JSP Básico

Índice

Copyright © 2006 Depto. CCIA All rights reserved.

1. Introducción a JSP

JSP (**JavaServer Pages**) es una tecnología que permite incluir código Java en páginas web. El denominado *contenedor JSP* (que sería un componente del servidor web) es el encargado de tomar la página, sustituir el código Java que contiene por el resultado de su ejecución, y enviarla al cliente. Así, se pueden diseñar fácilmente páginas con partes fijas y partes variables. El siguiente es un ejemplo muy sencillo de página JSP:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
<title>Mi primera página JSP</title>
</head>
<body>
<h1> Hoy es:
<%= new java.util.Date() %>
</h1>
</body>
</html>
```

Para ejecutar la página basta con colocarla en una aplicación web (por ejemplo, en Tomcat, dentro de `webapps/ROOT`). No es necesario que sea en un directorio específico como ocurre con los servlets, sino que puede ir en cualquier directorio en el que se colocaría normalmente un HTML.

La última versión de la especificación JSP es la 2.0, aunque es de reciente aparición (Tomcat 4.x implementa la versión anterior, la 1.2). Como se verá, es una especificación paralela al API de servlets, concretamente a la versión 2.3.

Aunque JSP y servlets parecen a primera vista tecnologías distintas, en realidad el servidor web traduce internamente el JSP a un servlet, lo compila y finalmente lo ejecuta cada vez que el cliente solicita la página JSP. Por ello, en principio, JSPs y servlets ofrecen la misma funcionalidad, aunque sus características los hacen apropiados para distinto tipo de tareas. Los JSP son mejores para generar páginas con gran parte de contenido estático. Un servlet que realice la misma función debe incluir gran cantidad de sentencias del tipo `out.println()` para producir el HTML. Por el contrario, los servlets son mejores en tareas que generen poca salida, datos binarios o páginas con gran parte de contenido variable. En proyectos más complejos, lo recomendable es combinar ambas tecnologías: los servlets para el procesamiento de información y los JSP para presentar los datos al cliente.

2. Traducción de los JSP a servlets

Como se ha comentado, la primera vez que se solicita una página JSP, el servidor genera el

Copyright © 2006 Depto. CCIA All rights reserved.

servlet equivalente, lo compila y lo ejecuta. Para las siguientes solicitudes, solo es necesario ejecutar el código compilado. El servlet generado de manera automática tiene un método `_jspService` que es el equivalente al `service` de los servlets "generados manualmente". En este método es donde se genera el código HTML, mediante instrucciones `println` y donde se ejecuta el código Java insertado en la página. Por ejemplo, la página **primera.jsp** podría generar un servlet con estructura similar al siguiente:

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse  response)
                        throws java.io.IOException, ServletException {
   JspWriter out = null;
   response.setContentType("text/html;ISO-8859-1");
   out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0
               Transitional//EN\">);
   out.println("<html>");
   out.println("<head>");
   out.println("<title>Mi primera pagina JSP</title>");
   out.println("</head>");
   out.println("<body>");
   out.print("Hoy es ");
   out.println(new java.util.Date());
   out.println("</body>");
   out.println("</html>");
}
```

El directorio donde se coloca el servlet generado, así como su nombre, dependen del servidor web. Por ejemplo, Tomcat utiliza su directorio `work/Catalina/localhost/`*`aplicacion_web`*. En caso de que la página esté en `ROOT`, el nombre de la aplicación se sustituye por un carácter de subrayado (_).

3. Elementos de JSP

Existen tres tipos de elementos JSP que podemos insertar en una página web:

- **Código**: podemos "incrustar" código Java de distintos tipos (declaraciones de variables y/o métodos, expresiones, sentencias) para que lo ejecute el contenedor JSP.
- **Directivas**: permiten controlar distintos parámetros del servlet resultante de la traducción automática del JSP
- **Acciones**: normalmente sirven para alterar el flujo normal de ejecución de la página (p.ej. redirecciones), aunque tienen usos variados.

Se pueden poner comentarios en una página JSP entre los símbolos `<%--` y `--%>`. El contenedor JSP ignorará todo lo contenido entre ambos. Dentro de los fragmentos de código Java también se pueden colocar comentarios siguiendo la sintaxis habitual del lenguaje.

Copyright © 2006 Depto. CCIA All rights reserved.

4. Inserción de código en páginas JSP

Hay tres formas de insertar código Java en una página JSP:

- **Expresiones** de la forma `<%= expresión %>`: en este caso, la expresión se evalúa, su resultado se convierte a `String` y se inserta en la salida.
- **Scriptlets** de la forma `<% código %>` : el código se ejecuta dentro del método `_jspService` del servlet generado.
- **Declaraciones** de la forma `<%! código %>`: se insertan en el cuerpo del servlet generado, fuera de sus métodos.

4.1. Expresiones

Como se ha visto, se evalúan, su resultado se convierte a un `String` y se escriben en la salida (el objeto predefinido `out`). La forma de traducir una expresión a código de servlet es imprimiéndola en `out` (mediante una sentencia `out.write(expresion)`) o similar.

4.2. Scriptlets

Permiten ejecutar código arbitrario, cuyo resultado no es necesario enviar a la salida. Si desde un *scriptlet* se desea escribir algo en ésta, bastará con utilizar el objeto predefinido `out`. Un uso común de los *scriptlets* es hacer que ciertas partes de código HTML aparezcan o no en función de una condición. Por ejemplo:

```
<%
  java.util.Calendar ahora = java.util.Calendar.getInstance();
  int hora = ahora.get(java.util.Calendar.HOUR_OF_DAY);
%>
<b> Hola mundo, <i>
<% if ((hora>20)||(hora<6)) { %>
     buenas noches
<% }
   else if ((hora>=6)&&(hora<=12)) { %>
          buenos días
<%       }
         else { %>
          buenas tardes
<%       } %>
</i> </b>
```

4.3. Declaraciones

Permiten definir variables o métodos que se insertarán dentro del cuerpo del servlet

Copyright © 2006 Depto. CCIA All rights reserved.

generado. Esto da la posibilidad de sobreescribir los métodos `jspInit` y `jspDestroy` que son el equivalente en JSP del `init` y `destroy` de los servlets. Las variables declaradas conservarán su valor entre sucesivas llamadas a la página, ya que son variables miembro del servlet y no locales al método `jspService`. Esto nos permite, por ejemplo, crear un contador de accesos a la página:

```
<%! private int accesos = 0; %>
<h1> Visitas: <%= ++accesos %> </h1>
```

4.4. Objetos implícitos de JSP

En cualquiera de estas tres formas, se puede hacer referencia a una serie de *objetos implícitos* , que se corresponden con objetos útiles del API de servlets (petición, respuesta, ...) y que en realidad son variables instanciadas de manera automática en el servlet generado a partir del JSP. Los objetos predefinidos en JSP se referencian en la tabla 1.

Objeto	Significado
`request`	el objeto `HttpServletRequest` asociado con la petición
`response`	el objeto `HttpServletResponse` asociado con la respuesta
`out`	el `Writer` empleado para enviar la salida al cliente. La salida de los JSP emplea un *buffer* que permite que se envíen cabeceras HTTP o códigos de estado aunque ya se haya empezado a escribir en la salida (`out` no es un `PrintWriter` sino un objeto de la clase especial `JspWriter`).
`session`	el objeto `HttpSession` asociado con la petición actual. En JSP, las sesiones se crean automáticamente, de modo que este objeto está instanciado aunque no se cree explícitamente una sesión.
`application`	el objeto `ServletContext`, común a todos los servlets de la aplicación web.
`config`	el objeto `ServletConfig`, empleado para leer parámetros de inicialización.
`pageContext`	permite acceder desde un único objeto a todos los demás objetos implícitos
`page`	referencia al propio servlet generado (tiene el

Copyright © 2006 Depto. CCIA All rights reserved.

	mismo valor que `this`).Como tal, en Java no tiene demasiado sentido utilizarla, pero está pensada para el caso en que se utilizara un lenguaje de programación distinto.
`exception`	Representa un error producido en la aplicación. Solo es accesible si la página se ha designado como página de error (mediante la directiva `page isErrorPage`).

5. Directivas de página

Las *directivas* influyen en la estructura que tendrá el servlet generado a partir de la página JSP. Hay tres tipos de directivas:

- `page`: tiene varios usos: importar clases de Java, fijar el tipo MIME de la respuesta, controlar el *buffer* de salida,...
- `include`: sirve para incluir código en la página *antes de que se realice la compilación del JSP*.
- `taglib`: se emplea cuando el JSP hace uso de etiquetas definidas por el usuario.

El formato genérico de una directiva es:

```
<%@ directiva atributo="valor" %>
```

algunas directivas admiten más de un atributo.

5.1. La directiva page

La tabla 2 recoge los distintos atributos que admite la directiva `page` y su significado.

Atributo	Significado	Ejemplo
`import`	el equivalente a una sentencia `import` de Java	`<%@ page import="java.util.Date" %>`
`contentType`	genera una cabecera HTTP `Content-Type`	`<%@ page contentType="text/plain" %>`
`isThreadSafe`	si es `false`, el servlet generado implementará el interface `SingleThreadModel` (ún único hilo para todas las peticiones). Por defecto, el valor es `true`.	

Copyright © 2006 Depto. CCIA All rights reserved.

`session`	Si es `false`, no se creará un objeto `session` de manera automática. Por defecto, es `true`.	
`buffer`	Define el tamaño del *buffer* para la salida (en kb), o `none` si no se desea *buffer*. Su existencia permite generar cabeceras HTTP o códigos de estado aunque ya se haya comenzado a escribir la salida.	`<%@ page buffer="64kb" %>`
`autoflush`	Si es `true` (valor por defecto), el buffer se envía automáticamente a la salida al llenarse. Si es `false`, al llenarse el buffer se genera una excepción.	
`extends`	Permite especificar de qué clase debe descender el servlet generado a partir de la página JSP. No es habitual cambiarlo.	
`info`	define una cadena que puede obtenerse a través del método `getServletInfo`	`<%@ page info="carro de la compra" %>`
`errorPage`	especifica la página JSP que debe procesar los errores generados y no capturados en la actual.	`<%@ page errorPage="error.jsp" %>`
`isErrorPage`	Si es `true`, indica que la página actúa como página de error para otro JSP. El valor por defecto es `false`.	
`language`	Permite especificar el lenguaje de programación usado en el JSP. En la práctica, el lenguaje siempre es Java, por lo que esta directiva no se usa.	
`pageEncoding`	define el juego de caracteres que usa la página. El valor por defecto es `ISO-8859-1`.	

Copyright © 2006 Depto. CCIA All rights reserved.

6. Acciones

En JSP existen varios mecanismos para incluir elementos externos en la página actual o redirigir la petición hacia otra página

- **La directiva `include`** permite insertar código en la página antes de que ésta se transforme en un servlet. De este modo se pueden reutilizar fragmentos de código JSP o HTML.
- **La acción `<jsp:include>`** permite insertar la salida de otra página JSP. Nótese que se incluye la *salida* generada por el código JSP, no el código propiamente dicho.
- **La acción `<jsp:plugin>`** permite incluir *applets* que hagan uso de Java 2.
- **La acción `<jsp:forward>`** sirve para redirigir la petición a otra página JSP

6.1. La directiva include

Es el equivalente al `#include` del lenguaje C. su sintaxis es:

```
<%@ include file="fichero" %>
```

Como el código se incluye en el servlet generado, los fragmentos de código incluidos pueden tener efecto sobre la página actual. Así, se puede utilizar esta directiva para definir constantes, generar cabeceras HTTP, ...

El problema de esta directiva es que el estándar no exige que el contenedor JSP detecte de manera automática los cambios en los ficheros incluidos, de manera que si cambia uno de ellos puede ser necesario forzar la recompilación de las páginas JSP que los incluyan.

La especificación JSP recomienda que si la página incluida no es una página JSP válida por sí sola (por ejemplo, porque utiliza variables que se confía que se hayan declarado previamente) se utilice la extensión "estándar" `.jspf` (JSP fragment) y se coloque en un directorio no público del contenedor JSP (por ejemplo, WEB-INF, que no es accesible desde el cliente, pero sí desde la directiva).

6.2. La acción <jsp:include>

Esta acción incluye en una página la salida generada por otra perteneciente a la misma aplicación web. La petición se redirige a la página incluida, y la respuesta que genera se incluye en la generada por la principal. Su sintaxis es:

```
<jsp:include page="URL relativa" flush="true|false"/>
```

El atributo `flush` especifica si el flujo de salida de la página principal debería ser enviado al cliente antes de enviar el de la página incluida. En JSP 1.2 este atributo es optativo, y su

Copyright © 2006 Depto. CCIA All rights reserved.

valor por defecto es `false`. En JSP 1.1 es obligatorio y siempre debía valer `true` (el forzar el vaciado de buffer era problemático porque una vez que ha sucedido esto no se pueden hacer redirecciones ni ir a páginas de error, ya que ya se han terminado de escribir las cabeceras).

Esta acción presenta la ventaja sobre la directiva del mismo nombre de que cambios en la página incluida no obligan a recompilar la "principal". No obstante, la página incluida solo tiene acceso al `JspWriter` de la "principal" y no puede generar cabeceras (por ejemplo, no puede crear *cookies*).

Por defecto, la petición que se le pasa a la página incluida es la original, pero se le pueden agregar parámetros adicionales, mediante la etiqueta `jsp:param`. Por ejemplo:

```
<jsp:include page="cabecera.jsp">
   <jsp:param name="color" value="YELLOW" />
</jsp:include>
```

6.3. La acción <jsp:plugin>

Esta acción sirve para incluir, de manera portable e independiente del navegador, *applets* que utilicen alguna librería de Java 2 (Swing, colecciones, Java 2D, ...), ya que las máquinas virtuales Java distribuidas con algunos navegadores relativamente antiguos (Explorer 5.x, Netscape 4.x,...) son de una versión anterior a Java 2.

6.4. La acción <jsp:forward>

Esta acción se utiliza para redirigir la petición hacia otra página JSP que esté en la misma aplicación web que la actual. Un ejemplo de su sintaxis básica es:

```
<jsp:forward page="principal.jsp"/>
```

La salida generada hasta el momento por la página actual se descarta (se borra el buffer). En caso de que no se utilizara buffer de salida, se produciría una excepción.

Al igual que en el caso de `<jsp:include>`, se pueden añadir parámetros a la petición original para que los reciba la nueva página JSP:

```
<jsp:forward page="principal.jsp">
   <jsp:param name="privilegios" value="root" />
</jsp:forward>
```

7. Servlets y JSPs

Los servlets y los JSPs son tecnologías complementarías. Cada una de ellas es más apropiada para realizar ciertas tareas. Por lo tanto, lo más adecuado será integrar ambas tecnologías, y

Copyright © 2006 Depto. CCIA All rights reserved.

realizar con cada una las tareas más apropiadas para ella.

Los servlets serán adecuados cuando se requiere mucha programación. Por el contrario, los JSPs serán más apropiados para generar la presentación en HTML. Por lo tanto, será conveniente combinar ambas tecnologías para separar el código y la presentación.

Podremos integrar ambas tecnologías, realizando la programación en el servlet, y redirigiendo al JSP adecuado para que produzca la presentación, utilizando el `RequestDispatcher` visto anteriormente, bien por el método `forward` o `include`.

Puede ser necesario que el servlet proporcione cierta información al JSP, ya que el servlet en el procesamiento puede haber producido ciertos datos que el JSP deberá presentar. Una forma para proporcionar estos datos es establecerlos como atributos en el objeto `ServletRequest`:

```
MiClase valor = generaDatos();
request.setAttribute("nombre", valor);
```

Este valor podrá ser cualquier objeto Java (`Object`). En el JSP podremos obtener dicho objeto de la petición:

```
<% MiClase valor = (MiClase)request.getAttribute("nombre"); %>
```

De esta forma pasaremos los datos para únicamente una petición determinada. Si queremos que estos datos estén disponibles para todas las peticiones que haga un mismo cliente a la aplicación, deberemos incluirlos en el ámbito de la sesión. Esto lo haremos en el *servlet* de la siguiente forma:

```
HttpSession session = request.getSession();
session.setAttribute("nombre", valor);
```

Este objeto podrá ser importado por el JSP mediante el objeto `session`:

```
<% MiClase valor = (MiClase)session.getAttribute("nombre"); %>
```

Podemos también incluir estos datos en ámbito del contexto de la aplicación si queremos que estos datos estén disponibles para todas las peticiones que se realicen a la aplicación de forma global. En este caso deberemos incluirlos en el ámbito del contexto de la siguiente forma dentro de nuestro *servlet*:

```
ServletContext context = getServletContext();
context.setAttribute("nombre", valor);
```

Este objeto podrá ser importado por el JSP mediante el objeto `application`:

```
<% MiClase valor = (MiClase)application.getAttribute("nombre"); %>
```


Copyright © 2006 Depto. CCIA All rights reserved.

Copyright © 2006 Depto. CCIA All rights reserved.