros tipos. La funcionalidad de estos métodos puede ser la que se desee. Además pueden utilizarse métodos *get* y *set* no asociados a propiedades del bean. Todo esto lo veremos en próximos ejemplos.

El bean DatosRegistro

En los ejemplos siguientes se va a trabajar con el bean *DatosRegistro*. Este bean encapsula los datos correspondientes a un usuario, *nombre* y *clave*. Se proporcionan métodos de acceso a estas propiedades y además los métodos *estado()* y *getExisteLogin()*, que comentaré más adelante. La clase del bean pertenece al paquete *beans*[46] e implementa la interfaz *Serializable*.

DatosRegistro.java

```
1.  package beans;
2.
3.  import java.io.*;
4.  import java.util.*;
5.
6.  /**
7.   * Encapsula los datos de registro de un usuario, consistente en un nombre
8.   * de usuario (equivale al concepto de login) y una clave. Implementa la interfaz
9.   * Serializable con objeto de que pueda almacenarse el estado a través del
10.  * contenedor web.
11.  */
12. public class DatosRegistro implements Serializable
13. {
14.
15.  /**
16.   * Contiene el nombre de usuario. Tiene la función de login.
17.   */
18.  private String nombre = "ninguno";
19.
20.  /**
21.   * Clave empleada por el usuario para acceder a servicios.
22.   */
23.  private String clave = "ninguna";
24.
25.  /**
26.   * Constructor sin argumentos. Necesario en el uso de JavaBeans.
27.   */
28.  public DatosRegistro()
29.  {
30.  }
31.
32.  /**
33.   * Obtiene el valor de la propiedad nombre.
34.   * @return Cadena con el nombre de usuario.
35.   */
36.  public String getNombre()
37.  {
38.    return nombre;
39.  }
40.  /**
41.   * Establece el valor de la propiedad nombre.
42.   * @param nombre  El nombre del usuario.
43.   */
```

46 Puede tener el nombre que deseemos.

```
44.  public void setNombre (String nombre)
45.  {
46.    this.nombre = nombre;
47.  }
48.  /**
49.   * Obtiene el valor de la propiedad clave.
50.   * @return La clave del usuario.
51.   */
52.  public String getClave()
53.  {
54.    return clave;
55.  }
56.  /**
57.   * Establece el valor de la propiedad clave.
58.   * @param clave Clave de usuario.
59.   */
60.  public void setClave(String clave)
61.  {
62.    this.clave = clave;
63.  }
64.  /**
65.   * Comprueba si el nombre de usuario almacenado en la propiedad nombre
66.   * está ya almacenada en el archivo /home/alumno/login.dat. Este archivo
67.   * sólo contiene logins (nombre de usuario). Está serializado mediante
68.   * un array de String.
69.   * @return true Si existe ya ese nombre de usuario.
70.   * @return false Si no se ha encontrado.
71.   */
72.  public boolean getExisteLogin()
73.  {
74.        ArrayList listaLogin = null;
75.        ObjectInputStream ois = null;
76.        File archivoLogin = new File ("/home/alumno/login.dat");
77.        if (archivoLogin.exists()) {
78.          try {
79.            ois = new ObjectInputStream(new FileInputStream(archivoLogin));
80.            listaLogin = (ArrayList) ois.readObject();
81.            for (int i = 0; i<listaLogin.size(); i++) {
82.              if (((String) listaLogin.get(i)).equalsIgnoreCase(nombre)) {
83.                return true;
84.              }
85.            }
86.          }
87.          catch (Exception e)  //Dada cualquier excepción
88.          {
89.            return false;
90.          }
91.        }
92.      return false;
93.  }
94.
95.  /**
96.   * Añade un nombre de usuario al archivo
97.   * de login. No hace uso de la variable nombre.
98.   * @param login Nombre que se añadirá al archivo.
99.   */
100. public void setAddLogin(String login)
101. {
102.   ArrayList listaLogin;
103.
104.   ObjectOutputStream oos = null;
105.   ObjectInputStream ois = null;
106.
107.   File archivoLogin = new File ("/home/alumno/login.dat");
108.   try
109.   {
110.     if (archivoLogin.exists()) {
111.       ois = new ObjectInputStream(new FileInputStream(archivoLogin));
112.       listaLogin = (ArrayList) ois.readObject();
113.     }
```

```
114.      else
115.      {
116.        listaLogin = new ArrayList();
117.      }
118.      listaLogin.add(login);
119.      oos = new ObjectOutputStream(new FileOutputStream(archivoLogin));
120.      oos.writeObject(listaLogin);
121.    }
122.
123.    catch (FileNotFoundException fne)
124.    {
125.      System.out.println("Error, archivo no encontrado.");
126.    }
127.    catch (ClassNotFoundException cnfe)
128.    {
129.      System.out.println("Error, clase no encontrada al leer archivo de login.");
130.    }
131.    catch (IOException ioe)
132.    {
133.      System.out.println("Error de E/S en proceso setAddLogin()");
134.    }
135.  }
136.  /**
137.   * Devuelve un array de dos String conteniendo el primer elemento el valor
138.   * de la propiedad nombre y el segundo el valor de la propiedad clave.
139.   * @return Array de dos cadenas con nombre y clave respectivamente.
140.   */
141.  public String[] estado()
142.  {
143.    String[] estado = new String[2];
144.    estado[0] = nombre;
145.    estado[1] = clave;
146.    return estado;
147.  }
148.}
```

Como puede observarse la clase *DatosRegistro* proporciona la funcionalidad asociada al concepto de datos de usuario registrados en un servicio. En este caso los datos son simplemente el nombre del usuario, que hará la función de login, y la clave asociada al mismo. Los beans obtenidos a partir de esta clase (instancias de la misma) podrán ser serializados por el contenedor web si así lo considerara oportuno éste (o nosotros). Además de los métodos *get* y *set* para acceder a las propiedades *nombre* y *clave*, se incluyen otros tres métodos. El primero, *getExisteLogin()* permite comprobar si el nombre de usuario existe en el archivo */home/alumno/login.dat*[47], lo que permitirá añadir el usuario al registro de usuarios o bien considerar que ya está registrado. El método *estado()* obtiene los valores de las propiedades en forma de array de String. El tercer método, *setAddLogin()* añade un login de usuario al archivo de login de usuarios *login.dat*. Este método no recurre directamente a la propiedad *nombre*, sino que toma un String como argumento de entrada, que se añadirá al archivo. Se hace así ya que se emplea este método como ejemplo de uso de etiqueta *<jsp:setProperty>* no asociada a propiedad, pero requiere un parámetro *value*. Estos métodos se incluyen con el propósito de mostrar que un JavaBean puede tener métodos diferentes a los *set* y *get* de sus propiedades.

Seguidamente planteo una batería de ejemplos en los que se emplean JavaBeans. Crearemos el proyecto *EjJavaBeans*.

Ejemplo 1

Este ejemplo muestra cómo usar un JavaBean sin pasar por el uso de etiquetas JSP. Este caso es el empleado cuando se trabaja con instancias de clases proporcionadas junto a la aplicación, y se trata de la manera general de uso de objetos en una página JSP. Por tanto, con este ejemplo, se comprueba que pueden crearse en la página JSP los objetos que se necesiten, simplemente usando el operador *new* y

47 Lo adecuado en este caso es proporcionar una propiedad más a la clase DatosRegistro que incorpore el nombre del archivo.

accediendo a sus propiedades y métodos como se hace en un código Java normal.

El ejemplo se ejecuta apartir de un formulario, *ej1JavaBean.html*, que proporciona un nombre de usuario y una contraseña. Estos datos se envían a la página *ej1JavaBean.jsp*, que creará un objeto del tipo *DatosRegistro*, y le asignará los valores recibidos en la petición, mostrándolos para finalizar.

El código es el siguiente:

ej1JavaBean.html

```
1.  <html>
2.    <head>
3.      <meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-15"></meta>
4.      <title>Ejemplo 1 de JavaBeans</title>
5.    </head>
6.    <body>
7.      <form name="Enviar" action="ej1JavaBean.jsp" method="post">
8.        <P>Introduzca nombre:
9.        <input type="text" name="nombre"/>
10.       </P>
11.       <P>Introduzca clave:
12.       <input type="password" name="clave"/></P>
13.       <P>
14.         <input type="submit" value="enviar"/>
15.       </P>
16.     </form>
17.   </body>
18. </html>
```

ej1JavaBean.jsp

```
1.  <%@ page contentType="text/html;charset=ISO-8859-15"%>
2.  <html>
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5.      <title>Ejemplo 1 de JavaBeans</title>
6.    </head>
7.    <body>
8.    Ejemplo de uso de un JavaBean sin usar las etiquetas JSP.
9.    <br>
10.   <% beans.DatosRegistro dr = new beans.DatosRegistro(); %>
11.   Los datos actuales del registro son:<br>
12.   Nombre : <%= dr.getNombre()%><br>
13.   Clave : <%= dr.getClave()%><br>
14.   Se establecen nuevos valores:<br>
15.   <%
16.     dr.setNombre(request.getParameter("nombre"));
17.     dr.setClave(request.getParameter("clave"));
18.   %>
19.   Nombre : <%= dr.getNombre()%><br>
20.   Clave : <%= dr.getClave()%>
21.   </body>
22. </html>
```

Como se observa, se usa el JavaBean como un objeto normal de Java, de hecho, cualquier objeto de Java puede considerarse como un JavaBean siempre que cumpla con las condiciones comentadas en el primer apartado de este tema.

En este código no es fundamental que los métodos de acceso a las propiedades tengan una sintaxis determinada, ya que se usan nombrando a los métodos explícitamente, al contrario que en las etiquetas JSP correspondientes.

Ejemplo 2

El código es:

ej2JavaBean.jsp

```
1.  <%@ page contentType="text/html;charset=ISO-8859-15" %>
2.  <html>
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5.      <title>Ejemplo 2 de JavaBeans</title>
6.    </head>
7.    <body>
8.      Empleando un JavaBean mediante sus propias etiquetas JSP
9.      <jsp:useBean id="dr" class="beans.DatosRegistro"/>
10.     Los datos actuales del registro son:<br>
11.     Nombre: <jsp:getProperty name="dr" property="nombre" />
12.     <br>
13.     Clave:<jsp:getProperty name="dr" property="clave" />
14.     <br>
15.     Se cambian los datos del registro a:<br>
16.     <jsp:setProperty name="dr" property="nombre" value="Frolik"/>
17.     <jsp:setProperty name="dr" property="clave" value="secreta"/>
18.     Nombre:<jsp:getProperty name="dr" property="nombre" />
19.     <br>
20.     Clave:<jsp:getProperty name="dr" property="clave" />
21.     <br>
22.     Este login ¿existe ya? :
23.     <jsp:getProperty name="dr" property="existeLogin" />
24.     <br>
25.     Estado actual del JavaBean:<br>
26.     <%
27.       String[] estado = dr.estado();
28.     %>
29.     <%= estado[0]%><br>
30.     <%= estado[1]%><br>
31.     <% if (!dr.getExisteLogin())
32.          {
33.     %>
34.       <jsp:setProperty name="dr" property="addLogin" value="<%=dr.getNombre()%>"/>
35.     <%
36.          }
37.     %>
38.   </body>
39. </html>
```

```
9.      <jsp:useBean id="dr" class="beans.DatosRegistro"/>
```

Se emplea la etiqueta *<jsp:useBean>* para definir e inicializar un Bean. Se nombra al Bean como *dr*, y se indica que será una instancia de la clase *beans.DatosRegistro*. En este llamada se crea el código fuente siguiente:

```
if ((dr = (beans.DatosRegistro) pageContext.getAttribute( "dr", PageContext.PAGE_SCOPE)) == null) {
        dr = (beans.DatosRegistro) new beans.DatosRegistro();
        pageContext.setAttribute( "dr", dr, PageContext.PAGE_SCOPE);
      }
```

En este código se intenta obtener un atributo, a nivel del contexto establecido en *PageContext.PAGE_SCOPE*[48], que se llame *dr*. Si no existe lo crea, mediante el uso del operador *new* y lo asigna a *dr*. En caso contrario, simplemente, lo asigna a *dr*. Esto significa que cuando se emplea la etiqueta *<jsp:useBean>* si el bean ya existía no lo vuelve a crear. Esto sucede habitualmente cuando se crea en una página y luego ésta hace un reenvío a otra. Entonces, esta otra ya tiene disponible el Bean, si aquel se incluyó en el ámbito *request*, o superior. Como consecuencia, se debe recordar que los beans, se almacenan en uno de los cuatro posibles ámbitos de ejecución de una página JSP.

48 Recordar que el ámbito por defecto es *page*.

```
11.  Nombre: <jsp:getProperty name="dr" property="nombre" />
12.  <br>
13.  Clave:<jsp:getProperty name="dr" property="clave" />
14.  <br>
```

Se accede al valor de la propiedad *nombre* del bean *dr*. Esta etiqueta supone la ejecución del método *getNombre()*. El valor devuelto se muestra en el flujo de salida asociado a la página JSP, es decir, en el navegador del cliente que hizo la petición. Lo mismo se aplica a la propiedad *clave*.

```
16.  <jsp:setProperty name="dr" property="nombre" value="Frolik"/>
17.  <jsp:setProperty name="dr" property="clave" value="secreta"/>
18.  Nombre:<jsp:getProperty name="dr" property="nombre" />
19.  <br>
20.  Clave:<jsp:getProperty name="dr" property="clave" />
21.  <br>
```

Se cambian los valores de las propiedades. En el primer caso se asigna el valor *"Frolik"* a la propiedad *nombre* del bean *dr*. Se emplea el método *setNombre()*. Y en el segundo caso, se procede de la misma manera pero con la propiedad *clave*.

Seguidamente, se vuelven a mostrar los valores de las propiedades, ya con los cambios aplicados, de nuevo mediante las etiquetas *<jsp:getProperty>*.

```
23.  <jsp:getProperty name="dr" property="existeLogin" />
```

Se llama al método *existeLogin()* para comprobar si ese usuario existía ya en el archivo *login.dat* o no. Es curiosa la forma en que se ejecuta este método. Se hace como si se estuviera accediendo al valor de una propiedad del bean. Si se indica como nombre de propiedad *existeLogin*, la etiqueta intentará ejecutar un método denominado *getExisteLogin()*. Aunque no exista la propiedad *existeLogin*, el método se ejecutará, ya que la única comprobación que hace la etiqueta es la del nombre del método, no que exista esa propiedad. Por supuesto, como el método devuelve un valor, éste se muestra en el flujo de salida.

Es importante conocer bien la manera en que el contenedor web genera el código fuente de las etiquetas. Por ejemplo en el caso que nos ocupa, el código generado es :

out.print(dr.getExisteLogin());

Por tanto, se hace necesario recalcar que una etiqueta *<jsp:getProperty>* supone siempre que la llamada al método correspondiente se proporciona como argumento al método *out.print()*. Esto implica que no podemos usar esta etiqueta dentro de expresiones.

Se comprueba que dentro de un bean se puede trabajar con archivos como se hace en una aplicación Java *standalone*. El método *getExisteLogin()* accede sin dificultad a un archivo serializado en la ubicación establecida.

```
26.  <%
27.    String[] estado = dr.estado();
28.  %>
29.  <%= estado[0]%><br>
30.  <%= estado[1]%><br>
```

Se demuestra que una vez creado el Bean, puede accederse también directamente como si se hubiera creado mediante el uso del operador **new** dentro de un scriptlet. En el caso de ejemplo, se ejecuta el método *estado()* y se muestran los valores del array de String.

```
31. <% if (!dr.getExisteLogin())
32.      {
33.  %>
```

```
34.      <jsp:setProperty name="dr" property="addLogin" value="<%=dr.getNombre()%>"/>
35.    <%
36.      }
37.    %>
```

Y para finalizar, se ejecuta la etiqueta *<jsp:setProperty>* si el login no existe en el archivo de login, proporcionando como nombre de propiedad, el del método sin la partícula *set*, y como valor, el de la propiedad *nombre* del bean *dr*.

Ejemplo 3

ej3JavaBean.jsp

```
1.  <%@ page contentType="text/html;charset=ISO-8859-15"%>
2.  <html>
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5.      <title>Ejemplo 3a de JavaBean</title>
6.    </head>
7.    <body>
8.      <jsp:useBean id="dr" scope="session" class="beans.DatosRegistro"/>
9.      <jsp:setProperty name="dr" property="nombre" value="Ursula"/>
10.     <jsp:setProperty name="dr" property="clave" value="secreta"/>
11.     <jsp:forward page="ej4JavaBean.jsp" />
12.   </body>
13. </html>
```

ej4JavaBean.jsp

```
1.  <%@ page contentType="text/html;charset=ISO-8859-15"%>
2.  <html>
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5.      <title>Ejemplo 3b</title>
6.    </head>
7.    <body>
8.      <jsp:useBean id="dr" scope="session" class="beans.DatosRegistro"/> <br>
9.      <jsp:forward page="ej5JavaBean.jsp" />
10.   </body>
11. </html>
```

ej5JavaBean.jsp

```
1.  <%@ page contentType="text/html;charset=ISO-8859-15"%>
2.  <html>
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5.      <title>Ejemplo 3c</title>
6.    </head>
7.    <body>
8.      <jsp:useBean id="dr" scope="session" class="beans.DatosRegistro" />
9.      <jsp:setProperty name="dr" property="nombre" value="Eduardo"/>
10.     <jsp:getProperty name="dr" property="existeLogin" /><br>
11.     Estado actual del JavaBean en ej5:<br>
12.     <%
13.       String[] estado = dr.estado();
14.     %>
15.     <%= estado[0]%><br>
16.     <%= estado[1]%><br>
17.   </body>
18. </html>
```

Este ejemplo muestra cómo un bean puede transmitirse entre páginas JSP, estableciendo el ámbito bajo el que se almacena. En este caso se establece como ámbito el de sesión.

Esto se lleva a cabo en

```
8.    <jsp:useBean id="dr" scope="session" class="beans.DatosRegistro"/>
```

donde *scope* establece el ámbito de sesión.

Luego de crear e inicializar el bean a nivel de sesión, se establecen los valores de sus propiedades y finalmente, se produce un reenvío de la petición a la página *ej4JavaBean.jsp*. En esta página se vuelve a usar la etiqueta *<jsp:useBean>* pero se comprueba que realmente no crea de nuevo el bean, sino que como detecta que a nivel de sesión ya hay uno con el nombre de *dr*, simplemente lo recuperará. Se comprueba si existe el login proporcionado en el bean y se reenvía de nuevo la petición a la página *ej5JavaBean.jsp*. En esta página se le cambia el valor a la propiedad *nombre* y se vuelve a comprobar si el login cambiado existe. Finalmente se muestra el estado.

Ejemplo 4

ej7JavaBean.html

```
1.  <html>
2.    <head>
3.      <meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-15"></meta>
4.      <title>Ejemplo 4 de JavaBeans</title>
5.    </head>
6.    <body>
7.      <form name="Enviar" action="ej7JavaBean.jsp" method="post">
8.        <P>Introduzca nombre:
9.        <input type="text" name="nombre"/>
10.       </P>
11.       <P>Introduzca clave:
12.       <input type="password" name="clave"/></P>
13.       <P>
14.         <input type="submit" value="enviar"/>
15.       </P>
16.     </form>
17.   </body>
18. </html>
```

ej7JavaBean.jsp

```
1.  <%@ page contentType="text/html;charset=ISO-8859-15"%>
2.  <html>
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5.      <title>Ejemplo 4 de JavaBeans</title>
6.    </head>
7.    <body>
8.  Empleando un JavaBean mediante sus propias etiquetas JSP, partiendo de un formulario.
9.      <jsp:useBean id="dr" class="beans.DatosRegistro"/>
10.     Los datos actuales del registro son:<br>
11.     Nombre: <jsp:getProperty name="dr" property="nombre" />
12.     <br>
13.     Clave:<jsp:getProperty name="dr" property="clave" />
14.     <br>
15.     Se cambian los datos del registro a:<br>
16.     <jsp:setProperty name="dr" property="nombre" value="<%= request.getParameter(\"nombre\")%>"/>
17.     <jsp:setProperty name="dr" property="clave" value="<%= request.getParameter(\"clave\")%>"/>
18.     Nombre:<jsp:getProperty name="dr" property="nombre" />
19.     <br>
20.     Clave:<jsp:getProperty name="dr" property="clave" />
21.     <br>
22.     Este login ¿existe ya? :
23.     <jsp:getProperty name="dr" property="existeLogin" /><br>
24.     Estado actual del JavaBean:<br>
25.     <%
26.       String[] estado = dr.estado();
27.     %>
28.     <%= estado[0]%><br>
29.     <%= estado[1]%><br>
30.   </body>
```

```
31. </html>
```

En este ejemplo se envía mediante un formulario en la página *ej7JavaBean.html*, un nombre de usuario y una contraseña a la página *ej7JavaBean.jsp*, en la que mediante una etiqueta *<jsp:useBean>* se crea un bean de ámbito de página, y se asignan los valores recibidos desde el formulario a las propiedades del bean, mostrando esta información junto con la respuesta devuelta por la etiqueta que consulta si el nombre de usuario estaba ya registrado en el archivo de login.

Observamos que los nombres de los campos del formulario coinciden con los nombres de las propiedades del bean. Entonces se puede sustituir las etiquetas de asignación de valores a las propiedades del bean, *<jsp:setProperty>*, por esta otra, que realizará la misma función:

<jsp:setProperty name="dr" property="" />*

Ejemplo 5

ej8JavaBean.jsp

```
1.  <%@ page contentType="text/html;charset=ISO-8859-15"%>
2.  <html>
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5.      <title>Ejemplo 5 de JavaBeans</title>
6.    </head>
7.    <body>
8.      <!-- Ejemplo para comprobar los ámbitos page, request, session y application-->
9.      <%-- Ámbito de la sesión proporcionado por el atributo scope:
10.       * page: El JavaBean es válido exclusivamente durante la ejecución de la
11.         página JSP. Es el ámbito predeterminado.
12.       * request: Existe para la página JSP y para cualquier otra a la que se
13.         acceda mediante "<jsp:forward>" o "<jsp:include>".
14.       * session: Válido durante la ejecución de cualquier página JSP en la
15.         sesión actual.
16.       * application: Válido para cualquier página JSP de la aplicación web, es
17.         decir, que estén dentro del contexto de la aplicación.
18.     --%>
19.     <jsp:useBean id="dr" scope="application" class="beans.DatosRegistro" />
20.     <jsp:setProperty name="dr" property="nombre" value="bubu"/>
21.     <jsp:getProperty name="dr" property="existeLogin" /><br>
22.     <%--
23.     Estado actual del JavaBean en ej8JavaBean.jsp:<br>
24.     <%
25.       String[] estado = dr.estado();
26.     %>
27.     <%= estado[0]%><br>
28.     <%= estado[1]%><br>
29.     --%>
30.     <jsp:forward page="ej9JavaBean.jsp" />
31.   </body>
32. </html>
```

ej9JavaBean.jsp

```
1.  <%@ page contentType="text/html;charset=ISO-8859-15"%>
2.  <html>
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5.      <title>Ejemplo 5 de JavaBeans</title>
6.    </head>
7.    <body>
8.      Estado actual del JavaBean en ej9JavaBean.jsp:<br>
9.      <%
10.       String[] estado = {"-1","-1"};
11.       beans.DatosRegistro dr2 = (beans.DatosRegistro) application.getAttribute("dr");
12.       if (dr2!=null)
13.         estado = dr2.estado();
```

```
14.     %>
15.     <%= estado[0]%><br>
16.     <%= estado[1]%><br> <hr>
17.     <jsp:useBean id="dr" scope="application" class="beans.DatosRegistro" />
18.     Estado actual del JavaBean en ej9JavaBean.jsp:<br>
19.     <%
20.        estado = dr.estado();
21.     %>
22.     <%= estado[0]%><br>
23.     <%= estado[1]%><br>
24.     <%= session.getId()%><br>
25.   </body>
26. </html>
```

ej10JavaBean.jsp

```
1.  <%@ page contentType="text/html;charset=ISO-8859-15"%>
2.  <html>
3.    <head>
4.      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-15">
5.      <title>Ejemplo 5 de JavaBeans</title>
6.    </head>
7.    <body>
8.      <jsp:useBean id="dr" scope="session" class="beans.DatosRegistro">
9.        Se crea el bean por vez primera, se solicita incluir información:
10.       <jsp:include page="formulario.html" flush="true" />
11.     </jsp:useBean>
12.     <jsp:setProperty name="dr" property="*" />
13.     Estado actual del JavaBean en ej10JavaBean.jsp:<br>
14.     <%
15.        String[] estado = dr.estado();
16.     %>
17.     Nombre:<%= estado[0]%><br>
18.     Clave:<%= estado[1]%><br>
19.     Sesión ID:<%= session.getId()%><br>
20.   </body>
21. </html>
```

formulario.html

```
1.  <html>
2.    <head>
3.      <meta HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-15"></meta>
4.      <title>Ejemplo 5 de JavaBeans</title>
5.    </head>
6.    <body>
7.      <form action="ej10JavaBean.jsp" method=post>
8.        <P>Nombre:
9.        <input type="text" name="nombre"/></P>
10.       <P>Clave:
11.       <input type="password" name="clave" /></P>
12.       <P>
13.         <input type="submit" value="Enviar"/>
14.       </P>
15.     </form>
16.   </body>
17. </html>
```

Este ejemplo sirve para comprender el uso de los beans en los diferentes ámbitos de ejecución de las páginas JSP. En la página *ej8JavaBean.jsp*, se crea un bean de aplicación, se proporciona un valor a la propiedad *nombre*, dejando la propiedad *clave* con su valor por defecto. Seguidamente se imprime el resultado devuelto por el proceso de comprobación de existencia de login llevado a cabo con la etiqueta correspondiente y se muestra el estado del bean. Esto se visualiza si no se envía la petición a la página *ej9JavaBean.jsp*. Si se envía, que es tal como está diseñado el código inicialmente, esta información no aparecerá, y se mostrará la que visualice la página *ej9JavaBean.jsp*.

Al llegar la petición a la página *ej9JavaBean.jsp*, se ejecuta el siguiente scriptlet:

```
8.    Estado actual del JavaBean en ej9JavaBean.jsp:<br>
9.    <%
10.     String[] estado = {"-1","-1"};
11.     beans.DatosRegistro dr2 = (beans.DatosRegistro) application.getAttribute("dr");
12.     if (dr2!=null)
13.       estado = dr2.estado();
14.   %>
```

Este código inicializa el array *estado* con las cadenas *"-1"*. Luego accede al objeto que encapsula la información de nivel de aplicación, objeto *application*, y extrae el atributo *dr*, precisamente el bean que se creó en la página desde la que proviene la petición. Este atributo que se extrae se asigna al objeto *dr2*. Si este objeto no es *null*, como sucederá con el código de partida, se asignará a *estado* los valores de las propiedades del objeto *dr2*.

Con esto se muestra que los beans se almacenan como atributos dentro de los objetos que encapsulan la información asociada ál ámbito al que pertenecen, empleándose los métodos *getAttribute()*, *setAttribute()* y *removeAttribute()* para gestionarlos, tal como se hace con otros objetos dentro de las páginas JSP.

El código continúa imprimiendo el estado del bean, y luego empleando la etiqueta *<jsp:useBean>* para extraer el bean *dr*. Desde este punto, se dispone de dos objetos que reprentan el mismo bean, *dr2* y *dr*.

Se asigna a *estado* el estado actual del bean, y se finaliza imprimiendo esta información y la ID de la sesión creada.

Si cambiáramos ahora el ámbito del bean en la página *ej8JavaBean.jsp* del nivel de aplicación al nivel de petición, al realizar el reenvío a la página *ej9JavaBean.jsp*, ésta mostrará los valores por defecto del bean, ya que la etiqueta *<jsp:useBean>* no encuentra ninguno con ese nombre en el contexto de aplicación, procediendo a crear e inicializar un bean dentro de ese ámbito.

Finalmente, para mostrar el uso de procesos asociados a la creación de beans, se puede introducir el siguiente código como cuerpo en la etiqueta *<jsp:useBean>*:

```
<jsp:useBean id="dr" scope="application" class="beans.DatosRegistro">
  Se crea un nuevo bean <br>
</jsp:useBean>
```

Esto hará que cada vez que esta etiqueta proceda a crear un bean, muestre el mensaje *"Se crea un nuevo bean"*. Precisamente, si se ejecuta esta aplicación desde *ej8JavaBean.jsp*, tal como dejamos su código escrito en el párrafo anterior, comprobaremos que aparecerá este mensaje, pero si volvemos a cambiar el ámbito del bean creado en *ej8JavaBean.jsp* por el de aplicación, el mensaje ya no se mostrará, lo que indicará que el bean no se ha creado, sino que se ha recuperado partiendo del existente en el objeto *application*.

Por tanto, todo el código, tanto html, scriptlet o etiquetas JSP que se incluyan en el cuerpo de una etiqueta *<jsp:useBean>* tendrá efecto sólo en el caso en que ésta haya creado un bean.

En el cuerpo de estas etiquetas puede incluirse la inicialización a determinados valores de las propiedades del bean. Esto puede hacerse haciendo uso de un código similar al siguiente:

```
<jsp:useBean id="dr" scope="application" class="beans.DatosRegistro">
  <%
    String clave = "secreta";
    dr.setNombre("bubu");
  %>
  <jsp:setProperty name="dr" property="clave" value="<%=clave%>" />
  Se crea un nuevo bean con los siguientes datos: <br>
```

```
  <jsp:getProperty name="dr" property="nombre" />
  <%=dr.getClave()%>
</jsp:useBean>
```

En este código se muestran todas las posibilidades que ofrece este mecanismo, uso de scriptlets, etiquetas HTML, etiquetas JSP, no sólo a nivel de beans, ya que podría incluso emplearse una etiqueta *<jsp:include>* para mostrar, por ejemplo, un formulario en caso de que se haya creado el bean. Ejemplo:

```
<jsp:useBean id="dr" scope="application" class="beans.DatosRegistro">
  Introduzca los datos necesarios para continuar: <br>
  <jsp:include page="formulario.html" flush="true" />
</jsp:useBean>
```

Como muestra de este uso se ofrece el documento *formulario.html* y la página *ej10JavaBean.jsp*. Si se carga esta página por vez primera, se crea una nueva sesión y un nuevo bean. Entonces se muestra el formulario que si se rellena y se envía, será recibido en la misma página, pero esta vez, se mantiene la sesión y se recupera el bean creado en la fase anterior, dejándose de mostrar el formulario.

Otros usos de los JavaBeans

Se ha planteado el uso de los JavaBeans básicamente como elemento encapsulador de información recibida mediante parámetros a nivel de petición, teniendo como origen un formulario. También se ha comprobado que esta información puede ubicarse en otros ámbitos, por ejemplo en el de sesión. Sin embargo, un JavaBean puede crearse para encapsular información de error que será transmitida a páginas de gestión de errores[49], para reunir los datos necesarios para proceder a crear conexiones a bases de datos, para almacenar la información asociada a un registro de una base de datos, y en general como elemento agutinador de información que será procesada en páginas o procesos posteriores, o generalizando, como el mecanismo que posibilita reunir la información (estructura de datos junto con los procesos y datos correspondientes) con la que trabajan las aplicaciones web. Esta información además puede ser serializada permitiéndose su recuperación más allá del tiempo de vida de las propias aplicaciones.

49 Esto puede hacerse simplemente capturando la excepción, creado el bean, incluyéndolo por ejemplo en la sesión, y relanzando la excepción. Entonces se ejecutará la página de gestión de errores asociada a la página en la que se ha lanzado la excepción y lo primero que deberá hacerse será recuperar desde la sesión el bean de error.