

JSP's

- Una JSP es un documento de texto “parecido a HTML” que describe como procesar una petición para crear una respuesta.
- Permiten separar la generación de interfaz de usuario (HTML) de la lógica de negocio.

Elementos sintácticos de una JSP.

- Una página JSP contiene **elementos** (cosas que conoce y sabe lo que son) y datos de plantilla (el resto de cosas). Hay tres tipos de elementos:
 - **Directivas:** Proveen información para la fase de traslación, por tanto no dependen de la petición concreta. Tienen el formato:


```
<%@ directiva ...%>
```
 - **Elementos de script:** Permiten insertar código Java dentro del servlet que se generará a partir de la página.
 - **Elementos de acción:** Proveen información para la fase de ejecución. Su ejecución dependerá en muchos casos de la petición. Los elementos de acción pueden ser estándares o definidos como extensiones. Tienen la sintaxis de un elemento XML.

Elementos de script.

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

- **Expresiones**

`<%= expresión %>`

que son evaluadas y retornan un String que se inserta en esa posición en la página.

- **Scriptlets**

`<% código %>`

El código se inserta dentro del método service del servlet en la fase de compilación.

- **Declaraciones**

`<%! código %>`

que se insertan en el cuerpo de la clase del servlet, fuera de cualquier método existente.

Hola mundo

```
<HTML>  
<BODY>  
Hola, mundo  
</BODY>  
</HTML>
```

Hola.jsp

Expresiones

```
<HTML>  
<BODY>  
¡Hola! Son las <%= new java.util.Date() %>  
</BODY>  
</HTML>
```

HolaHora.jsp

Scriptlets

```
<HTML>
<BODY>
<%
    // Esto es un scriptlet. La variable fecha que
    // declaramos aquí esta disponible después en
    // la expresión..
    System.out.println( "Calculando la fecha" );
    java.util.Date fecha = new java.util.Date();
%>
¡Hola! Son las <%= fecha %>
</BODY>
</HTML>
```

Scriptlets

- Ejemplos:

```
<%
```

```
String queryData = request.getQueryString();  
out.println("Attached GET data: " + queryData);  
%>
```

```
<% if (Math.random() < 0.5) { %>
```

```
Have a <B>nice</B> day!
```

```
<% } else { %>
```

```
Have a <B>lousy</B> day!
```

```
<% } %>
```

Variables predefinidas

- `request`: La petición HTTP
- `response`: La respuesta HTTP
- `session`: La sesión asociada a la petición actual, caso de existir.
- `out`: El `PrintWriter`

Variables predefinidas. out

```
<HTML>
<BODY>
<%
    // Este scriptlet declara e inicializa fecha.
    System.out.println( "Calculando la fecha" );
    java.util.Date fecha = new java.util.Date();
%>
¡Hola! Son las
<%
    // Este scriptlet genera salida HTML
    out.println( String.valueOf( date ));
%>
</BODY>
</HTML>
```

Variables predefinidas. request

```
<HTML>
<BODY>
<%
    // Este scriptlet declara e inicializa fecha.
    System.out.println( "Calculando la fecha" );
    java.util.Date fecha = new java.util.Date();
%>
¡Hola! Son las
<%
    out.println( String.valueOf( date ));
    out.println( "<BR>La dirección de tu máquina es" );
    out.println( request.getRemoteHost());
%>
</BODY>
</HTML>
```

Mezclando HTML con scriptlets

```
<TABLE BORDER=2>
<%
  for ( int i = 0; i < n; i++ ) {
    %>
    <TR>
    <TD>Número</TD>
    <TD><%= i+1 %></TD>
    </TR>
    %>
  }
%>
</TABLE>
```

Mezclando HTML con scriptlets (II)

```
...  
<%  
    if ( hola ) {  
        %>  
        <P>Hola mundo  
        <%  
    } else {  
        %>  
        <P>Adios mundo cruel  
        <%  
    }  
%>  
...
```

Declaraciones

- Permiten definir métodos o campos que se insertarán en el código de la clase Servlet, fuera del método service.

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

Accesos a la página desde la última vez que se cargó el servlet:

```
<%= ++accessCount %>
```

JSP's. Fases de ejecución.

- De cara a su ejecución, una JSP pasa por dos fases:
 - **Fase de traslación** (compilación). Mediante este proceso se pasa de un archivo JSP a una clase Java compilada que implementa la interficie Servlet.
 - **Fase de ejecución**. Cuando llega una petición que va destinada a ser servida por una JSP, el contenedor de servlets localiza (o crea si no existe) una instancia de la clase resultado de la compilación de esa JSP y se invoca su método service.

Directivas JSP.

Existen tres directivas:

- **page**: define características de la página.
- **taglib**: permite utilizar un taglib con elementos de acción definidos como extensiones.
- **include**: permite incluir otra página durante la fase de compilación.

Directiva page.

Ejemplos:

```
<%@ page info="my latest JSP Example" %>
```

```
<%@ page buffer="none" errorPage="/oops.jsp" %>
```

```
<%@ page language="java" import="com.myco.*" buffer="16kb" %>
```

```
<%@ page page_directive_attr_list %>
```

```
page_directive_attr_list ::= { language="scriptingLanguage"
                                { extends="className"           }
                                { import="importList"           }
                                { session="true|false"          }
                                { buffer="none|sizekb"           }
                                { autoFlush="true|false"         }
                                { isThreadSafe="true|false"      }
                                { info="info_text"              }
                                { errorPage="error_url"          }
                                { isErrorPage="true|false"       }
                                { contentType="ctinfo"          }
                                { pageEncoding="peinfo"          }
                                { isELIgnored="true|false"       }
```


Tags JSP

- JSP nos permite definir librerías de tags, para automatizar las tareas más comunes sin incrustar código en las páginas JSP.
- La Java Standard Tag Library (JSTL) nos ofrece un conjunto de tags de uso común.

```
<table>
```

```
<c:forEach var="name" items="$ {customerNames}">
```

```
<tr><td><c:out value="$ {name}"/></td></tr>
```

```
</c:forEach>
```

```
</table>
```

Directiva taglib.

Ejemplos:

```
<%@ taglib uri="http://www.mycorp/supertags" prefix="super" %>  
...  
<super:doMagic>  
...  
</super:doMagic>
```

Sintaxis:

```
<%@ taglib ( uri="tagLibraryURI" | tagdir="tagDir" ) prefix="tagPrefix" %>
```

Directiva include.

Permite incluir una página durante la fase de compilación (por tanto HTML independiente de la petición)

Ejemplo:

```
<%@ include file="copyright.html" %>
```

Sintaxis:

```
<%@ include file="relativeURLspec" %>
```

Acciones

- Las acciones 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, etc.

- **jsp:include**: Permite insertar un fichero en la respuesta HTTP durante la fase de ejecución.

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

- **jsp:forward**: Reenvía a otra página.

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

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

Directivas y elementos de script (.jsp)

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

```
<HTML>
```

```
<HEAD> ... Aquí habían cosas no interesantes ...
```

```
</HEAD>
```

```
<BODY BGCOLOR="#FDF5E6" TEXT="#000000" LINK="#0000EE"
      VLINK="#551A8B" ALINK="#FF0000">
```

```
<CENTER>
```

```
<table BORDER=5 BGCOLOR="#EF8429">
```

```
  <TR><TH CLASS="TITLE">
```

```
    Using JavaServer Pages</table>
```

```
</CENTER>
```

```
<P>
```

Some dynamic content created using various JSP mechanisms:

```
<UL>
```

```
  <LI><B>Expression.</B><BR>
```

```
    Your hostname: <%= request.getRemoteHost() %>.
```

```
  <LI><B>Scriptlet.</B><BR>
```

```
    <% out.println("Attached GET data: " +
                  request.getQueryString()); %>
```

```
  <LI><B>Declaration (plus expression).</B><BR>
```

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

```
    Accesses to page since server reboot: <%= ++accessCount %>
```

```
  <LI><B>Directive (plus expression).</B><BR>
```

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

```
    Current date: <%= new Date() %>
```

```
</UL>
```

```
</BODY>
```

```
</HTML>
```



Directivas y elementos de script (resultado)

Accesso a Java Beans (BeanTest.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="test" class="hall.SimpleBean" />
<jsp:setProperty name="test"
                  property="message"
                  value="Hello WWW" />

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

</BODY>
</HTML>
```

Acceso a Java Beans (SimpleBean.java)

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


Accesso a Java Beans II (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="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>
```

Accesso a Java Beans II (NumberedPrimes.java)

```
import java.util.*;

public class NumberedPrimes {
    private Vector primes;
    private int numPrimes = 15;
    private int numDigits = 50;

    public String getNumberedList() {
        if (primes == null) {
            PrimeList newPrimes =
                new PrimeList(numPrimes, numDigits, false);
            primes = newPrimes.getPrimes();
        }
        StringBuffer buff = new StringBuffer("<OL>\n");
        for(int i=0; i<numPrimes; i++) {
            buff.append("  <LI>");
            buff.append(primes.elementAt(i));
            buff.append("\n");
        }
        buff.append("</OL>");
        return(buff.toString());
    }

    public int getNumPrimes() {
        return(numPrimes);
    }
}
```

Accesso a Java Beans II (NumberedPrimes.java 2)

```
public void setNumPrimes(int numPrimes) {
    if (numPrimes != this.numPrimes)
        primes = null;
    this.numPrimes = numPrimes;
}

public int getNumDigits() {
    return(numDigits);
}

public void setNumDigits(int numDigits) {
    if (numDigits != this.numDigits)
        primes = null;
    this.numDigits = numDigits;
}

public Vector getPrimes() {
    return(primes);
}

public void setPrimes(Vector primes) {
    this.primes = primes;
}
}
```

Acceso a Java Beans II (resultado)

