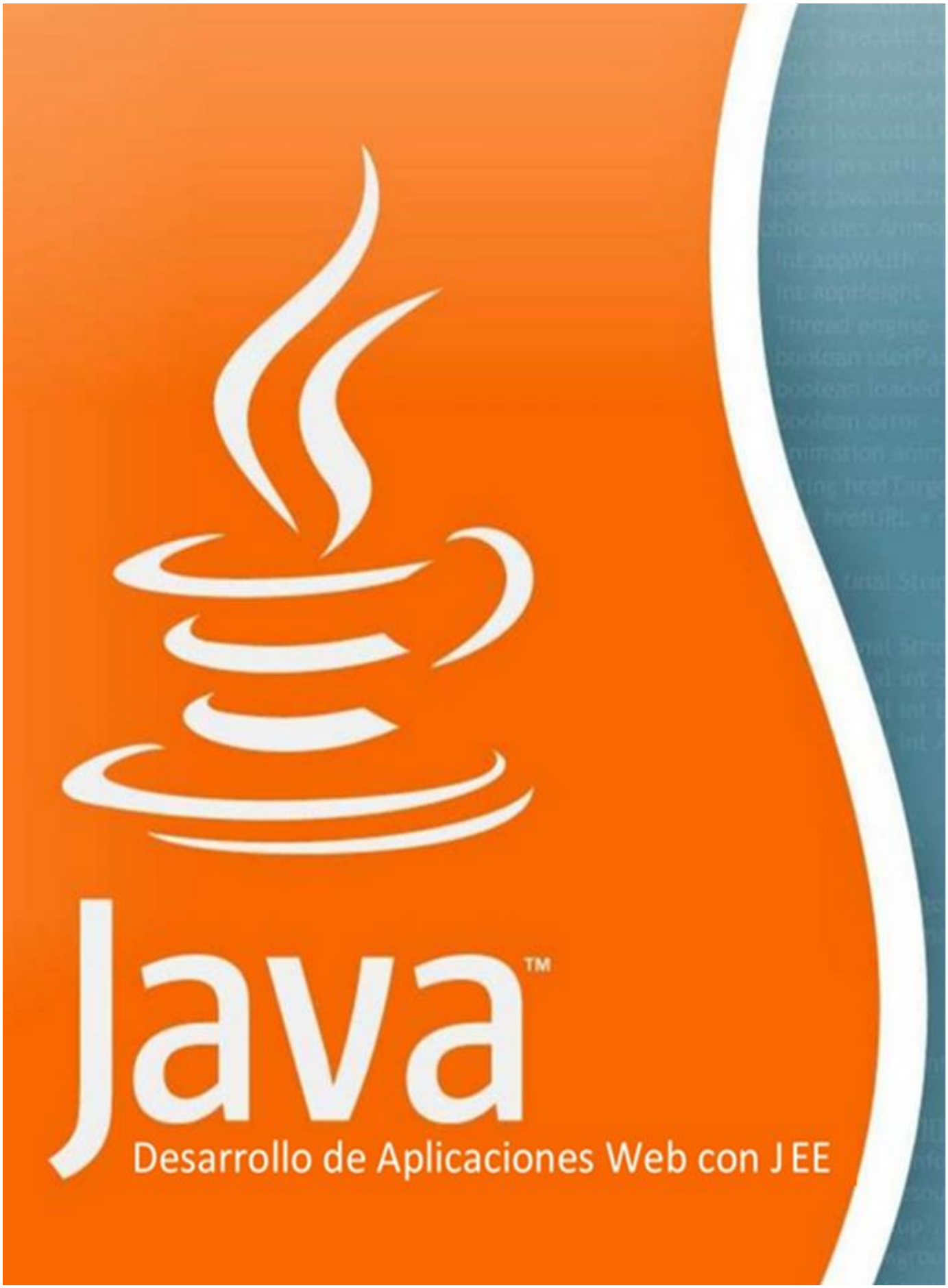

Java™
Desarrollo de Aplicaciones Web con J EE

## PARTE VIII

- **JSP**

## 8. JSP

Un Servlet podría verse como una herramienta que nos permite generar HTML desde código Java. Hasta ahora todos los ejemplos vistos en clase son bastante simples. En una web real, especialmente si queremos que tenga una apariencia razonablemente buena, tendremos que generar cantidades de HTML mucho mayores. Si hacemos esto desde un Servlet, será increíblemente tedioso.

Los Servlets son fundamentalmente programas Java. Y son una buena herramienta para hacer implementaciones relacionadas con programación, como la implementación de la lógica de negocio de una aplicación web, almacenar o recuperar datos de una base de datos, etc. Pero los Servlets son una herramienta pésima para crear la capa de presentación de la aplicación web, es decir, para generar el HTML que finalmente se va a visualizar en navegador web del usuario.

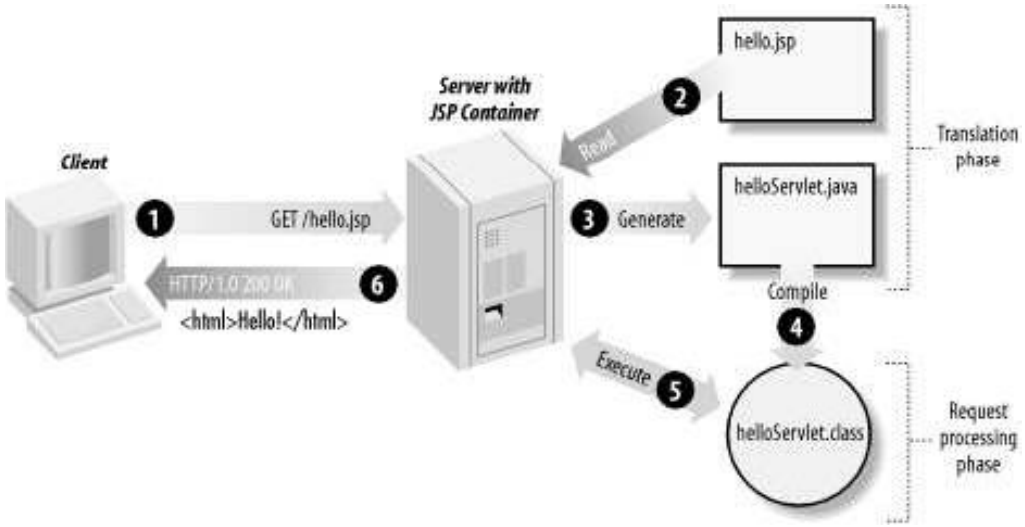
Dentro de Java EE tenemos una herramienta mucho más adecuada para generar HTML que son las **páginas JSP (Java Server Pages),** permite separar la parte dinámica de las páginas Web del HTML estático. Simplemente se escribe HTML y se encierra el código de las partes dinámicas entre etiquetas especiales, la mayoría de las cuales empiezan con **"<%"** y terminan con **"%>".**

Además del HTML normal, hay tres tipos de construcciones JSP que se pueden incrustar en una página JSP:

- Los **elementos de script** permiten especificar código Java que se convertirá en parte del servlet resultante
- Las **directivas** permiten controlar la estructura general del servlet
- Las **acciones** permiten especificar componentes que deberían ser usados, y de otro modo controlar el comportamiento del motor de JSP.
- Para simplificar los elementos de script, se tiene acceso a un número de variables predefinidas.

**A nivel de implementación, las páginas JSP son realmente Servlets**. Cuando un usuario de nuestra aplicación solicite por primera vez una página JSP, el contenedor web va a generar un archivo ".java" que contiene un Servlet que genera todo el HTML que contiene la página JSP, más el código Java que contenía dicha página. A continuación, compila este código Java generando el archivo class correspondiente. Finalmente, despliega el Servlet resultante en el contenedor web, y envía la petición del usuario ha dicho Servlet. La primera vez que una página JSP es alcanzada por una petición del usuario, el servidor va a

tardar un tiempo anormalmente alto en responder. Va a tener que traducir la página JSP a código Java, compilarlo, y desplegar el nuevo Servlet en el contenedor. A partir de la primera petición, la página JSP ya ha sido transformada a un Servlet, y su velocidad de respuesta será similar a la de cualquier otro Servlet.




Las páginas JSP se sitúan dentro de un proyecto Java EE de un modo similar a las páginas HTML estáticas, en desarrollo suelen estar colgando de la raíz de un directorio con nombre "**web**", posiblemente organizadas en una estructura de directorios adecuada para nuestra aplicación.

A menudo no queremos que un usuario pueda llegar directamente una página JSP. Puede que esa página esté preparada, por ejemplo, para recibir los datos de un formulario o que la página suponga que habrá algún dato guardado en la sesión del usuario. Sin embargo, si publicamos la página JSP como cualquier otra página HTML estática, nada le va a impedir al usuario teclear en su navegador la URL correspondiente con la página y acceder a ella directamente. Con las páginas JSP, esto podría dar lugar a un error en la aplicación porque la página ha sido accedida de un modo que no estaba previsto.

**Para resolver este problema se colocan las página JSP dentro del directorio /WEB-INF**. Este directorio nunca es accesible directamente a través de una URL. Sólo va a poder llegar a ellas a través de una redirección realizada por un Servlet o por otra página JSP. Obviamente, esta solución sólo es válida para aquellas páginas JSP a las que no nos interesa que el usuario pueda acceder a ellas directamente. Si

queremos que el usuario tenga acceso directo a la página, tendremos que colocarla fuera de ese directorio.

**8.1 Ejemplo15**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>HOLA MUNDO JSP</title>
</head>
<body>
<h1>Hola mundo JSP</h1>
<p>La fecha actual en el servidor es <%= new java.util.Date() %> </p>
</body>
</html>
```

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<%! String cadena="HOLA MUNDO JSP"; %>
<html>
<head>
<meta charset="ISO-8859-1">
<title>HOLA MUNDO JSP</title>
</head>
<body>
<%=cadena%>
</body>
</html>
```

## 8.2 Sintaxis de las JSP

### 8.2.1 Código Java

En muchos casos, un gran porcentaje de las páginas JSP consistirán en HTML estático, conocido como plantilla de texto. Este HTML es HTML normal, sigue las mismas reglas de sintaxis, y simplemente pasa a través del cliente por el servlet creado para manejar la página.

Se puede incluir código Java dentro de las páginas JSP mediante los elementos de script. Hay cuatro formas:

1. **Expresiones:** que se evalúan y el resultado se muestra en el navegador. Puede contener concatenaciones y acceso a atributos estáticos de clases de java. Su sintaxis es **<%= expresión Java %>**

   <%=cadena%>
   <%="El valor del número Pi es"+ Math.PI%>

   La expresión es evaluada, convertida en un String, e insertada en la página. Esta evaluación se realiza durante la ejecución (cuando se solicita la página) y así se tiene total acceso a la información sobre la solicitud.

2. **Scriptlets:** son fragmentos de código Java. Todo el código HTML que incluyamos dentro de la página JSP, así como las etiquetas de scriptlet, van a ser copiados a un método con nombre **_jspService()** del Servlet que se va a generar a partir de nuestra página JSP. Su sintaxis es **<% código Java %>**

```
<% if (saludar) { %>
<p> Hola</p>
<% }
else { %>
<p> Adios</p>
<% } %>
```

Si la variable saludar vale true, mostraremos en la página el saludo "Hola", en caso contrario, "Adios". El código de los scriptlets va a ser copiado tal cual al método _jspService(), mientras que el HTML se va a mandar a la respuesta del usuario. Por tanto, en el método _jspService() el código que las líneas anteriores van a generar será:

```
if (saludar) {
out.println("<p> Hola</p>");
}
else {
out.println("<p> Adios</p>");
}
```

El código dentro de un scriptlet se insertará exactamente como está escrito, y cualquier HTML estático anterior y posterior al scriptlet, se convierte en sentencias print. Esto significa que **los scriptlets no necesitan completar las sentencias Java**, y los bloques abiertos pueden afectar al HTML estático fuera de los scriptlets. Por ejemplo si en la expresión incluimos un objeto Java, como sucede siempre que se concatena una cadena de caracteres con un objeto Java, se invocará a su método toString():

```
<p> La hora actual es: <%= new java.util.Date() %> </p>
```

Dará lugar a:

```
out.println("<p> La hora actual es: " + (new
java.util.Date()).toString() + "</p>");
```

Si se quiere usar los caracteres “%>” dentro de un scriptlet, se debe usar “\%>”.

3. **Declaraciones:** se pueden incluir declaraciones de campos o de métodos que se encuentran entre los símbolos **<%!...%>**

<%!String cadena= “Hola Mundo!!”;%>
<%!private int contador=0;%>
Número de accesos a la página:
**<%=++contador%>**

4. **Comentarios:**

El siguiente tipo de comentario será incluido dentro del código del Servlet que será generado a partir de esta página JSP: **<!--Comentario-->**

El siguiente tipo de comentario sólo se ve en las páginas JSP. No se incluyen de ningún modo en el Servlet durante la fase de producción, ni se envían al navegador del usuario.

**<%-- Comentario de página JSP-- %>**

### 8.2.1.1 Ejemplo16

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<!--Ejemplo16.jsp-->
<%! int numero = 0;%>
<%! public String fecha() {
      return (new java.util.Date()).toString();
}%>
<html>
<head>
<meta charset="ISO-8859-1">
<title>EJEMPLO 16</title>
</head>
<body>
<h1>TABLA CONTADOR:</h1>
<table border=2>
<%
for (int i = numero; i < numero + 10; i++) {%>
<tr>
<td>Número</td>
<td><%= i %></td>
</tr>
<%}
numero += 10;
%>
</table>
<p>La fecha actual es: <%= fecha() %></p>
</body>
</html>
```

## 8.2.2 Directivas

Como hemos comentado los archivos JSP se transforman en un servlet para su ejecución. Las directivas influyen en la manera en que se realiza esa transformación. Su sintaxis es:

**<%@ directiva AtributoA=”valor1”… AtributoN=”valorN” % >**

Existen dos tipos principales de directivas:

**1.<%@page...%>**: Esta directiva modifica la forma en la que se traduce la página JSP a un Servlet, se utiliza para importar clases, acceder a la sesión, gestionar errores, etc.

- **import**: permite especificar los paquetes que deben ser importados. El atributo import es el único que puede aparecer múltiples veces. Por ejemplo:

  <%@ page import= "java.util.Date" % >

  <%@ page import= "java.util.*, java.text,java.sql.Date" % >

- **contentType:** especifica el tipo de MIME de la salida. El valor por defecto es text/html. Por ejemplo:

  <%@page contentType=”text/`plain”%> tiene el mismo resultado que <%response.setContentType (“text/plain”);%>

- **sesión**: un valor true por defecto, indica que la variable predefinida sesión (del tipo HttpSession) debería unirse a la sesión existente si existe una, y si no existe, se debería crear una nueva sesión. Un valor false, indica que no se usaran sesiones, y los intentos de acceder a la variable sesión resultarán ser errores al momento en que la página JSP sea traducida a servlet. Por ejemplo:

  <%@ page session= "false" % >

- **extends**: esto indica la superclase del servlet que se va a generar, Se debe usar con extrema precaución, ya que el servidor podría utilizar una superclase personalizada.
- **info:** define un string que puede usarse para ser recuperado mediante el método getServletInfo.
- **errorPage**: especifica una página JSP que se debería procesar, si se lanzara cualquier excepción, pero no fuera capturado en la página actual. Por ejemplo:

  <%@page errorPage=”GestionarError.jsp”%>

- **isErrorPage**: indica si la página actual actúa o no como página de error de otra página JSP. El valor por defecto es false.
- **buffer:** esto especifica el tamaño del buffer para el JSPWriter out en Kilobytes. El valor por defecto es específico del servidor, y debería ser de al menos 8 KB.
- **autoflush**: un valor a true indica que el buffer debería descargarse cuando esté lleno. Un valor a false, raramente se utiliza, indica que se debe lanzar una excepción cuando el buffer se sobrecargue.

2. **<%@include ...%>:** nos permite incluir un fichero dentro de otro. Simplemente copia el contenido de un fichero dentro de otro. La idea de estas directiva es incluir este contenido que se repite en múltiples páginas en un fichero independiente, y empleando esta directiva incluir dicho contenido en cada una de las páginas que lo va a usar. De este modo, cuando es necesario modificar ese contenido basta con modificar el fichero independiente y no es necesario modificar todas las páginas donde aparece.

<%@ include file="banner.jsp" %>

#### 8.2.2.1 Ejemplo17

```
<%@page session="true" %>
<%
Integer accesos = (Integer)session.getAttribute("accesos");
if (accesos==null)
accesos=0;
accesos = accesos.intValue()+1;
session.setAttribute("accesos", accesos);
if (request.getParameter("invalidaSesion")!=null)
session.invalidate();
%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Invalidar sesión</title>
</head>
<body>
<form>
<input type="submit" name="invalidaSesion"
value="Invalidar sesión"/>
<input type="submit" value="Recargar página"/>
</form>
<br/>
Contador:
<%= accesos.intValue() %>
</body>
</html>
```

### 8.2.3 Variables predefinidas

Para simplificar el código en expresiones y scriptlets JSP, se tiene las variables predefinidas, llamadas también objetos implícitos. Las variables disponibles son:

- **request:** este es el HttpServletRequest asociado con la petición, y permite usar los parámetros de la petición, el tipo de la petición y las cabeceras entrantes.
- **response:** este es el HttpServletResponse asociado con la respuesta al cliente. Permite entre otras cosas obtener el flujo de salida.
- **out**: es el PrintWriter usado para enviar la salida al cliente, que se obtiene del objeto response. Es una versión con buffer de PrintWriter llamada JspWriter. Se puede ajustar el tamaño del buffer, o incluso desactivarlo, usando el parámetro de la directiva page.
- **sesión**: este es el objeto HttpSession asociado con la petición. Las sesiones se crean automáticamente, por ello esta variable se une incluso si no hubiera una sesión de referencia entrante. La única excepción es usar el atributo sesión de la directiva page para desactivar las sesiones, en cuyo caso en los intentos de referencia a la variable sesión causarán un error en el momento de traducir la página JSP a un servlet.
- **application**: este es el ServletContext obtenido mediante getServletContext().getContext().
- **config**: este es el objeto ServletConfig para esta página.
- **page**: es un sinónimo de this.

## 8.2.4 Acciones

Las acciones JSP usan construcciones de sintaxis XML para controlar el comportamiento del motor de Servlets. Las acciones tienen diferentes usos, por ejemplo incluir la salida resultante de ejecutar un fichero JSP en otro, redirigir una petición de una página JSP a otra. La segunda tendrá acceso a los argumentos de la primera e incluso podrá recibir otros argumentos nuevos.

1. **jsp:include:** esta acción permite insertar archivos en una página que está siendo generada. La sintaxis es:

<jsp:include page=”URL” />

Al contrario de la directiva include, que inserta el archivo en el momento de la conversión de la página JSP a un servlet, esta acción inserta el archivo en el momento en que la página es solicitada. Esto se paga con eficiencia, e imposibilita a la página incluida contener código JSP general pero se obtiene una significante flexibilidad.

2. **jsp:forward:** esta acción permite reenviar la petición a otra página.

<jsp:forward page=”utils/errorReporter.jsp” />

<jsp:forward page=”<%=expresionJava%>” />”

**8**

### .2.4.1 Ejemplo18

**incluido.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:include page="cuerpo.jsp"/>
</body>
</html>
```

**cuerpo.jsp**

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<%for(int i=0;i<5;i++){
out.println("Hola DAM");%>
<br>
<%
}
%>
</body>
</html>
```

### 8.2.4.2 Ejemplo19

### primero.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<jsp:forward page="segundo.jsp">
  <jsp:param name="arg3" value="333"/>
</jsp:forward>
</body>
</html>
```

Se pasan dos argumentos por la URL:

http://localhost:8080/web-app-example/primero.jsp?arg1=12&arg2=78

```jsp
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>

<%
out.println(request.getParameter("arg1"));
out.println(request.getParameter("arg2"));
out.println(request.getParameter("arg3"));
%>

</body>
</html>
```

### 8.2.5 Gestión de errores

La ejecución de páginas JSP implica la ejecución de código Java. Java es un lenguaje orientado a objetos en el que el tratamiento de errores se realiza mediante excepciones. Por lo tanto, una aplicación escrita en JSP puede generar excepciones.

JSP incluye dos directivas para el tratamiento de excepciones: **errorPage y isErrorPage**. Para indicar en una página JSP que página se va a encargar de tratar la excepción debemos utilizar la directiva: <%@page errorPage=" " %>. Y si queremos que nos muestre la excepción generada, para eso debemos utilizar isErrorPage:<%@ page isErrorPage = ''true''%>

#### 8.1.5.1 Ejemplo20

**index.jsp**

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Página principal</title>
</head>
<body>
Número: <%= Integer.parseInt(request.getParameter("numero")) %>
</body>
</html>
```

Si el parámetro es un número no hay problema, pero si se para algo que no pueda transformarse en un entero, mediante una URL como esta por ejemplo:

http://localhost:8080/web-app-example/index.jsp?numero=rtr

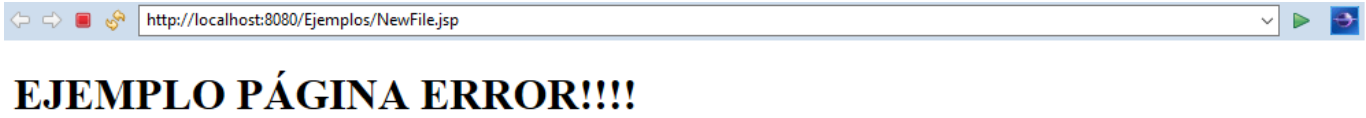
Se creará una excepción, que al no ser tratada, generará una página de respuesta de la forma :


Si modificamos la página anterior para indicar que página se va a encargar de los errores añadiendo al principio la directiva: <%@page errorPage="error.jsp" % >y escribimos dicha página, por ejemplo así:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" errorPage="error.jsp"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Página principal</title>
</head>
<body>
Número: <%= Integer.parseInt(request.getParameter("numero")) %>
</body>
</html>
```

```
 <%@ page isErrorPage = "true" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Página error</title>
</head>
<body>
<h1>EJEMPLO PÁGINA ERROR!!!!</h1>
<% if (exception != null) {
// sacar la traza por pantalla pasandole un flujo de salida de datos
exception.printStackTrace(new java.io.PrintWriter(out));
}%>
</body>
</html>
```

Si se produce un error, la página mostrada será error.jsp y no la página por defecto de Tomcat:



El único problema es que podemos mostrar una nueva web en caso de error, pero no podemos trabajar con la excepción generada. Para ello modificamos la página error.jsp añadiendo el siguiente código:

```
<%@ page isErrorPage = true%>
<% if (exception != null) {
exception.printStackTrace(new java.io.PrintWriter(out));
}%>
```

Nos muestra la pila de la llamada que ha generado la excepción: