Programación Orientada a Objetos

mayo, 2003

SERVLETS Y JSP

Alfonso Cubero Moral

Sergio Luna García

Departamento de Informática y Automática

Universidad de Salamanca

Información de los autores:

Alfonso Cubero Moral

3º de Ingeniería Técnica en Informática de Sistemas

Facultad de Ciencias – Universidad de Salamanca

alfonso_betico@hotmail.com

Sergio Luna García

3º de Ingeniería Técnica en Informática de Sistemas

Facultad de Ciencias – Universidad de Salamanca

sergiolunag_@hotmail.com

Resumen

En este documento se pretenden mostrar dos de las principales tecnologías de generación dinámica de páginas Web: los *servlets* y las *JavaServer Pages (JSP)*.

El trabajo está dividido en tres partes fundamentalmente: una primera en la que se describen someramente las diferentes tecnologías que se usaron y se usan actualmente para generar contenido dinámico; una segunda en la que se explican detalladamente los *servlets:* sus principales características, su esquema básico de funcionamiento, paquetes *Java* para la utilización de *servlets* y las ventajas que aporta, sobre todo respecto a otra técnica muy usada llamada *Common Gateway Interface (CGI)*, además de algún sencillo ejemplo ilustrativo; y la última que está centrada en las páginas *JSP*: sus características y funcionamiento, ventajas que aporta, ejemplos.

Finalmente se exponen unas pequeñas conclusiones acerca de las ventajas-desventajas y utilización tanto de *servlets* como de *JSP*.

Abstract

This document tries to show two of the main technologies of dynamic generation of web sites: *Servlets and JavaServer Pages (JSP)*.

The work is divided into three main parts: the first one describes in a general way the different technologies used in the past and in current use to generate dynamic content; the second one explains detail by detail the *servlets*: their main characteristics, their basic working scheme, Java packages to *servlets* using and the given advantages, mainly respect to other technique very used called *Common Gateway Interface (CGI)*, moreover some easy illustrative example; and the last one is focus on the *JSP* pages: their characteristics and working method, given advantages, examples.

Finally this work presents a few conclusions about the advantages and disadvantages of the use of *servlets* and *JSP*.

Tabla de Contenidos

Índice de figuras

1. Introducción

Tradicionalmente las páginas Web mostraban información que cambiaba rara vez o nunca. Esta forma estática de mostrar información era bastante eficiente puesto que la página se creaba una única vez y se presentaba. Cuando era necesario se hacían mínimos cambios y ya estaba lista otra vez. Pero rápidamente surgió la necesidad de interactuar con el usuario y de adaptar la información a sus necesidades, o mostrar información que se toma de bases de datos que cambian frecuentemente. Con la forma que había de crear documentos HTML estáticos era imposible mantener esas páginas si se querían construir con cierto dinamismo. Por ello aparecieron las técnicas de generación dinámica de páginas. Estas técnicas permiten de forma relativamente fácil mantener actualizadas las páginas aunque se muestre información que cambia frecuentemente y también posibilitan formas de establecer comunicaciones personalizadas con los usuarios.

2. Diferentes tecnologías para Web dinámicas

La generación de contenido dinámico para la Web requiere que el servidor realice algún tipo de procesamiento adicional sobre la petición HTTP iniciada por el cliente, con el fin de generar una respuesta personalizada.

Como solución a esto, a lo largo del tiempo han ido surgiendo diferentes soluciones:

- *Common Gateway Interface (CGI)*: los primeros servidores HTTP no incluían ningún mecanismo para generar respuestas dinámicamente, por lo tanto se crearon interfaces para comunicar el servidor con programas externos que implementaran dicha funcionalidad. El esquema de funcionamiento de esta aproximación se ilustra en la Figura 1:

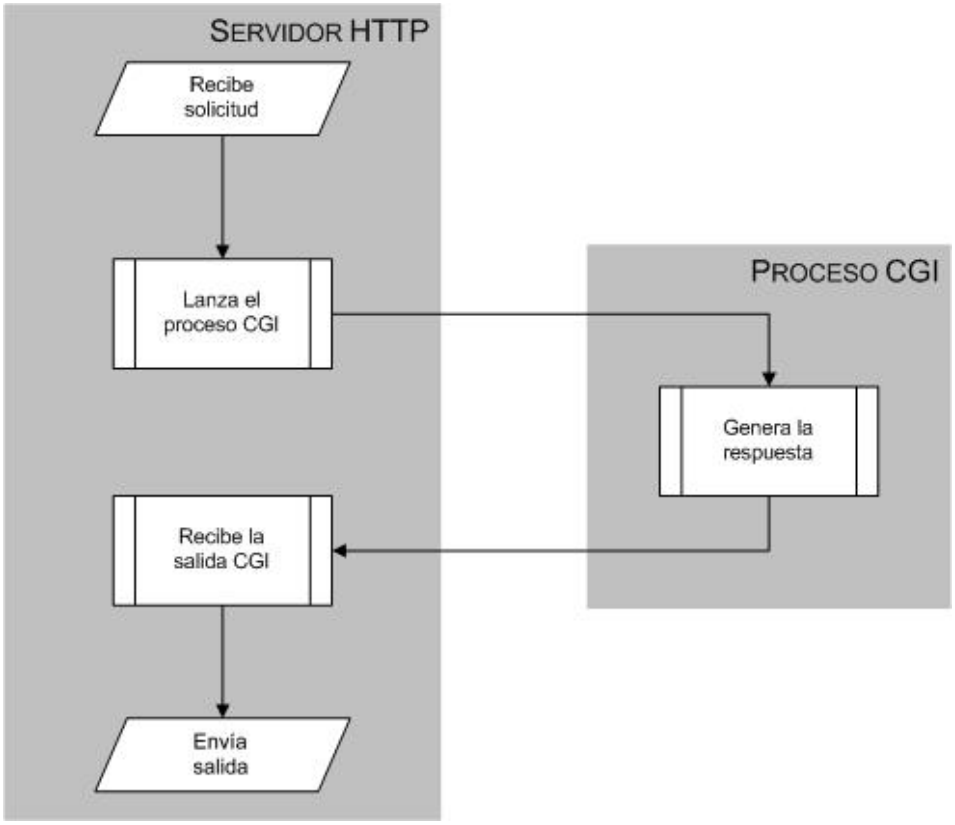


Figura 1. Esquema de funcionamiento de CGI

- *ColdFusion*: creado por Allaire, proporciona un conjunto de etiquetas similares a las de HTML, que encapsulan peticiones a bases de datos desde páginas Web. Actualmente, estas etiquetas se han extendido para soportar un amplio rango de orígenes de datos. Soporta plataformas UNIX y Microsoft Windows.

- *Active Server Pages (ASP)*: soporta múltiples lenguajes de *script*, entre los que se incluyen: PerlScript, JScript y VBScript. El lenguaje por defecto para ASP es VBScript (un subconjunto de Visual Basic). VBScript incluye soporte para acceso a componentes Active-X. La mayor limitación de ASP es que sólo está disponible para IIS (*Internet Information Server*) corriendo sobre el sistema operativo Windows NT. Aunque se han desarrollado herramientas para portar ASP a otras plataformas, la potencia de ASP está en el uso de objetos Active-X, que sólo están disponibles para plataformas Windows.

- *WebClass*: esta tecnología está fuertemente ligada a ASP. Se trata de un componente que agrega Visual Basic 6 como técnica de programación para Internet. Una WebClass es un componente que se ejecuta dentro de un IIS interceptando y procesando todas las peticiones que los clientes realizan a un documento ASP. La clase Web entra en funcionamiento cuando el explorador cliente hace referencia a una página en la aplicación y, desde ese instante, reacciona a las acciones del usuario realiza sobre la página.

- *Server-Side JavaScript (SSJS)*: el lenguaje que soporta esta tecnología es una extensión del núcleo de *JavaScript*. *JavaScript* es un lenguaje basado en prototipos, no en clases, donde se usan objetos pero no clases ni herencia. SSJS incorpora funcionalidades para soporte de bases de datos y correo electrónico, manejo de sesiones e interrelación con clases del lado servidor mediante la tecnología de Netscape LiveWire. SSJS es un lenguaje compilado, donde un conjunto de páginas Web que contienen SSJS se compilan en una aplicación Web que se ejecuta siempre que se solicita la URL asociada. El código compilado SSJS no depende de ninguna plataforma, aunque sí es específico de los servidores Web de Netscape.

- *Hipertext PreProcessor (PHP - Personal Home Page tools)*: es un producto de Open Source (gratuito). Proporciona grandes facilidades para acceso a bases de datos y manejo de patrones. También incorpora extensiones para comunicación con otros recursos de red como correo electrónico, servidores de directorio, SNMP e IMAP. PHP es compatible con Linux, Windows NT y varios sistemas operativos UNIX, así como con un amplio número de servidores Web como Apache, Microsoft IIS y Netscape Enterprise Server.

- *FastCGI*: FastCGI es la evolución natural de CGI en la que el programa CGI se conserva en memoria de forma persistente. Mediante FastCGI existe la posibilidad de que el programa CGI resida en máquinas diferentes a la del servidor Web, algo que conlleva ventajas a la hora de repartir la carga y fijar políticas de seguridad.

- *Java Servlets*: fueron introducidos por Sun en 1996 como pequeñas aplicaciones Java para añadir funcionalidad dinámica a los servidores Web. Los *servlets*, al igual que los *scripts* CGI, reciben una petición del cliente y generan los contenidos apropiados para su respuesta, aunque el esquema de funcionamiento es diferente.

- *JavaServer Pages (JSP)*: es una tecnología híbrida basada en template systems. Al igual que ASP, SSJS y PHP puede incorporar *scripts* para añadir código Java directamente a las páginas .jsp, pero también implemente, al estilo *ColdFusion*, un conjunto de etiquetas que interaccionan con los objetos Java del servidor, sin la necesidad de que aparezca código fuente en la página.

3. Servlets

Los *servlets* son módulos de código escrito en Java que añaden funcionalidad a un servidor Web. Fueron diseñados para aceptar peticiones de un cliente y generar los mensajes de respuesta correspondiente.

Los *servlets* no especifican ningún tipo de protocolo entre el cliente y el servidor, únicamente se limitan a recoger peticiones de usuario de una página HTML y generar respuestas.

3.1. Características

Entre las características principales de los *servlets* cabe citar las siguientes:

- Son independientes del servidor utilizado y de su sistema operativo, lo que quiere decir que a pesar de estar escritos en Java, el servidor puede estar escrito en cualquier lenguaje de programación.
- Los *servlets* pueden llamar a otros *servlets*, e incluso a métodos concretos de otros *servlets* (en la misma máquina o en una máquina remota). De esta forma se puede distribuir de forma más eficiente el trabajo a realizar. Por ejemplo, se podría tener un *servlet* encargado de la interacción con los clientes y que llamara a otro *servlet* para que a su vez se encargara de la comunicación con una base de datos.
- Los *servlets* pueden obtener fácilmente información acerca del cliente (la permitida por el protocolo HTTP), tal como su dirección IP, el puerto que se utiliza en la llamada, el método utilizado (GET, POST), etc.
- Permiten además la utilización de *cookies* y sesiones, de forma que se puede guardar información específica acerca de un usuario determinado, personalizando de esta forma la interacción cliente/servidor. Una clara aplicación es mantener la sesión con un cliente.
- Los *servlets* pueden actuar como enlace entre el cliente y una o varias bases de datos en arquitecturas cliente-servidor.
- Asimismo, pueden realizar tareas de *proxy* para un *applet*. Debido a las restricciones de seguridad, un *applet* no puede acceder directamente por ejemplo a un servidor de datos localizado en cualquier máquina remota, pero sí podría hacerlo a través de un *servlet*.
- Permiten la generación dinámica de código HTML, lo que se puede utilizar para la creación de contadores, *banners*, etc.

3.2. Funcionamiento

Como se ha mencionado en los puntos anteriores un *servlet* añade funcionalidad a una página Web. No siempre realizan las mismas tareas, es decir, no tienen un patrón fijo de trabajo, pero de forma general las podemos dividir de la siguiente forma:

1. Leer cualquier dato enviado por el usuario: los datos normalmente se introducen por medio de la página Web, pero también pueden obtenerse a partir de un *applet Java*.
2. Obtener otra información sobre la petición que se encuentra embebida en la propia petición HTTP: esta información se refiere por ejemplo a los *cookies*, el nombre del host de donde proviene la petición, etc.
3. Generar los resultados: esta parte puede requerir acceder a una base de datos, ejecutar una llamada RMI o CORBA, invocar a una aplicación o simplemente computar los datos de entrada.
4. Generar un documento con los resultados: debemos establecer el tipo de documento que va a ser devuelto (una página HTML, una imagen, un archivo comprimido, etc.).
5. Establecer los parámetros apropiados para la respuesta.
6. Enviar la respuesta al cliente: una vez que tenemos el formato del documento que entregaremos como respuesta y tenemos establecidos los parámetros de la comunicación enviamos la respuesta al cliente.

En la Figura 2 se muestra de forma genérica cómo funciona un *servlet*:

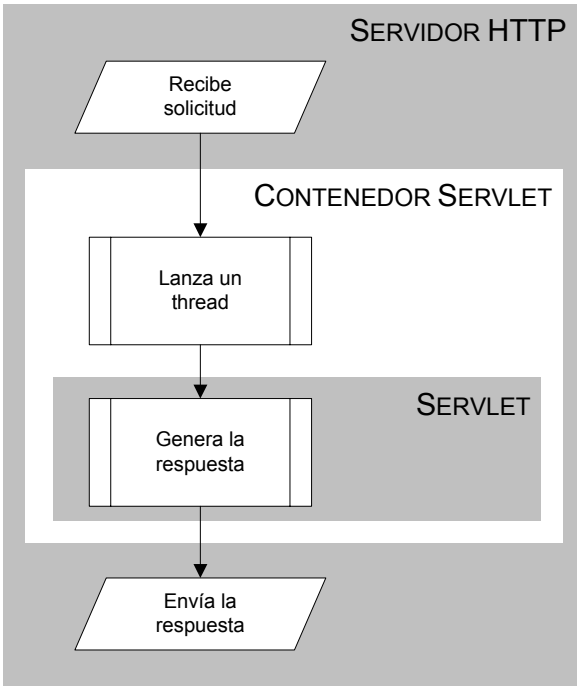


Figura 2. Esquema de funcionamiento de los servlets

3.3. Servlets VS CGI

Los primeros servidores HTTP no incluían ningún mecanismo para generar respuestas dinámicamente, por lo tanto se desarrollaron interfaces para comunicar el servidor con programas externos que implementan dicha funcionalidad.

La especificación *CGI* describe una interfaz estándar que sirve para que un servidor Web envíe solicitudes del navegador al programa *CGI*, y para que el programa *CGI* devuelva datos de respuesta al navegador a través del servidor Web.

Los *servlets* ofrecen la misma funcionalidad que los *CGI* pero evitan ciertos problemas que tenían éstos últimos. Entre las ventajas que ofrecen los *servlets* cabe destacar:

- Con *CGI* cada vez que se realizaba una petición se creaba un nuevo proceso para atenderla, con las consecuencias que conlleva (tiempo para el cambio de proceso, memoria ocupada, etc.). Esto no ocurre con los *servlets*, pues con cada petición no se genera un nuevo proceso, sino un nuevo hilo de ejecución.
- Si tenemos varias peticiones simultáneas y estamos utilizando *CGI* tendremos en memoria tantas instancias del programa como peticiones, lo que conlleva un gasto de memoria enorme si este número es muy grande y una degradación continua del rendimiento. Con los *servlets*, al tener hilos de ejecución, con una sola instancia de nuestro programa nos basta, ahorrando así muchos recursos.
- Cuando un programa *CGI* termina de responder a una petición, el programa finaliza. Esto hace que cada vez que se realiza una nueva petición todo el programa tiene que ser cargado de nuevo, establecer de nuevo las conexiones con las bases de datos, etc. Los *servlets*, sin embargo, permanecen en memoria entre peticiones, lo que hace que se gane mucho tiempo en las sucesivas peticiones porque el programa ya está cargado en memoria.
- Los *servlets* favorecen la independencia de la plataforma, ya que están escritos en *Java* y por tanto siguen el estándar API. Como consecuencia los *servlets* están escritos para ser ejecutados en una cantidad enorme de servidores (Apache, Microsoft IIS, Java Web Server, etc.).
- Un *servlet* puede ejecutarse en una *sandbox* o recinto de seguridad parecido al modelo que se sigue con los *applets*. Debido a esto pueden colocarse *servlets* en servidores dedicados a hosting sin que la empresa tema por la integridad del servidor y la seguridad de las aplicaciones.
- Una vez que tienes un servidor Web añadir un *servlet* supone un ligero incremento en el coste, a diferencia de los programas *CGI*, que necesitan una gran inversión para obtenerlos.

3.4. Paquetes Java para servlets

Existen una serie de paquetes escritos en *Java* que permiten trabajar de forma fácil y cómoda con los *servlets*: javax.servlet y javax.servlet.http.

El paquete javax.servlet define las siguientes clases e interfaces:

- Clases:
  - GenericServlet: implementa la interfaz Servlet.
  - ServletInputStream: se utiliza para acceder a información de las solicitudes Web.
  - ServletOutputStream: se utiliza para enviar la respuesta a un cliente.

- Interfaces:
  - Servlet: debe ser implementada por todos los *servlets*. Se utiliza para iniciar y detener el *servlet*.
  - ServletRequest: encapsula la solicitud de servicio de un cliente.
  - ServletResponse: es utilizada por el *servlet* para responder a una solicitud.
  - ServletConfig: para que el *servlet* conozca información del contenedor servlet.
  - ServletContext: define el entorno en el que se ejecuta un *servlet*.
  - SingleThreadModel: se usa para identificar a los *servlets* que no pueden ser accedidos por más de un hilo al mismo tiempo.

El paquete javax.servlet.http se usa para definir *servlets* específicos de HTTP y define las siguientes clases e interfaces:

- Clases:
  - Cookie: representa una *cookie* HTTP.
  - HttpServlet: amplía la clase GenericServlet con el fin de utilizar las interfaces HttpServletRequest y HttpServletResponse.
  - HttpSessionBindingEvent: implementa el evento que se genera cuando un objeto está ligado o se desliga de una sesión HTTP.
  - HttpUtils: analizar sintácticamente una cadena de consulta.

- Interfaces:
  - HttpServletRequest: amplía la interfaz ServletRequest agregando métodos para acceder a detalles de una solicitud HTTP.
  - HttpServletResponse: amplía la interfaz ServletResponse para devolver respuestas específicas de HTTP.
  - HttpSession: implementa los *servlets* que permiten dar soporte a las sesiones entre el navegador y el servidor.
  - HttpSessionBindingListener: la implementan clases cuyos objetos están asociados a sesiones HTTP.
  - HttpSessionContext: se utiliza para representar a una colección de objetos que están asociados a los identificadores de sesión.

3.5. Ejemplo de servlet

Este pequeño ejemplo muestra cómo un *servlet* procesa la información que introduce un usuario en un formulario contenido en una página HTML (Figura 3).

```
<body>

<FORM action="servlet/EchoRequest" method="POST"
                <P>
                NOMBRE: <BR> <INPUT type="text" name="Nombre">
                <P>
                E-MAIL: <BR> <INPUT type="text" name="Email">
                <P>
                TIPO DE INSCRIPCIÓN: <BR>
                                <INPUT type="checkbox" name="tipo1" value="Normal"> Normal <BR>
                                <INPUT type="checkbox" name="tipo2" value="Estudiante"> Estudiante <BR>
                                <INPUT type="checkbox" name="tipo3" value="Socio de ATI"> Socio de ATI
                <P>
                TIPO DE COMUNICACIÓN: <BR>
                                <INPUT type="radio" name="Comunicacion" value="paper"> Paper <BR>
                                <INPUT type="radio" name="Comunicacion" value="poster"> Poster <BR>
                                <INPUT type="radio" name="Comunicacion" value="demo"> Demo
                <P>
                ORGANIZACIÓN: <BR> <INPUT type="text" name="Organizacion">
                <P>
                PAÍS: <BR>
                    <SELECT size="5" name="Pais">
                                <OPTION>ESPAÑA</OPTION>
                                <OPTION>PORTUGAL</OPTION>
                                <OPTION>FRANCIA</OPTION>
                                <OPTION>ALEMANIA</OPTION>
                                <OPTION>ITALIA</OPTION>
                                <OPTION>REINO UNIDO</OPTION>
                                <OPTION>EEUU</OPTION>
                                <OPTION>AFRICA</OPTION>
                    </SELECT>
                <P>
                <INPUT type="submit" value="Realizar Inscripción">
                <INPUT type="reset" value="Borrar">
</FORM>

</body>
```

Figura 3. Código de formulario HTML

El usuario introduce los datos que le pide el formulario (Figura 4) y son enviados al *servlet* indicado en la línea que posee la etiqueta <FORM>.

Figura 4. Resultado de la carga del fichero HTML

El *servlet* (Figura 5) invocado desde la página HTML procesa los datos introducidos en el formulario. En concreto, aquí lo único que se hace es generar una salida (Figura 6) con los propios datos introducidos por el usuario.

```
import javax.servlet.*;
import java.util.*;
import java.io.*;

public class EchoRequest extends GenericServlet
{
        public String getServletInfo()
        {
                return "Echo Request Servlet";
        }

        public void service(ServletRequest peticion, ServletResponse respuesta)
                                                throws ServletException, IOException
        {
                respuesta.setContentType("text/plain");

                PrintStream outputStream = new PrintStream(respuesta.getOutputStream());
                outputStream.print("Servidor: " + peticion.getServerName() + ":");
                outputStream.println (peticion.getServerPort());
                outputStream.print("Cliente: " + peticion.getRemoteHost() + " ");
                outputStream.println(peticion.getRemoteAddr());
                outputStream.println("Protocolo: " + peticion.getProtocol());

                Enumeration params = peticion.getParameterNames();
                if (params != null)
                {
                        while (params.hasMoreElements())
                        {
                                String parametro = (String) params.nextElement();
                                String valor = peticion.getParameter(parametro);
                                outputStream.println(parametro + " = " + valor);
                        }
                }
        }
};
```

Figura 5. Código del servlet

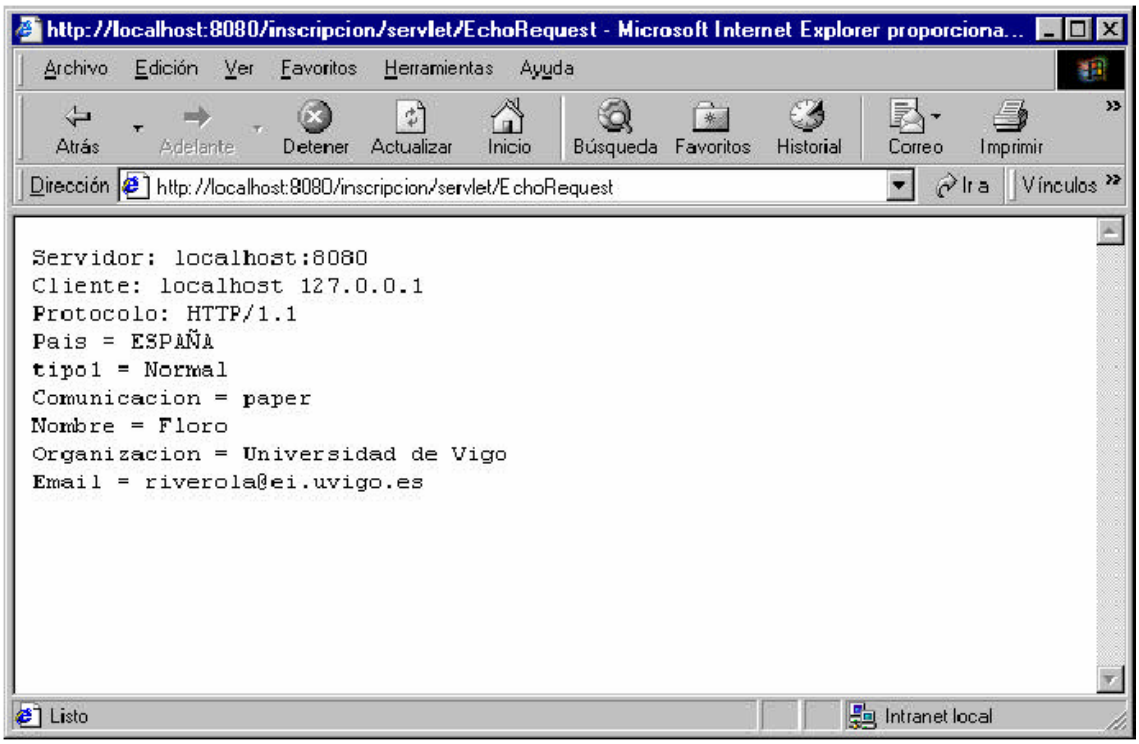


Figura 6. Salida generada por el servlet

4. JavaServer Pages (JSP)

La tecnología *JavaServer Pages (JSP)* provee una manera fácil para crear páginas Web dinámicas y simplificar la tarea de construir aplicaciones Web que trabajen con una amplia variedad de servidores Web, servidores de aplicaciones, navegadores y herramientas de desarrollo.

Pero ¿qué ES exactamente una *JavaServer Page*? Una página *JSP* es simplemente, en su forma básica, una página Web HTML que contiene unos bits adicionales de código que ejecutan la aplicación lógica para generar el contenido dinámico. Esta aplicación lógica puede envolver *JavaBeans*, objetos *JDBC*, *Enterprise JavaBeans (EJB)*, y objetos Remote Method Invocation (RMI), todos los cuales pueden ser fácilmente accedidos por una página *JSP*.

4.1. Características de JSP

Las características ofrecidas por *JSP* como alternativa a la generación de contenido dinámico para la Web se pueden resumir en:

- Mejoras en el rendimiento:
  - Utilización de procesos ligeros (hilos *Java*) para el manejo de las peticiones.
  - Manejo de múltiples peticiones sobre una página .jsp en un instante dado.
  - El contenedor servlet puede ser ejecutado como parte del servidor Web.
  - Facilidad para compartir recursos entre peticiones (hilos con el mismo padre: servlet container).

- Soporte de componentes reutilizables:
  - Creación, utilización y modificaciones de *JavaBeans* del servidor.
  - Los *JavaBeans* utilizados en páginas .jsp pueden ser utilizados en *servlets*, *applets* o aplicaciones *Java*.

- Separación entre código de presentación y código de implementación:
  - Cambios realizados en el código HTML relativos a cómo son mostrados los datos, no interfieren en la lógica de programación y viceversa.

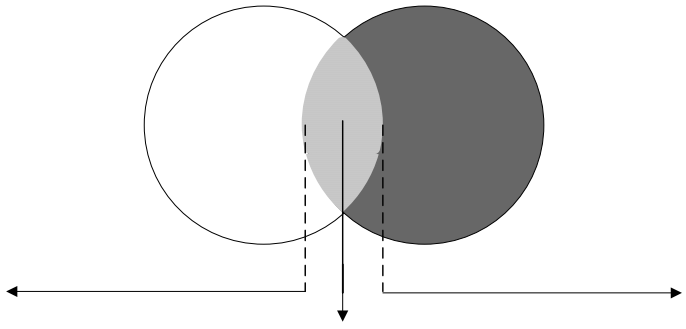


Figura 7. Separación entre código de presentación y código de implementación en JSP

- División del trabajo:
  - Los diseñadores de páginas pueden centrarse en el código HTML y los programadores en la lógica del programa.
  - Los desarrollos pueden hacerse independientemente.
  - Las frecuentes modificaciones de una página se realizan más eficientemente.

4.2. Funcionamiento

En primer lugar, para poder utilizar esta tecnología es necesario un servidor Web que de soporte a páginas .html, y código que implemente un contenedor JSP donde ejecutar las etiquetas *JSP*. Existen servidores Web que incorporan dicha capacidad dentro de su código, así como servidores escritos íntegramente en *Java* que dan soporte a esta tecnología directamente.

Sin embargo, para la mayoría de servidores Web es necesario añadir código suplementario que implemente el contenedor JSP. Para ello se han desarrollado API's del servidor para poder extender su funcionalidad y dar soporte a *JSP*.

Una vez que el contenedor JSP ha sido instalado y configurado, los ficheros .jsp se tratan igual que los ficheros .html, situándolos en cualquier lugar de la jerarquía de directorios. Cualquier clase *Java* que se utilice en un fichero .jsp, debe estar disponible en la variable CLASSPATH del contenedor JSP.

Aunque la especificación *JSP* no presupone nada sobre la implementación que da soporte a esta tecnología, la mayoría de las implementaciones disponibles están basadas en *servlets*.

El primer componente de las implementaciones basadas en *servlets*, es un *servlet* especial denominado Compilador de páginas. Este *servlet*, junto con sus clases *Java* asociadas, se conoce con el nombre de Contenedor JSP. El contenedor está configurado para llamar al compilador de páginas para todas las peticiones que coincidan con una página .jsp. Su misión es la de compilar cada página .jsp en un *servlet* cuya finalidad es la de generar el contenido dinámico especificado por el documento .jsp original.

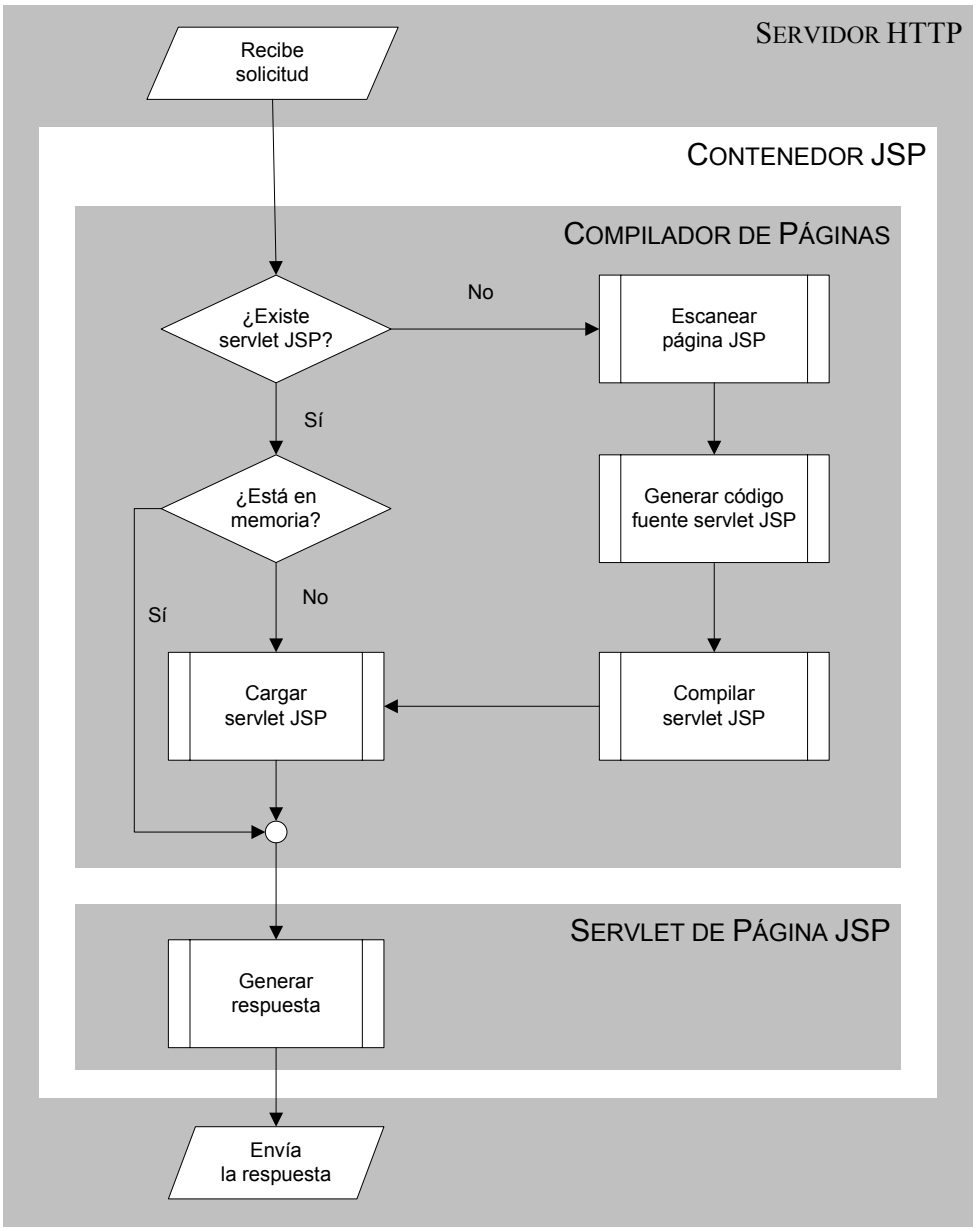


Figura 8. Esquema de ejecución de una página JSP

Para compilar una página, el compilador de páginas escanea el documento en busca de etiquetas *JSP*, generando el código *Java* correspondiente a cada una de ellas. Las etiquetas estáticas HTML son convertidas a Strings de *Java*. Una vez que el código del *servlet* ha sido construido, el compilador de páginas llama al compilador de *Java* para compilar el código fuente y añade el fichero de bytecodes resultante al directorio apropiado del contenedor JSP.

Una vez que el *servlet* correspondiente a la página .jsp ha sido generado, el compilador de páginas invoca al nuevo *servlet* para generar la respuesta al cliente. Mientras el código de la página .jsp no se altere, todas las referencias a la página las ejecutará el mismo *servlet*. Esto supone una cierta demora en la primera referencia a

la página, aunque existen mecanismos en *JSP* para precompilar la página .jsp antes de que se haya producido la primera petición.

4.3. Ventajas de JSP

JSP tiene unas cuantas ventajas sobre muchas de sus alternativas. Aquí van unas cuantas de ellas:

- Frente a HTML estático: el HTML normal no puede contener información dinámica, así que las páginas HTML no pueden estar basadas en la entrada del usuario o en fuentes de datos del lado del servidor. *JSP* es tan fácil y cómodo que es bastante razonable aumentar las páginas HTML, que sólo se benefician ligeramente por la inserción de datos dinámicos.

- Frente a *ASP*: *ASP* es la tecnología competidora de Microsoft. Las ventajas de *JSP* son dos. Primero, la parte dinámica está escrita en *Java*, no en VBScript o cualquier lenguaje específico de ASP, así que es más poderoso y mejor para desarrollar aplicaciones que requieren componentes reutilizables. Y segundo, *JSP* es portable a cualquier sistema operativo y servidor Web, no estás encerrado en Windows NT/2000 e IIS. Se puede utilizar el mismo argumento cuando comparamos *JSP* con *ColdFusion*: con *JSP* se puede usar *Java* y no estás atado a un servidor en particular.

- Frente a *PHP*: la ventaja de *JSP* es que la parte dinámica está escrita en *Java*, el cual es probable que ya se conozca, ya tiene una extensa API para el trabajo en red, acceso a bases de datos, objetos distribuidos, … frente a lo cual *PHP* requiere el aprendizaje de un nuevo lenguaje entero.

- Frente a los *servlets*: *JSP* no provee ninguna capacidad que no pueda ser, en principio, llevada a cabo con un *servlet*. En efecto, los documentos *JSP* son automáticamente traducidos en *servlets*. Pero es más cómodo escribir (¡y modificar!) HTML normal que tener “millones” de declaraciones println que generen el HTML. Además, separando la presentación del contenido, se puede colocar a diferentes personas en diferentes tareas: los expertos en diseño Web pueden construir el HTML usando sus herramientas habituales y dejar espacios para que los programadores de *servlets* inserten el contenido dinámico.

4.4. Ejemplo de JSP

Este pequeño ejemplo pretende ilustrar el funcionamiento de una página JSP. Como se puede ver en la Figura 9, en el código se mezclan etiquetas HTML para la presentación y código Java para generar la salida. En este caso calcula el factorial de los veinte primeros números.

```
<!-- factoriales.jsp -->
<HTML>
<HEAD>
        <TITLE>Cálculo de factoriales con JSP</TITLE>
</HEAD>

<BODY BGCOLOR="#FFFFFF">

<%! public long factorial (long numero)
 {
       if (numero == 0) return 1;
       else return numero * factorial(numero - 1);
 } %>

<H1>TABLA DE FACTORIALES</H1>
<table border="1">
<tr><th><i>x</i></th><th><i>x</i>!</th></tr>
<% for (long x = 0; x <=20; ++x) { %>
    <tr><td><%= x %></td><td><%= factorial (x) %></td></tr>
<% } %>
</table>

</BODY>
</HTML>
```

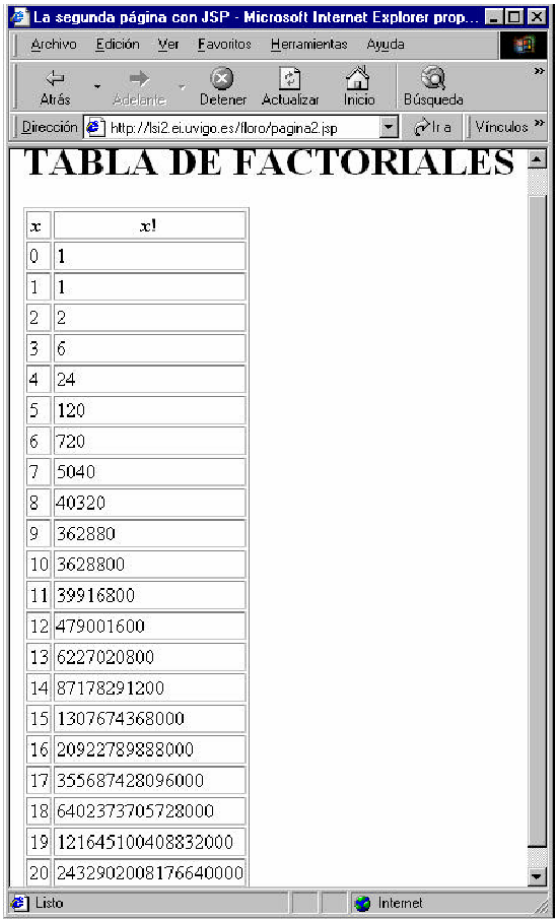
Figura 9. Código de la página JSP

El código Java introducido en la página JSP se ejecutará y generará dinámicamente el código HTML de la Figura 10, ahorrando el trabajo de escribirlo a mano.

```
<!—factoriales.jsp -->
<HTML>
<HEAD>
<TITLE>La segunda página con JSP</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<H1>TABLA DE FACTORIALES</H1>
<table border="1">
<tr><th><i>x</i></th><th><i>x</i>!</th></tr>
<tr><td>0</td> <td>1</td></tr>
<tr><td>1</td> <td>1</td></tr>
<tr><td>2</td> <td>2</td></tr>
<tr><td>3</td> <td>6</td></tr>
<tr><td>4</td> <td>24</td></tr>
<tr><td>5</td> <td>120</td></tr>
<tr><td>6</td> <td>720</td></tr>
<tr><td>7</td> <td>5040</td></tr>
<tr><td>8</td> <td>40320</td></tr>
<tr><td>9</td> <td>362880</td></tr>
<tr><td>10</td> <td>3628800</td></tr>
<tr><td>11</td> <td>39916800</td></tr>
<tr><td>12</td> <td>479001600</td></tr>
<tr><td>13</td> <td>6227020800</td></tr>
<tr><td>14</td> <td>87178291200</td></tr>
<tr><td>15</td> <td>1307674368000</td></tr>
<tr><td>16</td> <td>20922789888000</td></tr>
<tr><td>17</td> <td>355687428096000</td></tr>
<tr><td>18</td> <td>6402373705728000</td></tr>
<tr><td>19</td> <td>121645100408832000</td></tr>
<tr><td>20</td> <td>2432902008176640000</td></tr>
</table>
</BODY>
</HTML>
```

Figura 10. Código HTML generado por la página JSP

El resultado en pantalla será el mostrado en la Figura 11.



x	x!
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040
8	40320
9	362880
10	3628800
11	39916800
12	479001600
13	6227020800
14	87178291200
15	1307674368000
16	20922789888000
17	355687428096000
18	6402373705728000
19	121645100408832000
20	2432902008176640000

Figura 11. Cálculo de factoriales usando JSP

5. Entorno de explotación

Para poder utilizar tanto *servlets* como *JSP* y desarrollar sitios Web dinámicos son necesarios, básicamente, los siguientes elementos:

- JDK 1.2. o posteriores, que proporcionan la API Servlet. Esta API se distribuye como extensión estándar del JDK sin formar parte del núcleo de la plataforma *Java*.
- Un servidor HTTP que de soporte a estas dos tecnologías, es decir, software que implemente las especificaciones Java Servlet y JavaServer Pages.

Estos dos elementos son los realmente necesarios para poder explotar un sitio Web con *servlets* y páginas *JSP.* La JDK nos permite compilar los *servlets* y las *JSP* y el servidor ejecuta el código compilado y muestra el contenido generado.

Centrándonos en un caso práctico si queremos practicar con estas tecnologías lo recomendable es conseguir un servidor gratuito que nos facilite los elementos antes descritos. Existe gran variedad de servidores Web que soportan el API Servlet, como Apache Tomcat, JavaServer Web Development Kit, Allaire JRun o Sun´s Java Web Server.

Una de las opciones más extendidas es utilizar la combinación Apache-Tomcat. El servidor Apache nos proporciona páginas HTML de forma eficiente y rápida, mientras el servidor Tomcat proporciona el contenedor necesario para los *servlets* y las páginas *JSP.* Para utilizar esta combinación también es necesario un "puente" que conecte ambos servidores, de tal forma que cuando se pide una página HTML sea Apache directamente quien la sirva, mientras que si la petición es de un *servlet* o una *JSP* Apache recibirá esa petición y la transmitirá a Tomcat, que será el encargado de servir esa petición.

6. Conclusiones

Las tecnologías *servlet* y *JSP* de *Java*, no se plantean como dos alternativas a poder utilizar separadamente, sino como técnicas complementarias. Es más, las páginas *JSP* cuando se compilan se transforman en *servlets*.

Una de las ventajas importantes de usar cualquiera de estas dos técnicas es que se utiliza código escrito en *Java*, con todos los beneficios que ello conlleva: no hay necesidad de aprender un nuevo lenguaje, portabilidad, reutilización, etc.

El problema de utilizar *servlets* directamente es que, aunque son muy eficientes, son muy tediosos de programar puesto que hay que generar la salida en código HTML con gran cantidad de funciones println. Este problema se resuelve fácilmente utilizando *JSP*, puesto que aprovecha la eficiencia del código *Java*, para generar el contenido dinámico, y la lógica de presentación se realiza con HTML normal.

Sin embargo, cuando en una página *JSP* se necesita introducir mucha funcionalidad, es decir, introducir mucho código *Java* para generar el contenido dinámico de nuestra página, ese mismo hecho lleva a que el código de la página *JSP* no sea demasiado claro.

Por lo tanto, el dilema está en decidir cuándo utilizar *servlets* y cuándo *JSP*. Lo ideal sería usar *JSP* cuando el dinamismo que se pretende no supone introducir mucho código *Java* en las páginas, puesto que esto oscurecería el código. Sin embargo cuando hay mucha funcionalidad y necesitamos mucho código *Java*, lo ideal sería utilizar una página *JSP* que llamase a un *servlet* que contenga la funcionalidad necesaria para que éste realice el trabajo y genere la respuesta, ocupándose el código *JSP* de presentar la información que devuelve el *servlet*.

7. Bibliografía

ECKEL, Bruce. Piensa en Java. 2ª ed. Madrid. Pearson Education, S. A. 2002. 960 p. ISBN 84-205-3192-8

HALL, Marty. Core Servlets and JavaServer Pages. 1ª ed. Prentice Hall PTR. 2000. 608 p. ISBN 0-13-089340-4

Características de Java. http://fciencias.ens.uabc.mx/~mjava/Java/carac.html

Java en castellano. Introducción a Java.

http://programacion.com/java/tutorial/intjava

Java en castellano. Servlets y JSP.

http://programacion.com/java/tutorial/servlets_jsp/3

JavaServer Pages. http://www.verextremadura.com/miguel/jsp/JavaServerPages.pdf

Official Java Page. Java Servlet Technology.

http://java.sun.com/products/servlet/docs.html

Official Java Page. JavaServer Pages Technology.

http://java.sun.com/products/jsp/docs.html

JavaServer Pages: A developer's perspective. By Scott McPherson.

http://developer.java.sun.com/developer/technicalArticles/Programming/jsp

Página personal de Francisco J. García Peñalvo. Defensa de los trabajos voluntarios 2000-2001. http://tejo.usal.es/~fgarcia/docencia/poo/01-02/defensa.html

javaHispano. Tu lenguaje, tu comunidad. http://www.javahispano.com