UNIVERSIDAD CARLOS III DE MADRID

Tecnologías informáticas de la Web

TEMA 1 – JSP, Servlet y Filtros

JSP

Jesús Hernando Corrochano

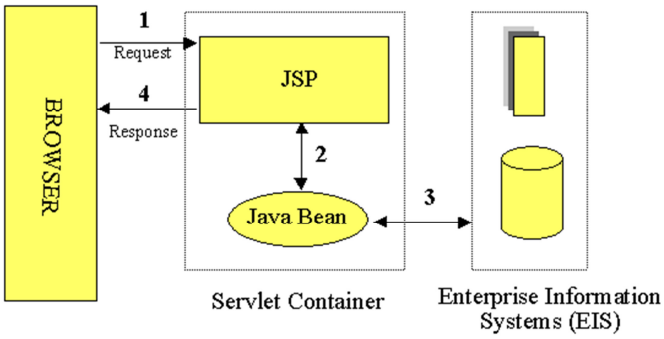
Telmo Zarraonandia Ayo

Curso 2011-12

ÍNDICE

Introducción

Las páginas JSP (Java Server Pages) surgen con la idea de facilitar la creación de contenido dinámico a desarrolladores sin necesidad de conocer a fondo el lenguaje Java. Una página JSP combina código HTML con fragmentos de código Java con el objeto de producir un contenido Web en el que se mezclan tanto componentes estáticos como dinámicos. Además de código HTML y fragmentos de código Java una página JSP puede instanciar clases, hacer llamadas a otras páginas JSP, Servlets e incluir JavaBeans y applets.



A primera vista una página JSP parece una página HTML normal en la que se ha incrustado algunos fragmentos de código Java. Un servidor Web, sin embargo, identificará una página JSP por medio de su extensión ".jsp", de tal forma que cada vez que encuentre una página con una extensión de este tipo sabrá que dicha página requiere un tratamiento especial y comprobará si ha sida solicitada con anterioridad. Si la respuesta es afirmativa la página ya se encuentra en memoria, por lo que sólo es necesario recuperar el código generado. En caso contrario se notifica al motor de JSP y se generará un servlet para la página. Cuando un cliente solicita una página jsp, se ejecuta en el servidor el código JSP de la página, dando como resultado una página HTML que se fusiona con el HTML original, generando una página HTML de respuesta que será enviada al cliente

Las páginas JSP son ideales para situaciones en las que necesitamos mostrar anotaciones con contenido dinámico integrado. Sin embargo, aunque generar HTML es mucho más fácil con JSP que con un Servlet, las páginas JSP son menos adecuadas para manejar la lógica de procesamiento.

Aunque en un JSP es posible replicar la funcionalidad de un Servlet, no es este su objetivo. La idea es que un JSP sea la capa de presentación, separándose así del negocio y el control de la aplicación.

La principal diferencia entre los Servlets y los JSPs es el enfoque de la programación:

- JSP es una página Web con etiquetas especiales y código Java incrustado, mientras que un Servlet es un programa que recibe peticiones y genera a partir de ellas una página Web.
- Pensamos en:
  - Los Servlets como en objetos controladores
  - Las JSP como objetos de vista o que presentan información al usuario.

- No es necesario elegir entre utilizar páginas JSP o Servlets en una aplicación Web. Son tecnologías complementarias y una aplicación Web compleja hará uso de ambas.

Los elementos de un JSP pueden ser expresados de dos maneras:

- Estándar
- XML. Una página en sintaxis XML es un documento XML que puede ser manipulado por herramientas XML y API's para XML.

En un fichero dado solo puede utilizarse una manera (no ambas).

Métodos Service e Init

Una JSP es compilada y convertida en un Servlet antes de enviar la respuesta al cliente, por consiguiente posee un método init y un método service como todo Servlet, pero ¿dónde están?:

lo que escribimos en una página JSP tanto HTML como código Java es luego insertado dentro del método service cuando se compila. Si queremos definir atributos y métodos fuera del service debemos hacer uso de lo que son la declaraciones en JSP, esto lo veremos más adelante.

Secciones / Elementos de un JSP

Una página JSP consiste de las siguientes secciones:

Comentarios

Dos tipos:

- Lado del servidor: son comentarios sólo visibles en el lado del servidor ya que son eliminados durante la compilación del JSP. Deben estar delimitados entre los tags <%-- de inicio y --%> de fin de comentario (se entienden como comentarios JSP). Un comentario JSP es ignorado por el traductor JSP-a-Scriptlet. Cualquier elemento de script, directivas o acciones embebidas son ignorados.

  Ejemplo:

  ```
  <%--
      - Autor(es):
      - Fecha:
      - Descripción:
  --%>
  ```

- Lado Cliente: si es necesario que los comentarios sean visibles en el lado del cliente (por ejemplo para informar sobre derechos de autor) se utilizan comentarios HTML delimitados entre <!-- de inicio y --> de fin de comentario. Un comentario HTML Sse pasa al HTML resultante. Cualquier elemento de script, directivas o acciones embebidas se ejecutan normalmente.

  Ejemplo:

  ```
  <!—Este contenido está protegido por copyright -->.
  ```

Código HTML

En muchos casos, un gran porcentaje de nuestras páginas JSP consistirá en HTML estático, conocido como plantilla de texto.

En casi todos los aspectos, este HTML se parece al HTML normal, sigue la mismas reglas de sintaxis, y simplemente "pasa a través" del cliente por el Servlet creado para manejar la página. No sólo el aspecto del HTML es normal, puede ser creado con cualquier herramienta que usemos para generar páginas Web.

Elementos de script JSP (JSP actions)

Los elementos de script nos permiten insertar código Java dentro del Servlet que se generará desde la página JSP actual. Hay tres formas:

- **Expresiones** de la forma <%= expresión %> que son evaluadas e insertadas en la salida.

- **Scriptlets** de la forma <% código %> que se insertan dentro del método service del Servlet.
- **Declaraciones** de la forma <%! código %> que se insertan en el cuerpo de la clase del Servlet, fuera de cualquier método existente.

Expresiones

Las expresiones son utilizadas para insertar valores Java (o del lenguaje de script definido) en la salida. Es de la forma:

<%= expresión java %>

La sentencia Java es evaluada en el periodo de procesamiento de la solicitud, convertida a String e insertada en la página.

Esta evaluación es realizada en tiempo de ejecución, con lo que se tiene acceso a la información de la petición.

```
<html>
<body>
…
        Hora actual <%= new java.util.Date() %>
…
</body>
</html>
```

La sintaxis XML alternativa para las expresiones es:

```
<jsp:expression>
        expresion java
</jsp:expression>
```

Asi:

```
<html>
<body>
…
        Hora actual <jsp:expression> new java.util.Date() </jsp:expression>
…
</body>
</html>
```

Scriptlets

Contienen fragmentos de código válidos en el lenguaje de script definido en la página (ver atributo language en directivas). Son ejecutados durante el periodo de procesamiento de solicitud.

Los Scriptlets, como cualquier otro bloque o método de código Java, pueden modificar objetos dentro de ellos como resultados de invocaciones de métodos.

Múltiples Scriptlets son combinados en la clase de Servlet generada en el orden en que aparecen en la JSP. Todo el código que aparece entre etiquetas <% y %> en la JSP es situado en el método service() del Servlet y en el orden en que aparecen. Es, por lo tanto, procesado para cada solicitud que recibe el Servlet.

Su sintaxis es:

<% fragmento de código %>

Ejemplo:

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

La sintaxis alternativa XML para los Scriptlets es:

```
<jsp:scriptlet>
      fragmento de código
</jsp:scriptlet>
```

Ejemplo:

```
<jsp:scriptlet>
      int contador = 0;
      contador++;
</jsp:scriptlet>
```

Declaraciones

Una declaración JSP permite definir métodos y variables que podrán ser utilizadas en el cuerpo del Servlet generado fuera del método service.

Las declaraciones son inicializadas cuando la página JSP es inicializada y tienen alcance instanciado en el Servlet generado, por lo que cualquier elemento definido en una declaración está disponible en JSP para otras declaraciones, expresiones y código.

Su sintaxis es :

<%! Código java %>

Ejemplo:

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

La variable contador que hemos definido será global a todas las peticiones que se hagan al JSP, es decir, actuará como variable compartida.

Las declaraciones deben terminar con punto y coma (;)

Ejemplo:

```
<%!
  private int contador = 0;
  private String cadena = new String();
  String getCadena () {
    return cadena;
  }
%>
```

La sintaxis XML alternativa para las declaraciones son:

```
<jsp:declaration>
        código java
</jsp:declaration>
```

Ejemplo:

```
<jsp:declaration>
    private int contador = 0;
    private String cadena = new String();
    String getCadena () {
      return cadena;
    }
<jsp:/declaration>
```

Directivas JSP

Las directivas JSP definen atributos de la página en el momento de compilación.

Una directiva JSP afecta a la estructura general de la clase Servlet. Normalmente tienen la siguiente forma.

```
<%@ page att="val" %>
```

No hay un límite en el número de directivas que puede tener una página JSP.

Ejemplo (dos formas distintas de definir las mismas directivas, podemos combinar múltiples selecciones de atributos para una sola directiva, de esta forma:):

Código 1:

```
<%@ page session="false" %>
<%@ page import="java.util.*" %>
<%@ page errorPage="/common/errorPage.jsp" %>
```

Código 2:

```
<%@ page session="false"
        import="java.util.*"
        errorPage="/common/errorPage.jsp"
%>
```

Si se quiere importar múltiples paquetes se separa con coma (,).

Ejemplo:

```
<%@ page session="false"
        import="java.util.*,java.text.*,
                com.mycorp.myapp.taglib.*,
                com.mycorp.myapp.sql.*"
%>
```

Directiva page

La directiva `page` es utilizada para definir y manipular una serie de atributos importantes que afectan al conjunto de la página JSP.

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

Una página puede contener cualquier número de directrices `page`, en cualquier orden, en cualquier lugar de la página JSP. Todas son asimiladas durante la traducción y aplicadas en conjunto a la página. Sin embargo, sólo puede haber un ejemplar de cualquier par valor-atributo definido por las directrices de la página en una determinada JSP. Una excepción a esta regla es el atributo `import`, puede haber múltiples importaciones. Por convención, las directrices `page` aparecen al principio de una JSP.

La sintaxis general de la directriz de página es la siguiente:

<%@ page ATRIBUTOS %>

La directiva `page` puede contar con los siguientes atributos:

- **`session="true | false"`** : Si este atributo vale `true` hace referencia a la sesión actual o se crea una nueva sesión, con lo que podremos guardar/recuperar información de la sesión actual del usuario. Si vale false estará deshabilitado el objeto Sesion. El valor por defecto es `true`.
- **`import= "{package.class | package.* }`** : Lista separada por comas (,) de paquetes Java que el JSP debe importar.
- **`isThreadSafe="true | false"`** : Indica si el contenedor puede enviar múltiples peticiones concurrentes a la página JSP. Debe escribirse código en la página JSP para sincronizar los múltiples hilos de los clientes. El valor por defecto es `true`.
- **`info="text"`** : Texto que se incorpora en la página JSP compilada. Puede ser recuperado posteriormente invocando al método Servlet.getServletInfo()
- **`errorPage="relativeURL"`** : Ruta a una página JSP a la que se envían las excepciones. Si la ruta comienza con / es relativa al directorio raíz de la aplicación, sino es relativa al JSP actual.
- **`isErrorPage="true | false"`** : Indica si la página JSP muestra una página de error. Si vale true se puede utilizar el objeto Exception, si vale `false` (valor por defecto) no se puede utilizar el objeto Exception en el JSP.
- **`language="java"`** : Define el lenguaje de scripting utilizado en los Scriptlets, declaraciones y expresiones en la página (ver código HTML y JSP más abajo), por lo general se especifica Java. Este atributo existe en el caso de que futuros contenedores apoyen múltiples lenguajes.
- **`extends ="package.class"`** : Nombre completo de la superclase Java (JSP) de la que heredará este JSP al compilarse. Este atributo debe ser evitado normalmente y sólo debe ser utilizado con extrema precaución, porque las máquinas JSP proporcionan habitualmente superclases especializadas con mucha funcionalidad que deben ser ampliadas por las clases de Servlet generadas. El uso del atributo extendí restringe algunas de las decisiones que un contenedor JSP puede tomar.
- **`contentType="mimeType [ ;charset=characterSet ]" | "text/html;charset=ISO-8859-1"`** : Define el `mimeType` y la codificación de caracteres que utiliza el JSP para la respuesta que envía al cliente. El valor por defecto del `mimeType` es `text/html` y el valor por defecto del charset es `ISO-8859-1`
- **`autoFlush="true | false"`** : Si vale `true` fuerza el volcado del contenido del buffer escribiéndolo en el stream de salida definido. Si vale `false` se lanzará una excepción cuando el buffer esté lleno. El valor por defecto es `true`
- **`buffer ="none | 8kb | sizekb"`** : Tamaño del buffer en kilobytes utilizado por el objeto out para manejar la salida enviada por el JSP compilado al navegador Web del cliente.

Los atributos no conocidos tienen como resultado errores fatales de compilación.

Ejemplo:

```
<%@ page language="Java" import="java.rmi.*, java.util.*"
        session="true" buffer="12kb" autoFlush="true"
        info="mi jsp de directivas" errorPage="error.jsp"
        isErrorPage="false" isThreadSafe="true" %>
<html>
  <head>
    <title>Página de ejemplo de directivas page</title>
  </head>
  <body>
    <h1>Página de ejemplo de directivas page</h1>
  </body>
</html>
```

Directiva include

Esta directiva permite insertar el contenido de un fichero en el momento en que un JSP es transformado en un Servlet. La sintaxis es:

<%@ include file="URL relativa" %>

La URL especificada es normalmente interpretada como relativa a la ubicación del JSP que la referencia. Sin embargo es posible hacer que sea relativa al directorio raíz de la aplicación Web comenzando la ruta con un slash (/). El contenido del fichero a incluir es parseado como un JSP normal, con lo que puede contener HTML estático, elementos de scripting, directivas y acciones.

Por ejemplo, muchos sitios Web incluyen una pequeña barra de navegación en cada página. Para evitar complicaciones y problemas con los frames HTML se puede hacer uso de esta directiva para incluir un fichero que contenga el código HTML de dicha barra de navegación.

El contenido a incluir puede ser una JSP, puede incluir fragmentos HTML, o incluso otro recurso de texto.

Ejemplo:

```
      <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0
Transitional//EN">
      <HTML>
      <HEAD>
      <TITLE>Ejemplo de directiva Include</TITLE>
      <META NAME="keywords" CONTENT="...">
      <META NAME="description" CONTENT="...">
      <LINK REL=STYLESHEET
            HREF="Site-Styles.css"
            TYPE="text/css">
      </HEAD>
```

```
    <BODY>
    <%@ include file="/otra.html" %>
    <!—Partes específicas de la página ... -->
    </BODY>
</HTML>
```

Como la inclusión del fichero con la barra de navegación se hace en tiempo de compilación si el contenido de la barra se modifica debe recompilarse el JSP para que los cambios sean visibles.

Directiva import

Esta directiva permite incluir el contenido de un fichero (por ejemplo otro JSP) en tiempo de compilación, antes de que el JSP sea ejecutado. La sintaxis es:

```
<%@ import file='cabecera.jsp' >
```

Primero se incluye el contenido del fichero y luego se interpreta.

`import` es una directiva de tiempo de compilación.

Directiva de Tag Library

Para poder declarar que una JSP va a utilizar tags definidos en una tag library (librería de etiquetas) es necesario incluir la directiva taglib en la página antes de que cualquier otro tag definido por el usuario sea utilizado:

```
<%@ taglib uri="/tlt" prefix="tlt" %>
```

Los atributos disponibles son:

- **`uri`** : se refiere a la la URI que identifica unívocamente al tag library. Esta URI puede ser relativa o absoluta. Si es relativa debe estar asociada a una ruta absoluta en el elemento taglib del fichero descriptor de despliegue de la aplicación Web, el fichero de configuración asociado con las aplicaciones desarrolladas de acuerdo con la especificación Java Servlet y Java Server Pages.
- **`prefix`** : define el prefijo que distingue los tags pertenecientes a una tag library de los pertenecientes a otra.

Variables predefinidas

Para simplificar el código en Expresiones y Scriplets JSP, tenemos ocho variables definidas automáticamente, algunas veces llamadas objetos implícitos. Las variables disponibles son:

`request, response, out, session, application,`

`config, pageContext,` y `page`.

- **Request.** Este es el `HttpServletRequest` asociado con la petición, y nos permite mirar los parámetros de la petición (mediante `getParameter`), el tipo de petición (GET, POST, HEAD, etc.), y las cabeceras HTTP entrantes (cookies, Referer, etc.). Estrictamente hablando, se permite que la petición sea una subclase de `ServletRequest` distinta de `HttpServletRequest`, si el protocolo de la petición es distinto del HTTP. Esto casi nunca se lleva a la práctica.
- **Response.** Este es el HttpServletResponse asociado con la respuesta al cliente. Observa que, como el stream de salida (ver out más abajo) tiene un buffer, es legal seleccionar los códigos de estado y cabeceras de respuesta, aunque no está permitido en los Servlets normales una vez que la salida ha sido enviada al cliente.
- **Out**. Este es el PrintWriter usado para enviar la salida al cliente. Sin embargo, para poder hacer útil el objeto response (ver la sección anterior), esta es una versión con buffer de PrintWriter llamada JspWriter. Observa que podemos ajustar el tamaño del buffer, o incluso desactivar el buffer, usando el atributo buffer de la directiva *page*. También observa que *out* se usa casi exclusivamente en Scriptlets ya que las expresiones JSP obtienen un lugar en el stream de salida, y por eso raramente se refieren explícitamente a **out**.
- **Session**. Este es el objeto HttpSession asociado con la petición. Recuerda que las sesiones se crean automáticamente, por esto esta variable se une incluso si no hubiera una sesión de referencia entrante. La única excepción es usar el atributo session de la directiva page para desactivar las sesiones, en cuyo caso los intentos de referenciar la variable session causarán un error en el momento de traducir la página JSP a un Servlet.
- **Application**. Este es el ServletContext obtenido mediante getServletConfig().getContext().
- **Config**. Este es el objeto ServletConfig para esta página.
- **pageContext**. JSP presenta una nueva clase llamada PageContext para encapsular características de uso específicas del servidor como JspWriters de alto rendimiento. La idea es que, si tenemos acceso a ellas a través de esta clase en vez de directamente, nuestro código seguirá funcionando en motores Servlet/JSP "normales".
- **Page**. Esto es sólo un sinónimo de this, y no es muy útil en Java. Fue creado como prevención pensando en el día que el los lenguajes de script puedan incluir otros lenguajes distintos de Java.

JavaBean

Los JavaBeans son clases escritas en el lenguaje Java y que representan componentes de software reusables.

Son utilizados para encapsular varios objetos en uno solo (el Bean), para que puedan ser utilizados como un único objeto en lugar de varios objetos individuales.

Un JavaBean debe cumplir varios requisitos para que sea considerado como tal:

- Debe tener un constructor público por defecto. Esto permite la fácil inicialización y edición desde frameworks.
- Las propiedades deben ser accesibles utilizando métodos get y set y otros métodos, siguiendo una convención de nomenclatura. Esto permite la fácil inspección automática y actualización del estado de los beans dentro de los frameworks, muchos de los cuales incluyen editores para varios tipos de propiedades.
- La clase de debe ser serializable. Esto permite a las aplicaciones y los frameworks almacenar y restaurar el estado de un Bean de una manera independiente de la máquina virtual y de la plataforma.

Ejemplo de un JavaBean:

```
/**
 * Class <code>PersonaBean</code>.
 */
public class PersonaBean implements java.io.Serializable {

    private String nombre;

    private boolean fallecido;

    /** Constructor sin argumentos (no recibe parámetros). */
    public PersonaBean() {
    }

    /**
     * Propiedad <code>nombre</code>
     */
    public String getNombre() {
        return this.nombre;
    }

    /**
     * Setter para la propiedad <code>nombre</code>.
     * @param nombre
     */
    public void setNombre(final String nombre) {
       this.nombre = nombre;
    }

   /**
    * Getter para la propiedad "fallecido"
    * Sintáxis diferente para un campo boolean (is en lugar de
    * get)
    */
```

```
    public boolean isFallecido() {
        return this.fallecido;
    }

    /**
     * Setter para la propiedad <code>fallecido</code>.
     * @param fallecido
     */
    public void setFallecido(final boolean fallecido) {
        this.fallecido = fallecido;
    }
}
```

Acciones

Las acciones estándar JSP usan construcciones de sintaxis XML para controlar el comportamiento del motor de Servlets. Podemos insertar un fichero dinámicamente, reutilizar componentes JavaBeans, reenviar al usuario a otra página, o generar HTML para el plug-in Java. Las acciones estándar sirven para proporcionar a los autores de las páginas la funcionalidad básica para explotar tareas comunes. Los tipos de acción estándar son los siguientes:

<jsp:useBean>

Esta acción es utilizada para instanciar un JavaBean o para localizar una instancia bean existente y asignarla a un nombre de variable (o ID).

La sintaxis más simple para especificar que se debería usar un Bean es:

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

Esto normalmente significa "instancia un objeto de la clase especificada por `class`, y únelo a una variable con el nombre especificado por `id`". Sin embargo, también podemos especificar un atributo `scope` que hace que ese Bean se asocie con más de una página. En este caso, es útil obtener referencias a los beans existentes, y la acción `jsp:useBean` especifica que se instanciará un nuevo objeto si no existe uno con el mismo nombre y ámbito.

Hemos de decir sobre los beans que cuando decimos "*este bean tiene una propiedad del tipo X llamada* `direccion`", realmente queremos decir "esta clase tiene un método `getDireccion` que devuelve algo del tipo X, y otro método llamado `setDireccion` que toma un X como un argumento".

Una vez que tenemos un bean, podemos modificar sus propiedades mediante la acción **`jsp:setProperty`** que requerirá que suministremos un valor explícito para la propiedad, que proporcionaremos mediante el atributo `param`. Otra posibilidad es modificar sus propiedades usando un Scriptlet y llamando a un método explícitamente sobre el objeto con el nombre de la variable especificada anteriormente mediante el atributo `id`.

Por el contrario mediante la acción **`jsp:getProperty`** podremos leer las propiedades del bean en una expresión, lo cual podremos conseguir igualmente desde un Scriptlet JSP llamando al método `getXxx`

Ejemplo:.

BeanTest.jsp

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

<HTML>

<HEAD>

<TITLE>Reusing JavaBeans in JSP</TITLE>

</HEAD>

<BODY>

<CENTER>

<table BORDER=5>

  <TR><TH CLASS="TITLE">

      Reutilizando JavaBeans en JSP</table>

</CENTER>

<P>

<jsp:useBean id="prueba" class="tidw.TIDWBean" />

<jsp:setProperty name="prueba"

                 property="mensaje"

                 value=" Hola TIDW !!" />

<H1>Message: <I>

<jsp:getProperty name="prueba" property="mensaje" />

</I></H1>

</BODY>

</HTML>
```

TIDWBean.java

Código fuente usado para el Bean usado en la página BeanTest.jsp.

```
package tidw;

public class TIDWBean {

  private String mensaje = "Mensaje no especificado";

  public String getMensaje() {

    return(mensaje);

  }

  public void setMensaje(String mensaje) {

    this.mensaje = mensaje;

  }

}
```

Los procesos subyacentes que tienen lugar cuando se utiliza la acción useBean son los siguientes:

- El contenedor intenta localizar un objeto que tiene el `id` especificado dentro del alcance especificado.
- Si el objeto es encontrado y se ha especificado un tipo en la etiqueta, el contenedor intenta designar el objeto encontrado al tipo especificado. Si la asignación falla, se lanza una excepción `ClassCastException`.
- Si el objeto no se encuentra en el alcance especificado y la clase puede ser instanciada, entonces se instancia y una referencia del objeto es asociada con el `id` otorgado, en el alcance especificado.

  Si esto falla, se lanza una excepción `InstantiationException`.
- Si el objeto no se encuentra en el alcance especificado y se especifica un `beanName`, entonces se invoca el método `instantiate()` de `java.bean.Beans`, con `beanName` como argumento.

  Si este método tiene éxito, el nuevo objeto referencia es asociado el id otorgado, en el alcance especificado.
- Si una nueva instancia bean ha sido instanciada y el elemento `<jsp:useBean>` tiene un cuerpo no-vacío, el cuerpo es procesado; durante este procesamiento, la nueva variable es inicializada y está disponible.

  Los Scriplets o la acción estándar `<jsp:setProperty>` pueden ser utilizados para inicializar la instancia bean, si es necesario.

Atributos de la acción useBean

Además de `id` y `class`, hay otros tres atributos que podemos usar: `scope`, `type`, y `beanName`.

- **`Id`:** Da un nombre a la variable que referenciará el bean.

  Se usará un objeto bean anterior en lugar de instanciar uno nuevo si se puede encontrar uno con el mismo `id` y `scope`.

- **`class`:** es el nombre de la clase que se instancia y que define al Bean. Debe ser el nombre de una clase concreta y debe poseer un constructor público sin argumentos. Es sensitivo a las mayúsculas tanto para los nombres de los paquetes como para la clase. Si se especifican los atributos `beanName` y `type` el atributo `class` no es necesario, en caso contrario debe especificarse.

- **`beanName`:** debe seguir la especificación JavaBean. Este atributo es opcional y se suele especificar en tiempo de ejecución.

- **`type`:** Especifica el tipo de la variable a la que se referirá el objeto. Este debe corresponder con el nombre de la clase o ser una superclase o un interface que implemente la clase. Recuerda que el nombre de la variable se designa mediante el atributo `id`.

- **`scope`:** Indica el contexto en el que el bean debería estar disponible. Hay cuatro posibles valores: `page`, `request`, `session`, y `application`.

  - El valor por defecto, page, indica que el bean estará sólo disponible para la página actual (almacenado en el PageContext de la página actual).
  - Un valor de `request` indica que el bean sólo está disponible para la petición actual del cliente (almacenado en el objeto `ServletRequest`).
  - Un valor de `session` indica que el objeto está disponible para todas las páginas durante el tiempo de vida de la `HttpSession` actual.
  - Finalmente, un valor de `application` indica que está disponible para todas las páginas que compartan el mismo `ServletContext`.

  La razón de la importancia del ámbito es que una entrada `jsp:useBean` sólo resultará en la instanciación de un nuevo objeto si no había objetos anteriores con el mismo `id` y `scope`. De otra forma, se usarán los objetos existentes, y cualquier elemento `jsp:setParameter` u otras entradas entre las etiquetas de inicio `jsp:useBean` y la etiqueta de final, serán ignoradas.

<jsp:param>

La acción `<jsp:param>` es utilizada para proporcionar otras etiquetas de cierre con información adicional en forma de pares de valor nombre.

Es utilizado en conjunción con las acciones:

`<jsp:include>`,`<jsp:forward>`,`<jsp:plugin>`,

y su uso está descrito en las siguientes secciones relevantes. La sintaxis es:

```
<jsp:param name="paramname" value="paramvalue" />
```

Los atributos disponibles son:

- **name** : la clave asociada al atributo. (Los atributos con pares de valor clave.)
- **value** : el valor del atributo.

<jsp:include>

Esta acción nos permite insertar ficheros en una página que está siendo generada. La sintaxis se parece a esto:

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

Al contrario que la directiva `include`, que inserta el fichero en el momento de la conversión de la página JSP a un Servlet (include estático), esta acción *inserta el fichero en el momento en que la página es solicitada* (include dinámico).

En el siguiente ejemplo tenemos una página JSP que inserta cuatro puntos diferentes dentro de una página Web "What's New?". Cada vez que cambian las líneas de cabeceras, los autores sólo tienen que actualizar los cuatro ficheros, pero pueden dejar como estaba la página JSP principal.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>What's New</TITLE>
<LINK REL=STYLESHEET
      HREF="My-Style-Sheet.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">
      What's New at JspNews.com</table>
</CENTER>
<P>

Here is a summary of our four most recent news stories:
<OL>
  <LI><jsp:include page="news/Item1.html" flush="true"/>
  <LI><jsp:include page="news/Item2.html" flush="true"/>
  <LI><jsp:include page="news/Item3.html" flush="true"/>
  <LI><jsp:include page="news/Item4.html" flush="true"/>
```

```
</OL>
</BODY>
</HTML>
```

Atributos de la directiva `include`:

- **`page`** : el recurso que debe ser incluido. El formato URL es el mismo que el descrito anteriormente para la directriz `include`.
- **`flush`** : este atributo es opcional, con el valor por defecto `false`. Si el valor es `true`, el búfer del flujo de salida es vaciado antes de que se realice la inclusión.

<jsp:setProperty>

Usamos `jsp:setProperty` para establecer valores de propiedades de los beans que se han referenciado anteriormente.

Podemos hacer esto en dos contextos:

- Primero, podemos usar antes `jsp:setProperty`, pero fuera de un elemento `jsp:useBean`, de esta forma:

```
<jsp:useBean id="miBean" ... />

...

<jsp:setProperty name="miBean" property="algunaPropiedad"
... />
```

  En este caso, el `jsp:setProperty` se ejecuta sin importar si se ha instanciado un nuevo bean o se ha encontrado uno ya existente.

- Un segundo contexto en el que `jsp:setProperty` puede aparecer dentro del cuerpo de un elemento `jsp:useBean`, de esta forma:

```
<jsp:useBean id="miBean" ... >

  ...

  <jsp:setProperty name="miBean" property="algunaPropiedad"
... />

</jsp:useBean>
```

  Aquí, el `jsp:setProperty` sólo se ejecuta si se ha instanciado un nuevo objeto, no si se encontró uno ya existente.

Los cuatro atributos posibles de `jsp:setProperty` son:

Atributo	Uso
name	Este atributo requerido designa el bean cuya propiedad va a ser seleccionada. El elemento `jsp:useBean` debe aparecer antes del elemento `jsp:setProperty`.
property	Este atributo requerido indica la propiedad que queremos seleccionar. Sin embargo, hay un caso especial: un valor de "*" significa que todos los parámetros de la petición cuyos nombres correspondan con nombres de propiedades del Bean serán pasados a los métodos de selección apropiados.
value	Este atributo opcional especifica el valor para la propiedad. Los valores `String` son convertidos automáticamente a números, `boolean`, `Boolean`, `byte`, `Byte`, `char`, y `Character` mediante el método estándar `valueOf` en la fuente o la clase envolvente. Por ejemplo, un valor de "`true`" para una propiedad `boolean` o `Boolean` será convertido mediante `Boolean.valueOf`, y un valor de "42" para una propiedad `int` o `Integer` será convertido con `Integer.valueOf`. No podemos usar `value` y `param` juntos, pero si está permitido no usar ninguna.
param	Este parámetro opcional designa el parámetro de la petición del que se debería derivar la propiedad. Si la petición actual no tiene dicho parámetro, no se hace nada: el sistema no pasa `null` al método seleccionador de la propiedad.

El siguiente ejemplo 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`.

JspPrimes.jsp

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

<HTML>

<HEAD>

<TITLE>Reusing JavaBeans in JSP</TITLE>

<LINK REL=STYLESHEET

      HREF="My-Style-Sheet.css"

      TYPE="text/css">
```

```
</HEAD>

<BODY>

<CENTER>

<table BORDER=5>

  <TR><TH CLASS="TITLE">

      Reusing JavaBeans in JSP</table>

</CENTER>

<P>

<jsp:useBean id="primetable" class="tidw.NumberedPrimes" />

<jsp:setProperty name="primetable" property="numDigits" />

<jsp:setProperty name="primetable" property="numPrimes" />

Algunos <jsp:getProperty name="primetable" property="numDigits" />

       Números primos:

<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. Aquí tenemos un 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, a un Servlet o a un recurso estático.

La sintaxis es la siguiente:

```
<jsp:forward page="URL" />

Ó

<jsp:forward page="URL">

  <jsp:param name=”paramname” value=”paramvalue” />

</jsp:forward>
```

El recurso al que la solicitud está siendo reenviada debe estar en la misma aplicación Web que las JSP que está lanzando la solicitud.

La ejecución actual JSP se detiene cuando encuentra una etiqueta `<jsp:forward>`.

El búfer es limpiado (ésta es una redirección de lado servidor y el búfer de respuesta es limpiado durante el procesamiento) y la solicitud es modificada para asimilar cualquier parámetro adicional especificado.

Estos parámetros son asimilados del mismo modo descrito para la acción <jsp:include>.

Si el flujo de salida no estuviera buferado y se hubiera escrito en ella alguna información de salida, una acción <jsp:forward> lanzaría una excepción `java.lang.IllegalStateException`.

El comportamiento de esta acción es exactamente igual que el del método `forward()` de `javax.Servlet.RequestDispatcher`.

<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.

Tratamiento de excepciones

El tratamiento de excepciones en JSP se realiza mediante una mezcla de los atributos:

- errorPage
- isErrorPage

El atributo errorPage especifica una página JSP para que procese cualquier excepción arrojada pero no capturada en la página en curso. La excepción arrojada estará automáticamente disponible por la página del error mediante la variable *exception.*

En cada una de las JSps que generemos hemos de indicar si la destinamos a página de tratamiento de error o por el contrario (mejor opción) delegamos esta responsabilidad en otra página. Esto lo indicamos mediante el atributo isErrorPage.

Ejemplo:

```
<%@ page isErrorPage = "false"%> (no es necesario por defecto el
false)
<%@ page errorPage = "miPaginaDeError.jsp"%>
```

Indica que nuestra página no es una página de error y delega tal responsabilidad en la página `miPaginaDeError.jsp`.

En la página `miPaginaDeError.jsp` tendremos:

```
<%@ page isErrorPage = "true"%>
```

Para indicar que es una página de error y con ello tendrá acceso al objeto intrínseco exception para poder tratar la correspondiente excepción:

```
<%@ page isErrorPage = "true"%>
……
<% if (exception != null) { %>
      Tratamiento de la excepción
if%>
```

Existe una manera más profesional de hacer esto mismo: en vez de ir escribiendo las páginas de error en cada una de las jsp lo hacemos utilizando el fichero web.xml.

```
<error-page>
      <exception-type>java.lang.MyException</exception-type>
      <location>/error.jsp</location>
</error-page>
```

Esto mismo es aplicable a los códigos de error:

```
<error-page>
      <error-code>500</error-code>
      <location>/error500.jsp</location>
</error-page>
```