Uso combinado de *JavaBeans, servlets*, y JSP

1. Introducción

Actualmente, el modelo más habitual para estructurar un servicio web es conocido por el término MVC, que corresponde a las siglas de *Model-View-Controller*. Representan las tres tareas normalmente involucradas en un servicio web: un control del flujo del servicio (C), un modelo de datos y lógica de operación (M), y una capa de presentación de los resultados (V).

El modelo pretende separar en lo posible los elementos que articulan esas tres tareas, asignando la tarea de control a un *servlet*, la gestión de la información del modelo a clases Java auxiliares (*beans*) y la presentación a páginas JSP.

2. Java Beans

Las *beans* son clases Java que encapsulan una determinada funcionalidad, ofreciendo un interfaz de acceso para consultar o modificar determinadas variables. Una *bean* debe cumplir las siguientes reglas:

- Debe tener un constructor sin argumentos.
- Debe tener un conjunto de variables internas, privadas, ocultas al exterior (por ejemplo, supongamos que una de ellas es *msg* del tipo *String*).
- Debe ofrecer un interfaz con métodos para consultar (métodos *get*) y modificar (métodos *set*) cada una de las anteriores propiedades. Por ejemplo, para la variable *msg*, debe ofrecer los métodos *getMsg()* y *setMsg(String m)*.

Por ejemplo, el siguiente sería el código de una *bean*. Estaría dentro de un fichero *AuxBean.java*, que debe (en nuestro ejemplo) estar situado en un directorio *Xbean* dentro del directorio *classes* del contexto de usuario del TOMCAT.

```
package Xbean;
public class AuxBean {
   private String msg;

   public AuxBean() {
      msg="hola mundo";
   }

   public String  getMsg() {
      return (msg);
   }

   public void setMsg(String m) {
      msg=m;
   }
}
```

Evidentemente, la *bean* puede contener otras variables no accesibles a través de este interfaz, y otros métodos para provocar los procesos involucrados en las consultas, al margen de los métodos de su interfaz como *bean*.

3. El *Servlet*

El *servlet* se encarga de llevar a cabo el control del flujo del servicio, identificando la consulta que se solicita, instanciando las *beans* que deben resolverla y transfiriendo el control a la página JSP más apropiada para mostrar el resultado.

El siguiente es el código de un *servlet* que crea una *bean* del anterior tipo la primera vez que se lo llama (método *doGet()*), mientras que en las ocasiones posteriores modifica el valor de la variable *msg* con la fecha y horas actuales.

Tras ello, introduce la *bean* como un atributo del objeto *request* y transfiere el control a la página JSP llamada "/*pag.jsp*".

```
import java.util.Date;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import Xbean.AuxBean;

public class ejBean extends HttpServlet {

   AuxBean ab;

   protected void doGet (HttpServletRequest request, HttpServletResponse response)
                              throws ServletException, IOException {

      ab = new AuxBean();
      ab.setMsg("Pulsa para ver el mensaje de la bean...");
      request.setAttribute("laBean", ab);

      ServletContext sc = getServletContext();
      RequestDispatcher rd = sc.getRequestDispatcher("/pag.jsp");
      rd.forward(request,response);
   }

   protected void doPost (HttpServletRequest request, HttpServletResponse response)
                              throws ServletException, IOException {

      Date fecha = new Date();
      ab.setMsg(fecha.toString());
      request.setAttribute("laBean", ab);

      ServletContext sc = getServletContext();
      RequestDispatcher rd = sc.getRequestDispatcher("/pag.jsp");
      rd.forward(request,response);
   }
}
```

Las variables de clase, declaradas fuera de los métodos *doGet* y *doPost* (por ejemplo, en este caso *ab*), son persistentes, es decir, conservan sus valores en posteriores solicitudes al mismo *servlet*. Esto es así ya que el *ServletContainer* sólo crea una instancia del mismo *servlet*, creando posteriormente un nuevo hilo para servir cada solicitud.

4. JSP (Java Server Pages)

Podemos encontrar introducciones al desarrollo con JSP en múltiples libros y abundantes manuales o tutoriales en Internet. Por ejemplo:

- Ejemplos con TOMCAT: http://jsp-servlet.net/tomcat_examples.html
- Tutorial JSP: http://www.jsptut.com/

La descripción completa de la JSP API se puede consultar en múltiples sitios de Internet (buscando simplemente "jsp api") y en concreto, en el servidor de Oracle.

4.1 Fundamentos de JSP

La página JSP se encarga de realizar la presentación al usuario, sirviendo como plantilla que se completará con ciertos valores obtenidos dinámicamente en el momento de la consulta (calculados por otros elementos del servicio antes de que le fuera transferido el control).

Para ello, contiene elementos de distintos tipos:

- Declaraciones (líneas que comienzan por "<%@").
- Fragmentos de código Java (líneas que comienzan por "<%").
- Expresiones (líneas que comienzan por "<%="), que se evalúan proporcionando un valor que se incrusta en ese mismo lugar.

En lugar de ser devuelta al cliente, la página JSP es convertida a un *servlet*, incluyendo el código Java contenido en ella y añadiendo una línea *out.println()* por cada línea restante. Ese *servlet* es compilado (sólo la primera vez que se solicita) y a continuación ejecutado, y sus resultados devueltos al cliente.

4.2 Página JSP genérica

Por ejemplo, el siguiente código se corresponde con la página JSP "*pag.jsp*" a la cual transfiere el control el *servlet* de la sección anterior, y que deberá estar en el directorio *webapps* del contexto del usuario, de acuerdo a lo especificado por el *servlet*.

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<%@page import="Xbean.AuxBean"%>

<html>
   <head>
      <title> Fecha y hora </title>
   </head>
   <body>
      <% AuxBean ab = (AuxBean)request.getAttribute("laBean"); %>
      <b>Valor actual de fecha y hora obtenida de la clase:</b>
       <%= ab.getMsg() %>

       <form action="ejBean" method="post">
          <input name="Enviar" type="submit" value="Actualizar">
       </form>
   </body>
</html>
```

Como vemos, la página JSP declara el uso de una clase llamada *Xbean.AuxBean*. Luego, en su interior, incluye fragmentos de código Java que recuperan la clase *AuxBean* introducida como atributo por el *servlet* en el objeto *request*, y emplea sus métodos para acceder a la información, que incrusta en la plantilla HTML que contiene la página JSP.

4.3 Página JSP usando *beans*

La anterior página JSP sirve en realidad para usar cualquier clase *AuxBean* que se le pase como atributo, sea o no una *bean*.

En el caso de que lo que se le pase sea una *bean* (cumple las reglas mencionadas anteriormente), es posible emplear un conjunto de elementos propios de JSP que simplifican la página. Este es el caso del siguiente código, que es funcionalmente equivalente al anterior.

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<html>
   <head>
      <jsp:useBean id="laBean" scope="session" class="Xbean.AuxBean" />
      <title> Fecha y hora </title>
   </head>
   <body>
      <b>Valor actual de fecha y hora obtenida de la bean: </B>
      <jsp:getProperty name="laBean" property="msg" /> <p>

      <form action="ejBean" method="post">
          <input name="Enviar" type="submit" value="Actualizar">
      </form>
   </body>
</html>
```

En este caso, a través del elemento *useBean* se declara la *bean* (y se crea si no existe), mientras que con el elemento *getProperty* se obtienen los valores de las variables que se desean (en este caso *msg*).

Igualmente, también existe un elemento *setProperty* para que la página JSP pueda cambiar los valores de las variables de la *bean*.

4.3 Página JSP usando el lengua EL

En una versión más reciente de JSP se ha introducido un lenguaje de expresiones (EL) para facilitar aún más la tarea de escribir páginas JSP.

Empleando este lenguaje de expresiones, se muestra a continuación una tercera versión de la página anterior, en la que se supone que se ha recibido una *bean* llamada *laBean* como atributo del objeto *request*.

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<html>
   <head>
       <title> Fecha y hora </title>
   </head>
   <body>
      <b>Valor actual de fecha y hora obtenida de la bean: </B>
      ${laBean.msg} <p>

      <form action="ejBean" method="post">
          <input name="Enviar" type="submit" value="Actualizar">
      </form>
   </body>
</html>
```

A diferencia de las primitivas básicas de las páginas JSP mencionados anteriormente (*useBean*, *getProperty* y *setProperty*), en este caso no se crea la *bean* en caso de que no exista, ni es posible modificar los valores de las variables de la *bean* recibida, sólo consultarlos. Por el contrario, el uso de EL permite trabajar fácilmente con listas y arrays.