Java EE – JavaServer Pages (JSP)

Aplicaciones Web / Sistemas Web

Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense Madrid

Java EE Servlets y JSPs

- **Servlet**
  - Muy útiles para leer cabeceras de mensajes, datos de formularios, gestión de sesiones, procesar información, etc.
  - Pero tediosos para generar todo el código HTML
    - El mantenimiento del código HTML es difícil
- **JSP (JavaServer Pages)**
  - Fichero con código (X)HTML que incluye scripts codificados en Java
    - Permite usar (X)HTML para definir gran parte de la página
    - E introducir código Java en las partes dinámicas de la página
    - Mediante etiquetas especializadas (*Custom Tags*) que amplían la sintaxis de HTML
  - Se compila y se convierte en un servlet (solo la primera vez que se invoca)
    - Se ejecuta como un servlet
- Con JSP es más fácil que se distribuya la tarea de diseño de la página web y la programación de la aplicación web

JSP y servlets

- Los servlets tienen que generar todo el código HTML

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println ("<!DOCTYPE html>\n" +          "<html>\n" +
      "<head><title>Formulario de saludo</title></head>\n" +
      "<body>\n" +
      "<h1>Hola "+request.getParameter("cliente")+"</h1>\n" +
      "</body></html>");
```

- Las JavaServer Pages (JSP) permiten escribir código HTML e insertar código Java para las partes dinámicas

```
<!DOCTYPE html>
<html>
<head><title>Saludo JSP</title></head>
<body>
<h1>Hola
<% if (request.getParameter("cliente")!=null)
 out.println(request.getParameter("cliente")); %>
</h1>
</body>
</html>
```


Servlet generado

- Se encontrará en el directorio *work* del servidor tomcat

```
 public final class Saludo_jsp extends org.apache.jasper.runtime.HttpJspBase
//...
 public void _jspService(final javax.servlet.http.HttpServletRequest request,
          final javax.servlet.http.HttpServletResponse response)
          throws java.io.IOException, javax.servlet.ServletException {
//...
try {
      response.setContentType("text/html");
      pageContext = _jspxFactory.getPageContext(this, request, response, null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("<!DOCTYPE html>\r\n");
      out.write("<html>\r\n");
      out.write("<head>\r\n");
      out.write("<title>Saludo JSP</title>\r\n");
      out.write("</head>\r\n");
      out.write("<body>\r\n");
      out.write("<h1>Hola\r\n");
 // ...
```

JSP

- El texto HTML se denomina plantilla
- Los ficheros JSP deben tener la extensión .jsp
  - Se traducen en un servlet, que será compilado automáticamente
  - Se ejecutará el servlet generado
  - El cliente no
- En eclipse se crean dentro de WebContent
  - Igual que los ficheros .html
- El código Java se enmarca de varias maneras:
  - **< % =** expresión **% >**
    - El resultado de evaluar la expresión se inserta en la página HTML
  - **< %** código **% >**
    - Un ***scriptlet***
    - El código se insertará en el método de servicio del servlet
  - **< % !** declaraciones **% >**
    - Las declaraciones se insertan en la clase, no en un método
  - **< % --** Comentario **-- % >**
    - Comentario JSP

Objetos predefinidos

- **request**
  - Objeto HttpServletRequest que permite acceder a la información de la solicitud
- **response**
  - Objeto HttpServletResponse para generar la respuesta
- **session**
  - Objeto HttpSession asociado a la petición
  - Si no hubiera sesión será null
- **out**
  - Objeto JspWriter (similar a un PrintWriter) para generar la salida para el cliente
- **application**
  - El objeto ServletContext del contenedor web

Expresiones

- **< % =** expresión Java **% >**
  - El resultado de evaluar la expresión será un String que pasará a formar parte de la página HTML
    - Se genera un servlet donde el resultado de la expresión se pone como *out.println(expresión)* dentro del método *_jspService()*

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
<title>Hora</title>
</head>
<body>
<p>Hora actual: <%= new java.util.Date() %></p>
<p>Tu IP: <%= request.getRemoteHost() %></p>
</body>
</html>
```


Scriptles

- Es la forma de insertar código Java en JSP
  - Entre los símbolos <% y %>
  - Este código se insertará tal cual en el método *_jspService*() del servlet generado
  - En el scriptlet, el texto a generar de la página HTML tiene que ponerse con *out.print ()*
- Normalmente es más práctico usar scriptlets que expresiones
  - Muchas veces hay que comprobar valores, realizar acciones más complejas, etc.
    - Por ejemplo, en vez de la expresión siguiente

      `<p>Autor = <%= application.getInitParameter("Autor") %></p>`
    - Mejor el scriptlet:

```
<%
String autor = application.getInitParameter("Autor");
if ((autor == null)||(autor.trim().equals("")))
       autor = "Anónimo";
out.println("Autor = "+autor);
%>
```

Partes condicionales

- Con scriptles se puede condicionar la ejecución de partes del fichero JSP
  - No obstante este mecanismo puede dar lugar a código poco claro

```
<p>
<% String parametro = request.getParameter("nombre");
if ((parametro == null) || (parametro.trim().equals(""))) { %>
No nos has dado tu nombre.
<% } else { %>
Bienvenido,
<% out.println(parametro); } %>
</p>
```


Declaraciones

- Se pueden incluir declaraciones en la clase del servlet generado con <%! declaración %>
  - Este código se inserta fuera de los métodos de la clase, como nuevas declaraciones en la clase
    - Variables del servlet

      ```
      <%! private int edad; %>
      ```

      - Si se declaran variables con <% … %> serán locales al scriptlet
    - Métodos
      - Es mejor declararlo en una clase Java aparte

```
<%! private int contador = 0; %>
<p>Número de veces que se ha visitado esta página desde que se
arrancó el servidor:
<%= ++contador %>
</p>
```

Directivas

- Se aplican a la clase servlet generada
- Sintaxis:

```
<%@ directiva  atributo="valor" %>
```

o bien:

```
<%@ directiva atributo1="valor1"
              atributo2="valor2"
              ...
              atributoN="valorN" %>
```

- Directivas comunes
  - **include** – permite incluir otro fichero que se tratará como JSP
    - Puede tratarse de un fichero JSP, HTML, JavaScript, etc.
    - El fichero se referencia con una URL relativa a la página JSP o al servidor si va precedido de /

```
<%@ include file="/URL" %>
```

  - **page** – permite importar un paquete

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


Uso de Java Beans con JSP

- Los Java Beans son componentes Java que se usan habitualmente en aplicaciones Web para gestionar la lógica de negocio
- Se pueden utilizar en JSP para obtener información a visualizar
  - Más adelante se verá la arquitectura MVC donde JSP implementa la vista

Java Beans

- Clases Java que cumplen varias convenciones
  - **Declarados dentro de un paquete**
  - **Constructor sin argumentos**
    - O que no se defina ningún constructor
  - Todos los atributos son **private**
    - Estos atributos se denominan ***propiedades***
  - Métodos de acceso a las propiedades
    - **getPropiedad**() para lectura
      - Para los booleanos **isPropiedad**()
    - **setPropiedad(valor)** para escritura
  - Métodos para realizar funciones más complejas
- En Eclipse se crean con facilidad
  - Crear una clase Java (dentro de un paquete)
    - Para aplicaciones web, dentro del src/paquete
  - Especificar las propiedades como atributos private
  - Source → Generate getters and setters…
    - Se crean los métodos get y set que interesen

Java Beans

- Ejemplo de un Bean cliente

```
public class Cliente {
     private String nombre;
     private String nif;
     private String email;
     private String direccion;
     private String telefono;

public String getTelefono() {
     return telefono;
}
public void setTelefono(String telefono) {
     this.telefono = telefono;
}
public String getNombre() {
     return nombre;
}
public void setNombre(String nombre) {
     this.nombre = nombre;
}
```

```
public String getDireccion() {
     return direccion;
}
public void setDireccion(String direccion) {
     this.direccion = direccion;
}
public String getNif() {
     return nif;
}
public String getEmail() {
     return email;
}

}
```

JSP y Java Beans

- JSP proporciona varias etiquetas para usar Java Beans
  - **jsp:useBean**
    - Crea un Java Bean

```
<jsp:useBean id="nombreBean" class="paquete.Clase" />
```

    - Equivale a

```
<% paquete.Clase nombreBean = new paquete.Clase(); %>
```

  - **jsp:setProperty**
    - Modifica una propiedad llamando al método setPropiedad()

```
<jsp:setProperty name="nombreBean"
    property="propiedad" value="valor" />
```

    - Equivale a

```
<% nombreBean.setPropiedad("valor"); %>
```

  - **jsp:getProperty**
    - Obtiene el valor de una propiedad llamando a getPropiedad()

```
<jsp:getProperty name="nombreBean" property="propiedad" />
```

    - Equivale a

```
<%= nombreBean.getPropiedad() %>
```


Versión XML de JSP

- Si se quiere utilizar JSP con aplicaciones XML habrá que adaptar la sintaxis a XML
  - XHTML
  - Servicios Web
  - Aplicaciones Ajax
- Correspondencia en XML:
  - HTML: <%= expression %>
    XML: <jsp:expression>expresión</jsp:expression>
  - HTML: <% code %>
    XML: <jsp:scriptlet>código</jsp:scriptlet>
  - HTML: <%! declarations %>
    XML: <jsp:declaration>declaraciones</jsp:declaration>
  - HTML: <%@ include file=URL %>
    XML: <jsp:directive.include file="URL"/>

Versión XML de JSP

```
<?xml version="1.0" encoding="UTF-8" ?>
<html xmlns:jsp="http://java.sun.com/JSP/Page">
<jsp:output
omit-xml-declaration="true"
doctype-root-element="html"
doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd" />
<jsp:directive.page contentType="text/html"/>
<head>
<title>Título de la página</title>
</head>
<body>
Cuerpo con scriptles, etc.
</body>
</html>
```

Espacio de nombre para las etiquetas jsp:XXX

Declaración del tipo de documento

Acciones

- Etiquetas JSP con sintaxis XML para controlar el intérprete de servlets

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

  - Incluye el URL indicado en tiempo de ejecución en vez de en tiempo de compilación (que es lo que hace la directiva include)

- Esto sirve para datos que cambian con frecuencia

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

  - Salta al URL (estático) o al URL resultante de la expresión Java (calculado dinámicamente)

Ventajas de JSP

- PHP, ASP, ColdFusion
  - Al tener acceso a todo Java, JSP es más flexible y mejor lenguaje para la parte dinámica
  - Más herramientas de desarrollo (muchas de código abierto)
- Servlets
  - Mejor para crear y mantener el código (X)HTML
  - Se puede editar en herramientas típicas de desarrollo web
  - Aunque antes sigue siendo necesario conocer cómo programar servlets

Bibliografía

- Eric Jendrock et al. The Java EE 6 Tutorial (2013). http://docs.oracle.com/javaee/6/tutorial/doc/
- Building Web Apps in Java: Beginning & Intermediate Servlet & JSP Tutorials. http://courses.coreservlets.com/Course-Materials/csajsp2.html
- Documentación oficial
  - Java EE Specifications http://www.oracle.com/technetwork/java/javaee/tech/index.html
  - API specification for version 6 of Java EE http://docs.oracle.com/javaee/6/api/