JSP

Córdoba - 2008

Tabla de Contenido

Introducción

¿Qué es JSP?

JavaServer Pages (JSP) combinan HTML con fragmentos de Java para producir páginas web dinámicas.
Una página JSP es un archivo de texto simple que consiste en contenido XML o HTML con validaciones en JavaScript y con elementos JSP, escrito en cualquier editor de texto y almacenado con extensión jsp.

A modo de ejemplo, una página JSP que genera HTML

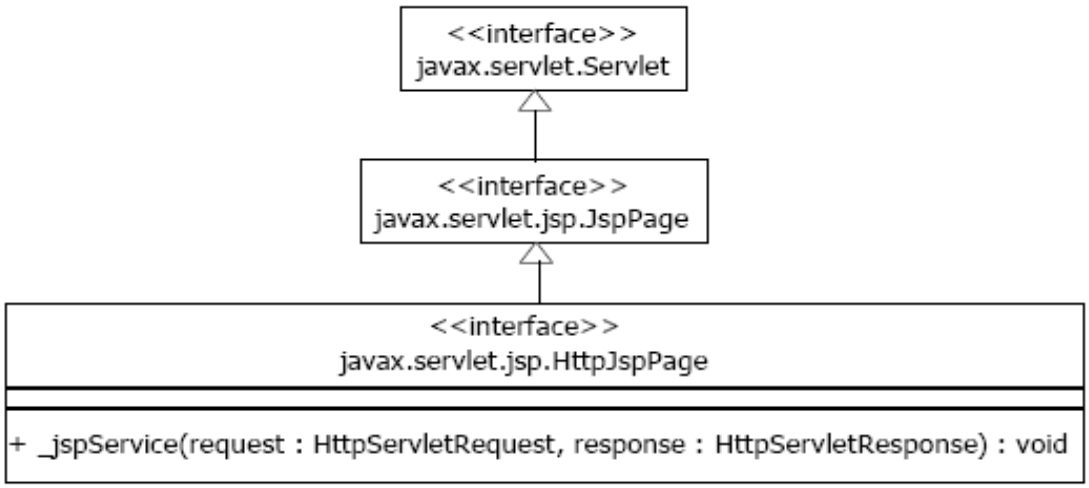
- Tiene el aspecto de una página HTML
- Puede incluir scriptlets (scripts) para generar HTML dinámicamente
- Típicamente los scriptlets se escriben en Java

Algunas especificaciones de su sintaxis: el texto completo lo encontrará en este documento en el artículo original Sun.

- Todas las etiquetas distinguen entre mayúsculas y minúsculas.
- Las comillas simples son equivalente a las comillas dobles.
- No se permiten espacios entre el signo igual y el valor de un atributo
    - texto normal = requerido
    - [] = opcional
    - negrita = defecto
    - { } = opción requerida
    - itálica = definido-usuario
    - … = lista de elementos
    - | = o

En realidad, una página JSP es un tipo especial de servlet (javax.servlet.jsp y javax.servlet.jsp.tagext) orientado a generar el texto de la interfaz gráfica invocables por GET y POST

¿De quién hereda?



Si nos fijamos en estas interfaces podemos encontrar las siguientes clases:

- JSPPage
- HttpJspPage

Ellas definen la interfase para el compilador de páginas JSP.
Encontramos también tres métodos:

- JspInit()
- JspDestroy()
- jspService(HttpServletRequest request, HttpServletResponse response)

Los dos primeros métodos pueden ser definidos por el autor de la página JSP, pero el tercer método es una versión compilada de la página JSP, y su creación es responsabilidad del motor de JSP.

¿Qué ocurre cuando se accede a una página JSP?

Si es la primera vez, el servidor de aplicaciones genera un servlet (que implementa javax.servlet.jsp.HttpJspPage) a partir de la página JSP, lo compila y lo carga en memoria, o sea, cada página es automáticamente compilada a servlet por el motor de JSP, en primer lugar es tomada y a continuación ejecutada.
Si no es la primera vez, le pasa la petición al servlet (ya compilado y creado en memoria)
Si la página se ha modificado desde la última compilación, el servidor se da cuenta, genera el nuevo servlet, lo compila y lo carga de nuevo.

JSP puede comunicarse con las clases de Java, servlets, applets y el servidor web de distintas formas; por esto se puede aplicar una funcionalidad a nuestra web a base de componentes.

Es posible ver el código del servlet generado, este código debe estar en el directorio que se informa en la estructura de directorios del servidor.

Veamos una página JSP bastante simple.

```
<HTML>
   <HEAD>
      <TITLE>Página simple JSP</TITLE>
   </HEAD>
<BODY>
comentarios
<%--INFORMACION GLOBAL PARA LA PAGINA%>
      directivas
<%@ page language=”java” %>
      comentarios
<% // DECLARACION DE VARIABLES %>
      script
<% char c = 0;%>
<%
for (int i=0;i < 26;i++)
{
   for (j = 0;j < 26;j++)
   {
      c = (char) (0x41 + (26 – i +j)%26);
      %>
      comentarios
      <%-- IMPRIME LAS LETRAS DEL ALFABETO EN MAYUSCULAS %>
      <%=c%>
      <%
   }
}
%>
</BODY>
</HTML>
```

Veamos otro ejemplo de JSP y lo cómo se representaría el servlet compilado:

```
<%@ page errorPage="myerror.jsp" %>
<%@ page import="com.foo.bar" %>
<html>
<head>
      <%! int serverInstanceVariable = 1;%>
      ...
      <% int localStackBasedVariable = 1; %>
```

Servlet

```
package jsp_servlet;

import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import com.foo.bar; //se importa por <%@ page import="com.foo.bar" %>
import ...

class _myserlvet implements javax.servlet.Servlet,
javax.servlet.jsp.HttpJspPage {
    //importado por <%! int serverInstanceVariable = 1;%>
    int serverInstanceVariable = 1;
    ...

public void _jspService( javax.servlet.http.HttpServletRequest request
, javax.servlet.http.HttpServletResponse response )
      throws javax.servlet.ServletException, java.io.IOException
    {
        javax.servlet.ServletConfig config = ...;
        Object page = this;
        PageContext pageContext=...; //obtiene el page context del req
        javax.servlet.jsp.JspWriter out = pageContext.getOut();
        HttpSession session = request.getSession( true );
        ...
```

Objetos que componen una página JSP

El código fuente de una página JSP incluye:

Directivas: Dan información global de la página, por ejemplo, importación de estamentos, página que maneja los errores o cuando la página forma parte de una sesión, en el ejemplo anterior informamos del tipo de script de Java.
Declaraciones: Sirven para declarar métodos y variables.
Scripts de JSP: Es el código Java embebido en la página.
Expresiones de JSP: Formatea las expresiones como cadenas para incluirlas en la página de salida.
Acciones: métodos predefinidos para controlar las operaciones del servlet mediante sintaxis XML similar a los taglibs

Estos elementos siguen una sintaxis como XML, así se obtiene un significado con una presentación totalmente separada de la lógica. Con el mecanismo de extensiones de tag se tiene la posibilidad de definir tags con acciones similares y poner la funcionalidad en una librería de tags.

Directivas

Una directiva de JSP es un fragmento que proporciona la información del motor de JSP para la página que la pide.

Su sintaxis general es: `<%@ directiva {atributo ="valor"} %>`

Donde la directiva debe tener un número de atributos. Cada directiva tiene un XML opcional equivalente.

También se pueden asocial varios atributos a la directiva.
Su sintaxis sería:

```
<%@ directive attribute1="value1"
              attribute2="value2"
              ...
              attributeN="valueN"
%>
```

Las posibles directivas en JSP son:

Page: Información para la página.
Include: Incluye archivos completos palabra por palabra.
Taglib: La dirección de la librería de tags que se usará en la página.

Page: La directiva page nos permite definir uno o más de los siguientes atributos sensibles a las mayúsculas

Atributos y posibles valores	Descripción
language=”java”	Comunica al servidor el lenguaje que va a ser utilizado en el archivo. Siendo Java el único posible
extends=”package.class”	Define la clase padre del servlet generado
import=”package.*,package.class”	Especifica los paquetes y clases que se quieran utilizar
session=”true/false”	Mantiene los datos de la sesión para la página. Por defecto es true
isThreadSafe=”true /false”	Especifica si la seguridad de threads está implementada en el fichero JSP. True, por defecto, significa que el motor puede enviar múltiples solicitudes concurrentes a la página
Info=”text”	Información en la página a la que puede accederse con el método Servlet.getServletInfo()
errorPage=”pagina error”	Informa la página que manejará las

	excepciones de errores, especifica un path a un fichero JSP al que este fichero JSP envía excepciones. Si el path empieza con una "/", el path es relativo al directorio raíz de documentos de la aplicación JSP y es resuelto por el servidor Web. Si no, el path es relativo al fichero JSP actual.
isErrorPage=”true / false”	Marca si será la página que manejará el error. Si es true, podemos usar el objeto Exception, que contiene una referencia a la excepción lanzada, en el fichero JSP. Si es false (el valor por defecto), significa que no podemos usar el objeto Exception en el fichero JSP
buffer="none|8kb|sizekb"	Especifica el tamaño del buffer en KB que será usado por el objeto out para manejar la salida enviada desde la página JSP compilada hasta el navegador cliente. El valor por defecto es 8kb
autoFlush="true|false"	Especifica si la salida sería enviada o no cuando el buffer esté lleno. Por defecto, el valor es true, el buffer será descargado. Si especificamos false, se lanzará una excepción cuando el buffer se sobrecargue
contentType="mimeType [; charset=characterSet]" | "text/html;charset=ISO-8859-1"	Este atributo especifica el tipo MIME y la codificación de caracteres que use el fichero JSP cuando se envía la respuesta al cliente. Podemos usar cualquier tipo MIME o conjunto de caracteres que sean válidos para el motor JSP. El tipo MIME por defecto es text/html, y el conjunto de caracteres por defecto es ISO-8859

.

.

.

.

Ejemplo de aplicación:

```
<%@ page
  [ language="java"]
  [ extends="package.class"]
  [ import= "{ package.class|package.*}, ..." ]
  [ session="true|false"]
  [ buffer="none|8kb|sizekb"]
  [ autoFlush="true|false"]
  [ isThreadSafe="true|false"]
  [ info="text"]
  [ errorPage="URLrelativa"]
  [ contentType="mimeType[ ;charset=characterSet]" |
                "text/html; charset=ISO-8859-1"]
  [ isErrorPage="true|false"]
%>
```

Otro ejemplo de aplicación sería:

```
<%@ page language='java'
           contentType='text/html'
           info='Mi primera página en JSP'
           import='java.util.*'
                 errorPage='errorQueTeCagas.jsp' %>
```

La síntaxis XML para definir directivas es:

```
<jsp:directive.TipoDirectiva atributo=valor />
```

Por ejemplo, el equivalente XML de:

```
<%@ page import="java.util.*" %>
```

es:

```
<jsp:directive.page import="java.util.*" />
```

Include: Esta directiva nos permite incluir ficheros en el momento en que la página JSP es traducida a un servlet. Su sintaxis es:

```
<%@ include file="url relativa" %>
```

Normalmente la URL especificada se interpreta como relativa a la página JSP a la que se refiere, pero, también podemos decirle al sistema que interprete la URL reltativa al directorio home del servidor Web empezando la URL con una barra invertida.

Los contenidos del fichero incluido son analizados como texto normal JSP, y así pueden incluir HTML estático, elementos de script, directivas y acciones.

Por ejemplo, Se puede incluir partes de la página que se repetirá en todas las páginas del sitio, así no tendremos que repetir el código.

Por ejemplo:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Servlet Tutorial: JavaServer Pages (JSP) 1.0</TITLE>
<META NAME="author" CONTENT=".....">
<META NAME="keywords" CONTENT="...">
<META NAME="description" CONTENT="...">
<LINK REL=STYLESHEET
      HREF="Site-Styles.css"
      TYPE="text/css">
</HEAD>

<BODY>
//inserta el fichero en el momento en que la página es traducida
<%@ include file="/isipe.html" %>

<!—especificaciones de la página ... -->

</BODY>
</HTML>
```

Cuando los archivos a incluir no cambian seguido, se puede utilizar la directiva en el formato anterior, primero se incluye el contenido del fichero y luego se interpreta. import es una directiva de tiempo de compilación. Pero si va a ser muy dinámica conviene utilizar el formato XML:

```
<jsp:directive.include file="/isipe.html" />
```

Esto incluye el fichero en el momento en que se solicita la página JSP, actúa en tiempo de ejecución.

Taglib: Nos da la posibilidad de utilizar librerías de tags.

Descarga y Uso de TagLibraries

Para poder usar las librerías de tags de JSTL, debemos descargar los ficheros binarios y registrarlos en el fichero web.xml.

http://apache.mesi.com.ar/jakarta/taglibs/standard/binaries/jakarta-taglibs-standard-1.1.2.zip

Descargamos y descomprimimos el fichero marcado.
Normalmente, independiente de la IDE que esté utilizando, deberíamos hacer lo siguiente:

Los ficheros con extensión .tld, los copiamos a nuestro directorio WEB_INF y los jar, al directorio WEB_INF/lib
Ahora modificamos el fichero web.xml para añadir la referencia a las librerías de etiquetas.

```
<jsp-config>
  <taglib>
      <taglib-uri>http://java.sun.com/jstl/fmt</taglib-uri>
      <taglib-location>/WEB-INF/fmt.tld</taglib-location>
  </taglib>

  <taglib>
      <taglib-uri>http://java.sun.com/jstl/fmt-rt</taglib-uri>
      <taglib-location>/WEB-INF/fmt-rt.tld</taglib-location>
  </taglib>

  <taglib>
      <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
      <taglib-location>/WEB-INF/c.tld</taglib-location>
  </taglib>

  <taglib>
      <taglib-uri>http://java.sun.com/jstl/core-rt</taglib-uri>
      <taglib-location>/WEB-INF/c-rt.tld</taglib-location>
  </taglib>

  <taglib>
      <taglib-uri>http://java.sun.com/jstl/sql</taglib-uri>
      <taglib-location>/WEB-INF/sql.tld</taglib-location>
  </taglib>

  <taglib>
      <taglib-uri>http://java.sun.com/jstl/sql-rt</taglib-uri>
      <taglib-location>/WEB-INF/sql-rt.tld</taglib-location>
  </taglib>

  <taglib>
      <taglib-uri>http://java.sun.com/jstl/x</taglib-uri>
      <taglib-location>/WEB-INF/x.tld</taglib-location>
  </taglib>

  <taglib>
      <taglib-uri>http://java.sun.com/jstl/x-rt</taglib-uri>
      <taglib-location>/WEB-INF/x-rt.tld</taglib-location>
  </taglib>
</jsp-config>
```

JSPs con TagLibs

Introducimos la directiva para usar la librería de etiquetas

```
<%@ taglib prefix="jsp2" uri="http://java.sun.com/jstl/core" %>
<html>
<head>
<title>Primer JSP 2.0</title>
</head>
<body>
  <center>
  <jsp2:set var="contador" scope="session" value="1"/>
   El valor del contador es <b> ${contador} </b>
</center>
</body>
</html>
```

Se pueden encontrar una buena referencia en
http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html

Declaraciones

Una declaración de JSP, puede definirse como una definición de variables y métodos a nivel de clase que son usadas en la página.

Un bloque de declaraciones típico sería

```
<%! declaración %>
```

Las declaraciones no producen salida, y su uso mas común es para la declaración de variables que no queremos que cambien desde la compilación de la pagina JSP, como por ejemplo pudiera ser un contador.

```
<%! int contador=0; %>
<% contador++; %>
```

O varias declaraciones: `<%! int a, b; double c; %>`

O la instanciación de una clase: `<%! Circle a = new Circle(2.0); %>`

De esta manera la variable contador aumentaría su valor en una unidad cada vez que la pagina fuese solicitada, el valor del contador se inicializara a 0 cada vez que se arranque el servidor o se modifique la clase.

Un ejemplo de declaración de script sería el siguiente:

```
<HTML>
<HEAD>
<TITLE>Página simple JSP</TITLE>
</HEAD>
<BODY>
<%! String strCadena = "x";
int intContador = 0;
%>
</BODY>
</HTML>
```

El equivalente XML de `<%! Código %>` es: `<jsp:declaration>`

Scripts de JSP

Los Scripts son bloques de código Java residentes entre los tags <% y %>. Este bloques de código estarán dentro del servlets generado incluidos en método _jspService().

En realidad el scriptlet <% .. %> puede manejar declaraciones, expresiones o cualquier otro tipo de fragmento de código válido en el lenguaje script de la página.

Cuando escribimos un scriptlet, lo terminamos con %> antes de cambiar a HTML, texto u otra etiqueta JSP.

Los Scripts pueden acceder a cualquier variable o Beans que haya sido declarado. También hay algunos objetos implícitos disponibles para los Scripts desde entorno del Servlet. Vamos a verlos a continuación

Objetos implícitos	Descripción
request	Petición del cliente. Normalmente es una subclase de la clase HttpServletRequest
response	Para la respuesta al cliente, subclase de HttpServletResponse
pageContext	Los objetos implícitos y atributos de la página necesitan ser accedidos a través de las API, para permitir la compilación de la misma al motor JSP. Como cada servidor tiene implementaciones específicas de los objetos y atributos de Page, lo soluciona utilizando la clase Factory. La clase PageContext es inicializada con los objetos “request” y “response”,

	agregando además algunos atributos de la directiva de la página (errorPage, session, buffer,…), facilitando también el manejo de los demás objetos implícitos
session	Objeto de sesión HTTP asociado a la petición.
application	Objeto que devuelve el servlet cuando se convoca a getServletConfig(). getContext()
out	Representa la salida de texto por pantalla
config	Objeto ServletConfig de la página
page	Forma de reverenciarse la página a sí misma. Alternativa de uso de “this”
exception	Subclase libre de Throwable que se pasa a la clase de manejo de errores

El siguiente fragmento de código muestra como obtener el valor de una parámetro mediante el objeto request, y como pasarlo a una cadena para mostrarlo en pantalla.

```
<%
String strNombre = request.getParameter("nombre");
out.println(strNombre);
%>
```

Es conveniente hacer una referencia al objeto “exception” para poder incorporar su uso en las JSP.

Las excepciones que ocurren durante la ejecución de una aplicación JSP se llaman excepciones en tiempo de ejecución y al igual que en una aplicación Java, una excepción es un objeto ejemplar de java.lang.Throwable o de una de sus subclases.

Las excepciones son condiciones que pueden capturarse y recuperarse de ellas. Estas excepciones podrían ser, por ejemplo, un NullPointerException o un ClassCastException, que nos dicen que se ha pasado un valor nulo o un dato del tipo erróneo a nuestra aplicación mientras se estaba ejecutando.

Las excepciones en tiempo de ejecución se manejan fácilmente en una aplicación JSP, ya que son almacenadas una cada vez en el objeto implícito llamado exception, el cual lo podemos usar en un tipo especial de página JSP llamado página de error, donde mostramos el nombre de la clase exception, su seguimiento de pila, y un mensaje informativo para el usuario.

Las excepciones en tiempo de ejecución son lanzadas por el fichero JSP compilado, el fichero class Java que contiene la versión traducida de nuestra página JSP. Esto significa que nuestra aplicación ha sido compilada y traducida correctamente.

Este tutorial describe cómo crear una sencilla aplicación JSP con varias páginas, un componente JavaBean y una página de error que ofrece mensajes informativos al usuario. En este ejemplo, el Bean sigue la pista sobre la página en la que estaba trabajando el usuario cuando se lanzó la excepción, que nos da a nosotros, el desarrollador, información útil para que podamos mostrar un mensaje informativo. Este es un simple mecanismo de seguimiento de error.

Ejemplo de cómo se podría escribir y lanzar una excepción:

Escribimos nuestro Bean (o bean enterprise, servlet, u otro componente) para que lance ciertas excepciones bajo ciertas condiciones.
Usamos un sencillo mecanismo de seguimiento en nuestro componente para ayudarnos a obtener información sobre lo que estaba haciendo el usuario cuando la excepción fue lanzada. (Si nos movemos en el desarrollo de aplicaciones J2EE, nuestra aplicación podrá grabar el estado, que es la mejor forma de proporcionar información).
El fichero JSP usa una directiva page con errorPage que selecciona el nombre de un fichero JSP que mostrará un mensaje al usuario cuando ocurre una excepción.
Escribir un fichero de página de error, usando una directiva page con isErrorPage="true".
En el fichero de la página de error, usa el objeto exception para obtener información sobre la excepción.
Usamos mensajes informativos, en nuestra página de error o incluida desde otros ficheros, para darle al usuario un mensaje relevantemente informativo sobre lo que el usuario estaba haciendo cuando se lanzó la excepción.

Expresiones de JSP

Las expresiones son una magnífica herramienta para insertar código embebido dentro de la página HTML. Cualquier cosa que este entre los tags <%= y %> será evaluado, convertido a cadena y posteriormente mostrado en pantalla. La conversión desde el tipo inicial a String es manejada automáticamente.

Por ejemplo:

```
<%= Math.sqrt(2) %>
<%= items[i] %>
<%= a + b + c %>
Fecha Actual: <%= new java.util.Date() %>
```

Es importante remarcar que la expresión no termina en punto y coma.

Las expresiones JSP permiten parametrizar las páginas HTML (es parecido a cuando parametriza una consulta SQL pero difieren la forma de los valores).

Una y otra vez, en el código de la página HTML, ser verán bucles o condiciones usando código Java, simplemente empezando y acabando las condiciones o bucles entre los tags <% y %>. Un ejemplo sería:

```
<%
for (int i=0;i<5;i++)
{
%>
<BR>El valor del contador es <%=i%>
<%
}
%>
```

La expresión es ejecutada, se convierte a una cadena de caracteres y insertada en la página.

Otro ejemplo:

```
<%
      String name = null;
      if (request.getParameter("name") == null) {
%>
```

En las expresiones podemos usar las variables predefinidas vistas en el cuadro del punto anterior.

Accciones

Las acciones sirven para controlar el motor de servlets del servidor mediante construcciones de sintaxis XML.
Entre las accriones que podremos realizar se encuentran:
insertar un fichero dinámicamente,
reutilizar componentes JavaBeans,
reenviar al usuario a otra página, o
generar HTML para el plug-in Java.

Recuerde que, como en XML, los nombres de elementos y atributos son sensibles a las mayúsculas.

Las acciones disponibles son las siguientes:

jsp:include: Incluye un fichero en el momento de petición de esta página.

Su sintaxis es:

```
<jsp:include page="relative URL" flush="true" />
```

No es igual que la directiva include, que como pudimos ver insertaba el fichero cuando se generaba el servlet después de realizar la primera petición a la pagina JSP, lo que hace esta acción es insertar el fichero en el momento que la página es solicitada. El problema de esta acción es que se pierde algo de velocidad, la página que vamos a cargar no puede tener código JSP. Ejemplo:

```
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

   <HTML>
   <HEAD>
   <TITLE>Lo mas nuevo</TITLE>
   <LINK REL=STYLESHEET
   HREF="estilo.css"
   TYPE="text/css">
   </HEAD>
<BODY BGCOLOR="#FDF5E6"    TEXT="#000000" LINK="#0000EE"
   VLINK="#551A8B" ALINK="#FF0000">
<CENTER>
   <TABLE BORDER=5 BGCOLOR="#EF8429">
   <TR><TH CLASS="TITLE">
   Lo mas nuevo...</TABLE>
   </CENTER>
   <P>
```

Aqui tenemos las últimas 4 noticias:

```
   <OL>
   <LI> <jsp:include page="noticia1.html" flush="true"/>
   <LI> <jsp:include page="noticia2.html" flush="true"/>
   <LI> <jsp:include page="noticia3.html" flush="true"/>
   <LI> <jsp:include page="noticia4.html" flush="true"/>
   </OL>
   </BODY>
   </HTML>
```

jsp:useBean: Encuentra o ejemplariza un JavaBean para su uso en una página JSP.

La sintaxis es la siguiente:

```
<jsp:useBean id="nombre" class="clase.class" />
```

O bien:

```
<jsp:useBean id="name" class="package.class" />
```

Esto significa que instanciamos un objeto de la clase especificada y lo asociamos a una variable con el nombre especificado por el ID. Podemos

añadir otro atributo más que es scope (ámbito) para que ese Bean se asocie con más de una página JSP.

Una vez que tenemos el bean podemos cambiar sus propiedades usando jsp:setProperty, o usando un scriptlet y llamando al método sobre el objeto que tiene el mismo nombre que el id que le pusimos al bean. La clase especificada por el bean debe estar en el path normal del servidor, no en la parte reservada que obtiene la recarga automática cuando se modifican.

Por ejemplo, en el Java Web Server, él y todas las clases que usa deben ir en el directorio classes o estar en un fichero JAR en el directorio lib, no en el directorio servlets.
Aquí tenemos un ejemplo muy sencillo que carga un bean y selecciona y obtiene un sencillo parámetro String.

```
   <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
   <HTML>
   <HEAD>
   <TITLE>Uso de javaBeans en JSP</TITLE>
   <LINK REL=STYLESHEET
   HREF="estilos.css"
   TYPE="text/css">
   </HEAD>
<BODY>
<CENTER>
   <TABLE BORDER=5>
   <TR><TH CLASS="TITLE">
   Uso de Beans en JSP
   </TABLE>
   </CENTER>
   <P>
<jsp:useBean id="mibean"    class="hall.SimpleBean" />
   <jsp:setProperty name="test"
   property="message"
   value="Hello WWW" />

   <H1>Message: <I>
   <jsp:getProperty name="mibean" property="message" />
   </I></H1>

   </BODY>
   </HTML>
```

SimpleBean.java

Aquí está el código fuente usado para el Bean usado en la página.

```
package hall;
   public class SimpleBean
   {
   private String message = "Mensaje no especificado";
 public String getMessage()
   {
   return(message);
   }
 public void setMessage(String message)
   {
   this.message = message;
   }
}
```

Hay otra forma de instanciar un bean, con formato contenedor:

```
   <jsp:useBean ...>
   Body
   </jsp:useBean>
```

De esta manera indicamos que el body sólo se ejecutará la primera vez que el bean sea instanciado, no cuando este ya instanciado, ya que varias paginas JSP podían compartir Beans, de hay que la instanciacion solo se realice una vez.
Tenemos otras 3 propiedades que podemos usar al instanciar un bean:

scope: Indica el contexto en el que el bean estará disponible: page, request, session y application.

El valor por defecto es page, que quiere decir que el bean solo estará disponible para la pagina JSP actual.
Si es request quiere decir que estará disponible para la petición actual del cliente almacenada en el objeto ServletRequest.
El valor session significa que el bean estará disponible para todas las páginas mientras dure la sesion actual.
El valor application significa que el bean estará disponible para las páginas que compartan el ServletContext.

type: Indica el tipo de variable a la que se referirá el objeto.
Debe corresponder con el nombre de la clase o ser una superclase o un interface que implemente la clase.

beanName: Da el nombre del bean, como lo suministraríamos en el método instantiate de Beans.
Esta permitido suministrar un type y un beanName, y omitir el atributo class.

jsp:setProperty: Sirve para asignar una propiedad a un bean.

Esto se puede hacer en dos contextos:
- Podemos usar jsp:setProperty fuera de un jsp:useBean , de esta forma se ejecuta el metodo sin que el bean haya sido instanciado.

Por ejemplo:

```
<jsp:useBean id="myName" ... />
...
<jsp:setProperty name="myName"
property="someProperty" ... />
```

- La segunda forma es dentro del cuerdo de jsp:useBean de esta forma:

```
<jsp:useBean id="myName" ... >
...
<jsp:setProperty name="myName"
property="someProperty" ... />
</jsp:useBean>
```

En este caso el jsp:setProperty solo se ejecutará en caso de que el bean sea instanciado, no si existía ya.

jsp:setProperty tiene 4 atributos posibles:

name: designa el bean cuya propiedad va a ser usada. Un jsp:useBean debe aparecer antes del elemento jsp:setProperty.

property: Indica la propiedad que vamos a usar. Hay un caso especial, un "*" significa que los parámetros que sean pasados con un nombre igual que una propiedad, serán asignados a dicha propiedad.

value: Especifica el valor para la propiedad. Los string se convierten a tipos numéricos, boolean, byte o char. No se puede usar param y value juntos.

param: Designa el parámetro de la petición que le pasamos a la propiedad del bean. Si no existe el parámetro no pasa nada, el sistema no hace nada y continúa la ejecución.

Ejemplo:

```
<jsp:setProperty name="orderBean"
property="numberOfItems"
param="numItems" />
```

En caso de que no pongamos a la propiedad ni value ni param, entonces lo suyo sería pasar paramentros que correspondan con nombres de propiedad.

Entonces si ponemos un setProperty con el "*" el servidor rellenara las propiedades con los parámetros y asociaría los que tienen el nombre igual.

Ejemplo que usa un bean para crear una tabla de números primos.

Si hay un parámetro llamado numDigits en los datos de la petición, se pasa dentro del bean a la propiedad numDigits. Al igual que en numPrimes.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD    HTML 4.0 Transitional//EN">
<HTML>
   <HEAD>
   <TITLE>Uso de JavaBeans en JSP</TITLE>
   <LINK REL=STYLESHEET
   HREF="My-Style-Sheet.css"
   TYPE="text/css">
   </HEAD>
<BODY>
<CENTER>
   <TABLE BORDER=5>
   <TR><TH CLASS="TITLE">
   Uso de JavaBeans en JSP</TABLE>
   </CENTER>
   <P>
<jsp:useBean id="primeTable"    class="hall.NumberedPrimes" />
   <jsp:setProperty name="primeTable" property="numDigits"    />
   <jsp:setProperty name="primeTable" property="numPrimes"    />
Some <jsp:getProperty name="primeTable"    property="numDigits" />
   digit primes:
   <jsp:getProperty name="primeTable" property="numberedList"    />
</BODY>
</HTML>
```

jsp:getProperty: Este elemento recupera el valor de una propiedad del bean, lo convierte a un string, e inserta el valor en la salida. Los dos atributos requeridos son name, el nombre de un bean referenciado anteriormente mediante jsp:useBean, y property, la propiedad cuyo valor debería ser insertado.

Por ejemplo:

```
<jsp:useBean id="itemBean" ... />
...
<UL>
<LI>Number of items:
<jsp:getProperty name="itemBean" property="numItems" />
<LI>Cost of each:
<jsp:getProperty name="itemBean" property="unitCost" />
</UL>
```

jsp:forward: Esta acción nos permite reenviar la petición a otra página. Tiene un sólo atributo, page, que debería consistir en una URL relativa.

Este podría ser un valor estático, o podría ser calculado en el momento de la petición, como en estos dos ejemplo:

```
<jsp:forward page="/utils/errorReporter.jsp" />
<jsp:forward page="<%= someJavaExpression %>" />
```

jsp:plugin: Esta acción nos permite insertar un elemento OBJECT o EMBED específico del navegador para especificar que el navegador debería ejecutar un applet usando el Plug-in Java.

Elemento JSP	Síntaxis	Interpretación	Notas
Expresión JSP	<%= expression %>	La Expresión es evaluada y situada en la salida.	El equivalente XML es <jsp:expression> expression </jsp:expression>. Las variables predefinidas son request, response, out, session, application, config, y pageContext.
Scriptlet JSP	<% code %>	El código se inserta en el método service.	El equivalente XML es: <jsp:scriptlet> code </jsp:scriptlet>.
Declaración JSP	<%! code %>	El código se inserta en el cuerpo de la clase del servlet, fuera del método service.	El equivalente XML es: <jsp:declaration> code </jsp:declaration>.
Directiva page JSP	<%@ page att="val" %>	Dirige al motor servlet sobre la configuración general.	El equivalente XML es: <jsp:directive.page att="val"\>. Los atributos legales son (con los valores por defecto en negrita): import="package.class" contentType="*MIME-Type*" isThreadSafe="true|false" session="true|false" buffer="*size*kb|none" autoflush="true|false" extends="package.class" info="*message*"

			errorPage="*url*" isErrorPage="true|false" language="java"
Directiva include JSP	<%@ include file="url" %>	Un fichero del sistema local se incluirá cuando la página se traduzca a un Servlet.	El equivalente XML es: <jsp:directive.include file="url"\>. La URL debe ser relativa. Usamos la acción jsp:include para incluir un fichero en el momento de la petición en vez del momento de la tradución.
Comentario JSP	<%-- comment --%>	Comentario ignorado cuando se traduce la página JSP en un servlet.	Si queremos un comentario en el HTML resultante, usamos la síntaxis de comentario normal del HTML <-- comment -->.
Acción jsp:include	<jsp:include page="relative URL" flush="true"/>	Incluye un fichero en el momento en que la página es solicitada.	Aviso: en algunos servidores, el fichero incluido debe ser un fichero HTML o JSP, según determine el servidor (normalmente basado en la extensión del fichero).
Acción jsp:useBean	<jsp:useBean att=val*/> o <jsp:useBean att=val*> ... </jsp:useBean>	Encuentra o construye un Java Bean.	Los posibles atributos son: id="*name*" scope="page|request|session|application" class="package.class" type="package.class" beanName="*package.class*"
Acción jsp:setProperty	<jsp:setProperty att=val*/>	Selecciona las propiedades del bean, bien directamente o o designando el valor que viene desde un parámetro de la petición.	Los atributos legales son: name="*beanName*" property="*propertyName*|*" param="parameterName" value="*val*"

Acción jsp:getProperty	<jsp:getProperty name="propertyName" value="*val*"/>	Recupera y saca las propiedades del Bean.	
Acción jsp:forward	<jsp:forward page="*relative URL*"/>	Reenvía la petición a otra página.	
Acción jsp:plugin	<jsp:plugin attribute="*value*"*> ... </jsp:plugin>	Genera etiquetas OBJECT o EMBED, apropiadas al tipo de navegador, pidiendo que se ejecute un applet usando el Java Plugin.	

Convenciones de Programación Java para Páginas JSP

Se presenta a continuación las especificaciones que realiza Sun para el buen uso y convenciones de la programación en JSP. Este material es original, está en Inglés y lo encontrará en la página oficial de Sun en la URL: http://java.sun.com/developer/technicalArticles/javaserverpages/code_convention/

Article
Code Conventions for the JavaServer Pages Technology Version 1.x Language

Por Kam Hay Fung, Java Technology Consultant y Mark Roth, JavaServer Pages 2.0 Specification Co-Lead, Febrero 2003

Articles Index

As JavaServer Pages (JSP) technology is becoming widely adopted in web-based applications, many JSP programmers and web developers embarking on developing and maintaining these applications face a dilemma like that of many Java programmers: "How do we structure JSP code that is easier to read, write and maintain consistently?"

In this article, we propose a set of standard conventions for writing JSP pages (versions 1.1 and 1.2) that should be followed on a typical software project using web components. The article draws on the Code Conventions for the Java Programming Language as a template to identify various important elements that should be addressed in a coding conventions specification (relevant to JSP technology). In particular, it addresses file names and organization, indentation, comments, directives, declarations, scriptlets, expressions, white space, naming conventions, and programming practices. As this is our first attempt at presenting a set of JSP coding conventions, we're quite interested in any feedback you may have on these recommendations. Please send all feedback to jsp-codeconv-comments@sun.com.

The JavaServer Pages 2.0 Specification, while fully backwards compatible with version 1.2, allows for a script-free programming style (without declarations, scriptlets and expressions) and has a number of new features that are expected to evolve these conventions. Where possible, this article chooses conventions that will leverage the new JSP 2.0 technology features.

Finally, we assume that you are familiar with JSP technology, Java technology and Java code conventions as already adopted by your organization for your projects. If not, we recommend that you start with Java technology here and JSP technology here.

Why Have Code Conventions?

Code conventions are important to programmers and web content developers for a number of reasons:

1. They improve the readability of software artifacts
2. They reduce training management and effort
3. They leverage organizational commitment towards standardization

File Names and Locations

File naming gives tool vendors and web containers a way to determine file types and interpret them accordingly. The following table lists our recommended file suffixes and locations.

File Type File Suffix Recommended Location		
JSP technology `.jsp` <context root>/<subsystem path>/		
JSP fragment `.jsp` <context root>/<subsystem path>/		
`.jspf` <context root>`/WEB-INF/jspf/`<subsystem path>/		
cascading style sheet `.css` <context root>`/css/`		
JavaScript technology `.js` <context root>`/js/`		
HTML page `.html` <context root>/<subsystem path>/		
web resource `.gif, .jpg`, etc. <context root>`/images/`		
tag library descriptor `.tld` <context root>`/WEB-INF/tld/`		