Tema 3
JSPs

J. Gutiérrez

Departament d'Informàtica
Universitat de València

DAW-TS (ISAW).
Curso 14-15

Índice

JSP: Java Server Pages

Fichero con extensión `jsp` que contiene código estático HTML y código para generar la parte dinámica.

El contenedor se encarga de transformar la página JSP a un Servlet la primera vez que se utiliza.

El servlet generado de manera automática tiene el método `_jspService` con sentencias `println` que generan la respuesta.

```
public void _jspService(HttpServletRequest request,
                        HttpServletResponse  response)
                        throws java.io.IOException,
                        ServletException {...}
```

JSP

Para controlar y generar la parte dinámica JSP ofrece construcciones de tres tipos:

Elementos de script

Código que será incluido en el Servlet generado: `<%! declaraciones %>`
`<%= expresion %>`
`<% codigo %>`

Directivas

Afectan a la estructura del servlet generado:
`<%@ directiva atributo=valor atributo=valor ... %>`

Acciones

Sirven para especificar y usar otros componentes usando una sintaxis XML:

```
<jsp:useBean ... \>
<jsp:setProperty ... \>
<jsp:getProperty ... \>
<jsp:include ... \>
<jsp:forward ... \>
<jsp:plugin ... \>
```

Declaraciones: `<%! codigo Java%>`

Sirven para declarar atributos de la clase Servlet generada:

```
<%! private int contador;

private java.util.Date fecha = new Date();%>
```

Expresiones: `<% = expresion Java%>`

Expresiones que son evaluadas y cuyo resultado se escribe en la salida que produce el servlet.

```
<% = new java.util.Date()%>
```

Scriplets: `<% codigo Java%>`

Sentencias Java completas que se insertan en el método `_jspService`

```
<% out.println("Página accedida: " + contador + " veces desde " + fecha );%>
```

Directivas: `<%@directiva atributo1=''valor1'' ... atributoN=''valorN''%>`

Las directivas afectan a la estructura del servlet generado.
Ejemplos de directivas:

- `page` que permite importar clases, indicar el tipo de contenido, establecer el tamaño de buffer, etc...,
- `include` que permite incluir contenido de otras páginas
- `taglib` que permite usar etiquetas que se transformarán en condicionales, bucles, etc.

http://www.ecst.csuchico.edu/~amk/aiweb/tutorials/servlet/hall/Servlet-Tutorial-JSP.html#Section5

Algunos atributos que se pueden especificar con la directiva page son:

- `import ''paquete1.clase1'',...,''paqueteN.claseN''` clases a importar
- `contentType=''MIME-Type''` o `contentType=''MIME-Type; charset=Codificacion''` contenido y codificación de caracteres
- `isThreadSafe=''(true)|false''` si se especifica `false` se indica que el servlet generado debe extender a `SingleThreadModel` (es decir, una instancia por petición).
- `session="(true)|false"` si se establece `false` se especifica que no se accede a la sesión (y si se intenta acceder se producirá un error).

- `extends=''paquete.clase''` clase a la que debe extender el servlet generado.
- `errorPage=''URL''` página que se debe cargar si se lanza una excepción y no se trata en la página
- `isErrorPage=''true|(false)''` indica si se trata de una página de tratamiento de excepciones

JSP — Ejemplo

```
<?xml version="1.0" encoding="UTF-8" ?>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<%!private java.util.Date fecha = new java.util.Date();
private int contador = 0; %>

<html>
<head>
<title>Ejemplo de JSP</title>
</head>

<body>

<p> Hora a la que se ha instanciado el servlet:
<% = fecha %>
</p>

<p> Numero de veces que se ha accedido al JSP desde que se ha instanciado:
<% = contador %>
<% contador = contador + 1; %>
</p>

<p> Hora a la que se ha generado la respuesta:
<% = new java.util.Date() %>
</p>

</body>
</html>
```

Un *JavaBean* es una clase que cumple las siguientes propiedades:

- tiene un constructor sin argumentos
- sus atributos son privados
- tiene métodos públicos para consultar y modificad los atributos (setters y getters).

La siguiente clase es un ejemplo de *JavaBean*:

```
package some.package;

public class Libro{
   private String autor;

   public Libro(){}

   // Setter
   public setAutor(String aut){
      autor = aut;
   }

   // Getter
   public String getAutor(){
      return autor;
   }
}
```

Acciones estándar: elementos con sintaxis XML

Ejemplos:

<jsp:useBean>: Crea o usa una instancia de un JavaBean

<jsp:setProperty>: Establece un valor a un atributo de un JavaBean

<jsp:getProperty>: Obtiene el valor de un atributo de un JavaBean

<jsp:include>: Incluye el resultado de otro componente web o página estática

<jsp:forward>: Reenvía la petición a otra página.

<jsp:plugin>: Permiten insertar un elemento OBJECT o EMBED para ejecutar un applet usando el plugin de Java.

http://java.sun.com/products/jsp/syntax/2.0/syntaxref20.html

El anterior *JavaBean* se podría usar en una página JSP del siguiente modo:

```
<jsp:useBean id="libro" class="some.package.Libro" scope="..."/>
```

Donde `scope` puede ser uno de los siguientes valores: `application`, `session`, `request`, `page` .

El atributo `id` actúa como una referencia: nos permite acceder posteriormente al JavaBean:

```
<jsp:setProperty name="libro" property="autor" value="..."/>
<jsp:getProperty name="libro" property="autor"/>
```

El valor puede ser un literal o una expresión Java.

Mapeo de parámetros de la petición a un JavaBean

JSP ofrece la posibilidad de insertar los datos enviados en una petición GET o POST directamente en un JavaBean.

Para ello los nombres de los parámetros que se especifican en el formulario deben coincidir con los nombres de los atributos en el JavaBean:

```
public class Libro{
   private String autor;
   private String editorial;
   private int year;

   public Libro(){}

   // Setters y getters
}
```

Mapeo de parámetros de la petición a un JavaBean

Datos en la petición `GET`:

```
GET /addLibro.jsp?autor=Liskova&editorial=Perason&year=2008 HTTP/1.1
...
```

Datos en la petición `POST`:

```
POST /addLibro.jsp  HTTP/1.1
...

autor=Liskova&editorial=Perason&year=2008
```

Mapeo de parámetros de la petición a un JavaBean I

Para que el JSP recoja los datos del formulario y los inserte en un JavaBean usamos la acción estándar:

```
<jsp:setProperty name="bean" property="*"/>
```

Ejemplo (asumimos que una instancia del tipo `Libros` se asocia a la sesión y permite añadir objetos del tipo `Libro`):

```
<?xml version="1.0" encoding="UTF-8" ?>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<jsp:useBean id="libros" class="Libros" scope="session"

<jsp:useBean id="libro" class="user.UserData" scope="page"/>
<jsp:setProperty name="libro" property="*"/>

<HTML>
<BODY>
<% libros.add(libro); %>
```

Mapeo de parámetros de la petición a un JavaBean II

```
El libro </br>

Autor: <jsp:getProperty name="libro" property="author"> <br />
Editorial: <jsp:getProperty name="libro" property="editorial"> <br />
Year: <jsp:getProperty name="libro" property="year"> <br /> </br>

Ha sido añadido correctamente.

</BODY>
</HTML>
```

Hay una serie de objetos predefinidos que podemos utilizar en el código JSP:

Nombre	Tipo
config	ServletConfig
application	ServletContext
session	HttpSession
page	Object
pageContext	PageContext
request	HttpServletRequest
response	HttpServletResponse
out	JspWriter
exception	Throwable

Como se observa en los ejemplos anteriores, todavía aparece código Java (aunque menos que con los servlets).

Para simplificar más la sintaxis de las páginas JSP, en la versión 2.0 se introduce el *expression language* (http://download.oracle.com/javaee/6/tutorial/doc/gjddd.html) que permite la inserción de contenido dinámico sin escribir código Java, ya que:

- Simplifica la salida
- Simplifica el acceso a los beans

Acceso a un *JavaBean*:

`${variable}`

Busca en los objetos `PageContext`, `HttpServletRequest`, `HttpSession` y `ServletContext` en ese orden (de más específico a más general) si existe un atributo llamado `variable` escribe en la respuesta el resultado de `variable.toString()`.

Si ese objeto es un *JavaBean* lo que querremos habitualmente es mostrar sus propiedades (o atributos):

`${variable.atributo}` *o* `${variable["atributo"]}`

Envía el mensaje `getAtributo()` al *JavaBean* escribe en la respuesta el resultado.

Acceso a arrays o a objetos que implementan a la interfaz `List`:

`${variable[indice]}`

Devuelve el elemento que está en la posición `indice` del array o de la lista

Acceso a objetos que implementan a la interfaz `Map`

`${variable.clave}` ó `${variable["clave"]}`

Busca en el objeto JavaBean la propiedad o si se trata de un mapa (`Map`) la clave especificada

Dentro de una expresión ${} se pueden usar una serie de objetos implícitos (predefinidos):

Nombre	Descripción
pageScope	Mapa que contiene los atributos establecidos en *page*
requestScope	Mapa que contiene los atributos establecidos en *request*
sessionScope	Mapa que contiene los atributos establecidos en *session*
applicationScope	Mapa que contiene los atributos establecidos en *application*
param	Mapa que contiene parámetros de la petición `Map<String,String>`
paramValues	Mapa que contiene parámetros de la petición `Map<String,String[]>`
header	Mapa que contiene campos de cabecera de la petición `Map<String,String>`
headerValues	Mapa que contiene campos de cabecera de la petición: `Map<String,String[]>`
cookie	Mapa con los cookies
initParam	Mapa con los parámetros iniciales de contexto (definidos en web.xml)
pageContext	Referencia al objeto PageContext que puede usarse como un bean

Operaciones aritméticas en EL

+,-

Si son números se realiza la operación, si son cadenas se convierten antes a números (si no son se lanzará una excepción), si son `BigInteger` o `BigDecimal` se llama a los métodos `add` o `substract`.

*, /, %

Operaciones multiplicación, división y módulo. Si los argumentos son cadenas realiza la conversión automática a número (si alguna cadena no representa un número entonces se lanzará una excepción del tipo `RuntimeException`).

Operaciones relacionales en EL

== y eq

Si son números se comparan con ==, si son `BigInteger` o `BigDecimal` se comparan usando `compareTo`, en otro caso se comparan usando `equals`.

!= y ne

Si son números se comparan con !=, si son `BigInteger` o `BigDecimal` se comparan usando `compareTo`, en otro caso se comparan usando `equals`.

<, >, <=, >=

Si los argumentos son números se comparan aritméticamente y si son cadenas se comparan lexicográficamente. También están las versiones `&lt`, `&gt`, `&le`, `&ge`.

Operaciones lógicas en EL

&&, and, ||, or, !, not

Operadores lógicos realizados sobre los argumentos que son convertidos a `Boolean`.

empty

Devuelve `true` si el argumento es null, si es una cadena vacía, si es un array vacío, si es un `Map` vacío o una colección vacía. En otro caso devuelve `false`.

Condicional simple en EL

EL soporta el operador rudimentario de comparación:

```
comprobacion ? valor_si_cierto : valor_si_falso
```

Ejemplo:

```
${ param.edad < 18 ? "Welcome" : "You are too old to join our club"}
```

A continuación se muestran dos situaciones que todavía requieren código Java:

- Supongamos que una de las variables que podemos consultar es una colección (alguna de las clases que implemente a `Collection<E>` . Si queremos recorrer y mostrar los elementos que contiene necesitamos usar un bucle.
- Supongamos que en función del valor de una variable queremos ejecutar un código u otro. Para ello se debe usar una estructura condicional.

Para evitarlo se pueden utilizar etiquetas predefinidas que serán transformadas a código Java cuando el contenedor transforme el JSP a un servlet.

Ejemplo de etiqueta que se expande en el código generado a un condicional:

```
<c:if test="${empty datos}">
   No hay datos que mostrar.
</c:if>
```

Ejemplo de un condicional:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<link type="text/css" rel="stylesheet" href="<c:url value='/estilo.css'/>"/>
</head>

<body>

<c:choose>
    <c:when test="${empty header['User-Agent']}">
       Contenido que se mostrara si se accede desde una aplicacion
    </c:when>
    <c:when test="${fn:containsIgnoreCase(header['User-Agent'],'linux')}">
       Contenido que se mostrara si se accede desde linux
    </c:when>
    <c:when test="${fn:containsIgnoreCase(header['User-Agent'],'windows')}">
          Contenido que se mostrara si se accede desde windows
    </c:when>
</c:choose>

</body>
</html>
```

También está `<c:if>` para realizar una comprobación única.

Ejemplo de un bucle:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>
<head>
<link type="text/css" rel="stylesheet" href="<c:url value='/estilo.css'/>"/>
</head>

<body>

<c:forEach var="libro" items="${libros}">
   <div id="${libro.id}">
      <p>${libro.titulo}</p>
      <p>${libro.autores}</p>
   </div>
</c:forEach>

</body>
</html>
```

En el ejemplo anterior se supone que:

- `libros` es una instancia de una subclase de `Collection` (ejemplos de subclases de `Collection`: `Vector`, `ArrayList`, `LinkedList`, ...)
- Los objetos que contiene son del tipo `Libro` el JavaBean que he puesto de ejemplo en la página 11.
- `libros` es visible para el JSP (es un atributo asociado a la aplicación, a la sesión o a la petición).

Índice

MVC: Servlet + JSPs (I)

El patrón *Modelo-Vista-Controlador* es usado en interfaces gráficas de usuario para tener separadas las responsabilidades en objetos diferentes.

MVC en aplicaciones Web con servlet Y JSPs:

- Servlet = Controlador
- JSPs = Vistas

Aproximación:

1. Las peticiones desde el cliente van al servlet,
2. El servlet (controlador), en función de los parámetros, realiza una tarea (puede ser la consulta a una base de datos,...), obtiene unos datos y delega la generación de la respuesta al JSP correspondiente pasándole los datos (por ejemplo añadiendo atributos en `request`).
3. Finalmente, el JSP (vista) genera lo que se va a enviar al cliente (que puede ser una página completa o no...).

MVC: Servlet + JSPs (II)

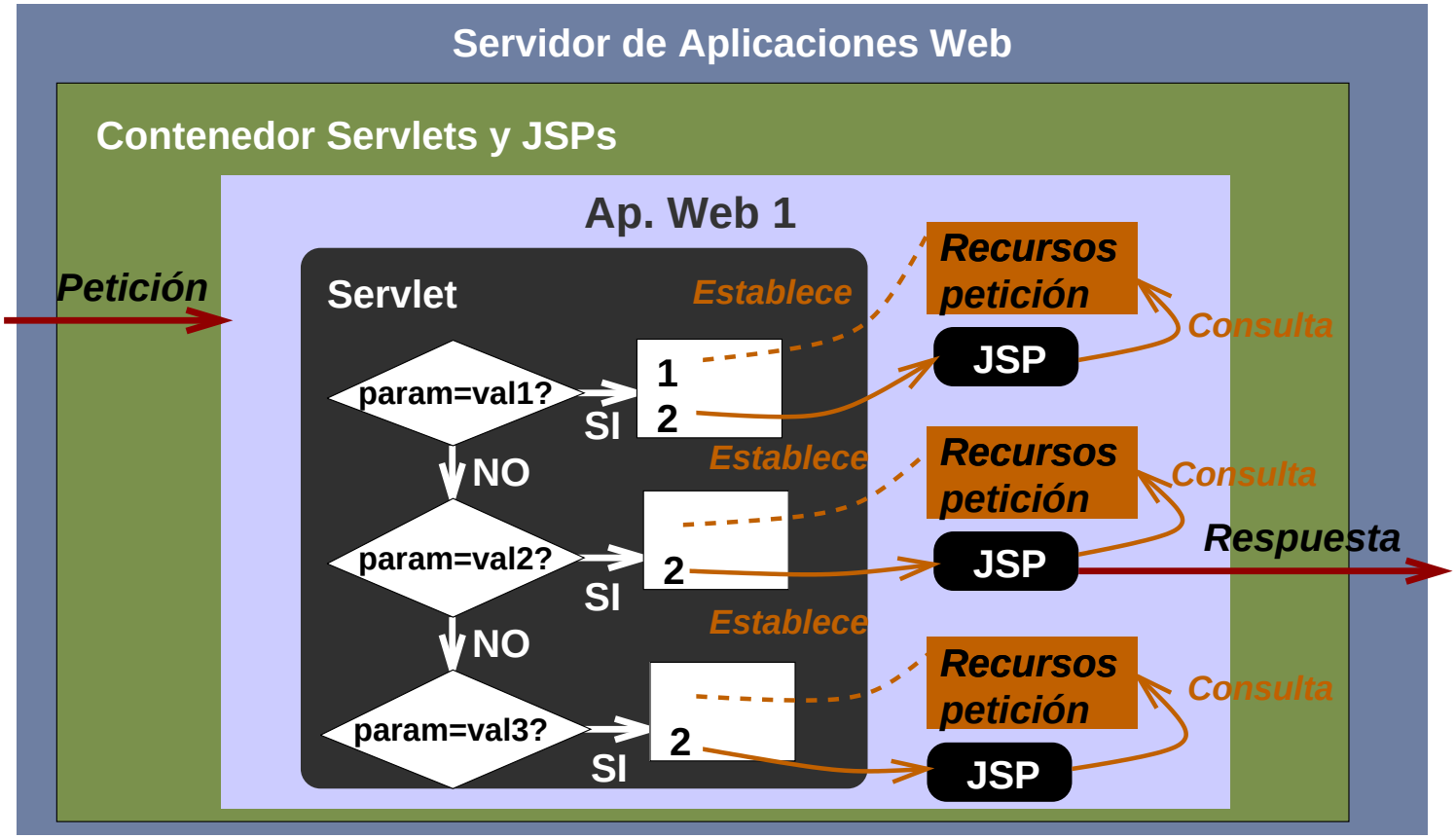
En este ejemplo desde el servlet se pasa un atributo al JSP a través del objeto `HttpServletRequest` (usando request scope, es decir que se coloca un atributo accesible durante la petición):

```
//imports
public class SimpleServlet extends HttpServlet{
   static final long serialVersionUID = 1L;

   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
      request.setAttribute("atributo", "Hola mundo");
      request.getRequestDispatcher("/saludo.jsp").forward(request,response);
   }
   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
         ServletException, IOException {
   }
}
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<html>
<body>
<h1> Hola ${atributo} </h1>
</body>
</html>
```

MVC: Servlet + JSPs (III)



Posibles peticiones

http://.../.../...?param=val1
http://.../.../...?param=val2
http://.../.../...?param=val3