Escuela Politécnica de Ingeniería
Grado de Ingeniería Informática en Tecnologías de la Información

Tecnologías Web

Tema 3

Tecnologías web de servidor: JEE (JSPs) 1/2

Índice

- Encadenamiento de Servlets/JSPs
- Introducción
- Elementos de JSP
  - Elementos de secuencias/Scripting
  - Directivas
  - Acciones

Encadenamiento de Servlets/JSPs

- Desde un Servlet concatenar otro Servlet:
  - RequestDispatcher/forward/include
- Desde un JSP
  - <jsp:forward>
  - <jsp:include>

Cómo reenviar peticiones

- Para reenviar peticiones o incluir un contenido externo se emplea:

  RequestDispatcher
  getServletContext().getRequestDispatcher(URL)

- Ejemplo:

  String url = “/presentaciones/presentacion1.jsp”;

  RequestDispatcher
  despachador=getServletContext().getRequestDispatcher(url);

  //Para pasar el control total usar **forward**

  despachador.forward(request, response);

  //forward genera las excepciones ServletException y IOException

Ejemplo de reenvío de peticiones

```
public void doGet(HttpServletRequest request,  HttpServletResponse response)
                    throws ServletException, IOException {
   String operation = request.getParameter("operation");
   if (operation == null) {
       operation = "unknown"; }

   if (operation.equals("operation1")) {
       gotoPage("/operations/presentation1.jsp", request, response); }
   else if (operation.equals("operation2")) {
       gotoPage("/operations/presentation2.jsp",request,response); }
   else {     gotoPage("/operations/unknownRequestHandler.jsp",  request, response); }
}
private void gotoPage(String address, HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
      RequestDispatcher dispatcher =    getServletContext().getRequestDispatcher(address);
      dispatcher.forward(request, response);
}
```

Paso de información procesada a Servlet/ JSP

- Criterio: El servlet procesará los datos y se los pasará a la/s página/s JSP cuando:
  - Los datos son complejos de procesar (el servlet es más adecuado para ello).
  - Son varias las páginas JSP las que pueden recibir los mismos datos (el servlet los procesa de forma más eficiente).
- Formas de pasar datos a un JSP desde un Servlet:
  - Almacenándolos en HttpServletRequest.
  - En el URL de la página objetivo.
  - Pasándolos con un Bean.
- Paso de datos en **HttpServletRequest**
  - En el servlet: **request.setAttribute("clave1", valor1);**
  - En JSP: **request.getAttribute("clave1");**
- Pasar datos en el **URL**
  - En el servlet:

    address ="/ruta/recurso.jsp?clave1=valor1"
    RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(address);
    dispatcher.forward(request, response);
  - En JSP: el nuevo parámetro se agrega al principio de los datos consultados.
    request.getParameter("clave1");

Cómo interpretar los URLs relativos en la página objetivo

- Un servlet puede reenviar peticiones a lugares arbitrarios en mismo servidor mediante forward(). Diferencias con sendRedirect():
  - sendRedirect ()
    - Necesita que el cliente se vuelva a conectar al nuevo recurso
    - No conserva todos los datos de la conexión
    - Trae consigo un URL final distinto
  - forward()
    - Se maneja internamente en servidor
    - Conserva los datos de la conexión
    - Conserva el URL relativo del servlet
- Si la página objetivo trae URLs relativos a la raíz del servidor y no a su propio directorio como el ejemplo siguiente:

<LINK REL=STYLESHEET HREF="mis_estilos.css" TYPE=text/css>

- En el caso de una redirección a esta página, se considerará mis_estilos.css relativo a la dirección del servlet que genera la etiqueta LINK y no a la página JSP generando un error. **Solución:**

<LINK REL=STYLESHEET HREF="(/ruta/mis_estilos.css" TYPE=text/css>

- Lo mismo para : <IMG SRC=…, <A HREF=…, …

Cómo incluir contenido estático o dinámico

- Si un servlet usa el método RequestDispatcher. forward() no podrá enviar ningún resultado al cliente, tendrá que dejarlo todo a la página objetivo.
- Para mezclar resultados del propio servlet con la página JSP o HTML se deberá emplear **RequestDispatcher.include()**:

```
out.println("…");
requestDispatcher dispatcher =   getServletContext().getRequestDispatcher(address);
dispatcher.include(request, response);
out.println("…");
```

- Las diferencias con una redirección consiste en:
  - Se pueden enviar datos al navegador antes de hacer la llamada. (out.println("…");)
  - El control se devuelve al servlet cuando finaliza include.
  - Las páginas JSP, servlets, HTML incluidas por el servlet no deben establecer encabezados HTTP de respuesta.
- include() se comporta igual que forward propagando toda la información de la petición original (GET o POST, parámetros de formulario, …) y además establece
  - **Javax.servlet.include.request_uri, context_path, servlet_path, path_info y query_string**

Ejemplo de inclusión de datos desde el servlet principal

```
public class ShowPage extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
                    throws ServletException, IOException {
    response.setContentType("text/html");  PrintWriter out = response.getWriter();
    String url = request.getParameter("url");
    out.println(ServletUtilities.headWithTitle(url) +"<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=CENTER>" + url + "</H1>\n" +"<FORM><CENTER>\n" +
            "<TEXTAREA ROWS=30 COLS=70>");
    if ((url == null) || (url.length() == 0)) { out.println("No URL specified."); }
    else {
      String data = request.getParameter("data");
      if ((data != null) && (data.length() > 0)) {
        url = url + "?" + data;
      }
      RequestDispatcher dispatcher=getServletContext().getRequestDispatcher(url);
      dispatcher.include(request, response);
    }
    out.println("</TEXTAREA>\n" + "</CENTER></FORM>\n" + "</BODY></HTML>");
  }
...
};
```

¿Qué es JSP?

- Una tecnología para crear páginas Web dinámicas
  - Contienen código HTML normal junto a elementos especiales de JSP
- Están construidas sobre **servlets**
  - Cuando se solicita una página JSP, la primera vez se compilará el código Java a un Servlet y se cargará en el Servidor de Servlets. El código HTML se adjuntará al código de salida del método service() del Servlet.

- Vienen a resolver el problema de aquellos (que también tenían los CGI):
  - Generar HTML directamente por código
    - Dificulta enormemente la **separación de tareas** entre **diseñadores** y **programadores**

Ejemplo de página JSP

```
<html>
  <head>
    <title>Saludo personalizado con JSP</title>
  </head>
  <body>

    <% java.util.Date hora = new java.util.Date(); %>

    <% if (hora.getHours() < 12) { %>
      <h1>¡Buenos días!</h1>
    <% } else if (hora.getHours() < 21) { %>
      <h1>¡Buenas tardes!</h1>
    <% } else { %>
      <h1>¡Buenas noches!</h1>
    <% } %>

    <p>Bienvenido a nuestro sitio Web, abierto las 24 horas del día.</p>

  </body>
</html>
```

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
        throws IOException, ServletException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();

    out.println("<html>");
    out.println("<head>");
    out.println("<title>Saludo personalizado con JSPout.println</title>");
    out.println("</head>");
    out.println("<body>");

    java.util.Date hora = new java.util.Date();
    if (hora.getHours() out.println("< 12) {
        out.println("<h1>¡Buenos días!</h1>");
    }
    else if (hora.getHours() out.println("< 21) {
        out.println("<h1>¡Buenas tardes!</h1>");
    }
    else {
        out.println("<h1>¡Buenas noches!</h1>");
    }

    out.println("<p>Bienvenido a nuestro sitio Web, abierto las 24 horas del día.</p>");
    out.println("</body>");
    out.println("</html>");
}
```

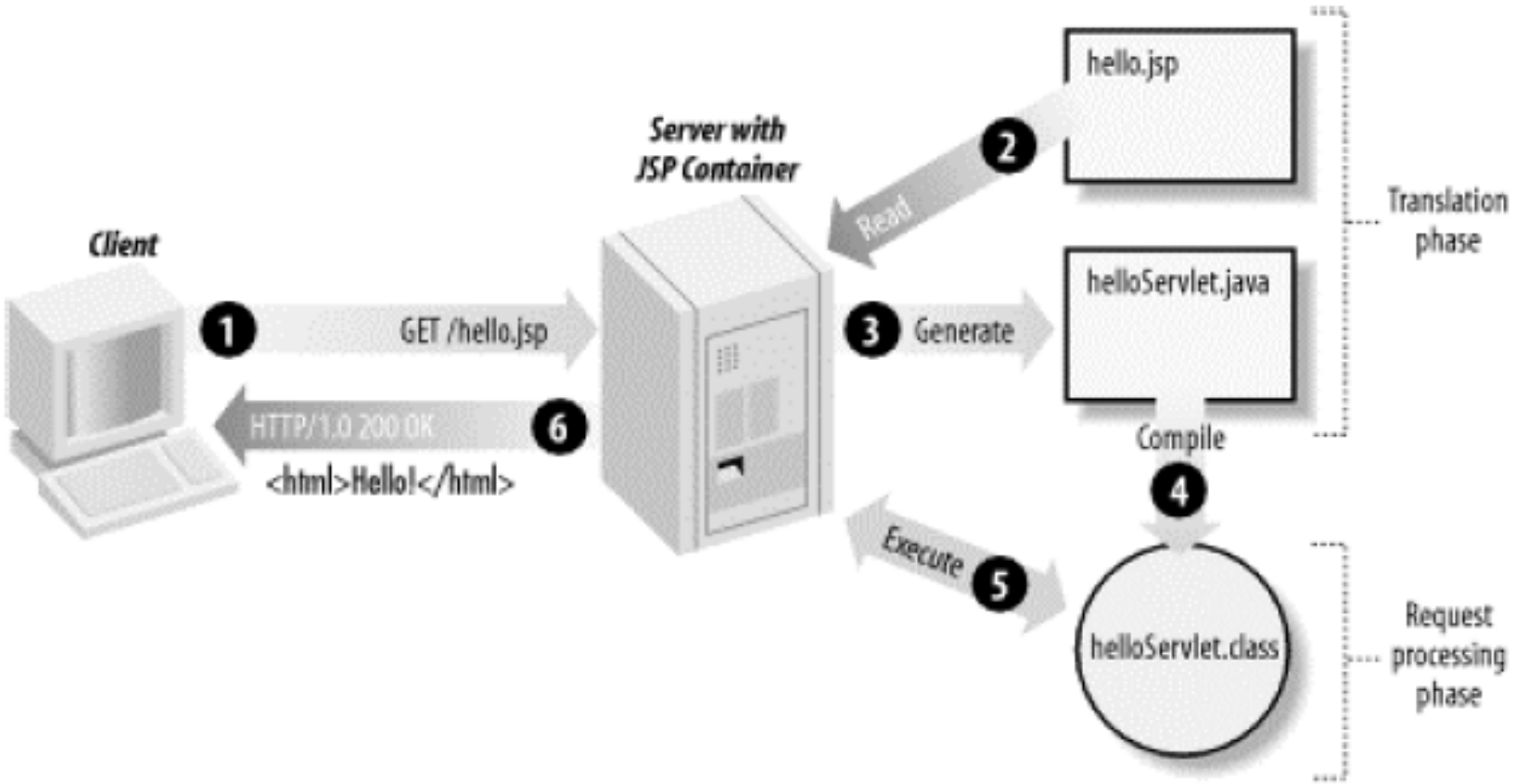
¿Beneficio?

- Incluir mucha lógica de programación en una página Web no es mucho mejor que generar el HTML por programa
  - Pero JSP proporciona acciones (***action elements***) que son como etiquetas HTML pero que representan código reutilizable
  - Además, se puede invocar a otras clases Java del servidor, a componentes (**Javabeans** o **EJB**) …

Separación de presentación y lógica

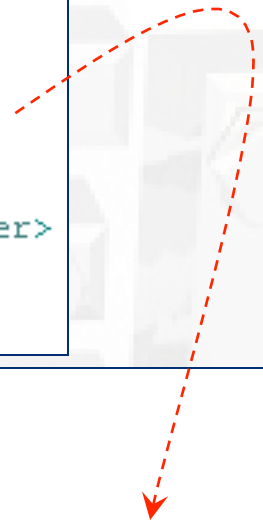
- En definitiva, lo que permite JSP (bien utilizado) es una **mayor separación entre la presentación de la página y la lógica de la aplicación**, que iría aparte
  - Desde la página JSP únicamente invocaríamos, de diferentes formas, a ese código

JSP: Proceso de compilación



Ejemplo: de JSP a Servlet (Tomcat)

```
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
    <center>Bienvenido a mi primera página Web!</center>
</BODY>
</HTML>
```



```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class index_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

  public void _jspInit() {
  }

  public void _jspDestroy() {
  }

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

JSP a Servlet.service()

```
public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws java.io.IOException, 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("<HTML>\r\n");
    out.write("<HEAD>\r\n");
    out.write("<TITLE>Hola Mundo!</TITLE>\r\n");
    out.write("</HEAD>\r\n");
    out.write("<BODY>\r\n");
    out.write("\t<center>Bienvenido a mi primera página Web!</center>\r\n");
    out.write("</BODY>\r\n");
    out.write("</HTML>\r\n");
    out.write("\r\n");
  } catch (Throwable t) {
    if (!(t instanceof SkipPageException)){
      out = _jspx_out;
      if (out != null && out.getBufferSize() != 0)
        try { out.clearBuffer(); } catch (java.io.IOException e) {}
      if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
    }
  } finally {
    _jspxFactory.releasePageContext(_jspx_page_context);
```

Índice

- Introducción
- Elementos de JSP
  - Elementos de secuencia/Scripting
  - Directivas
  - Acciones

Elementos JSP

- Tres tipos de elementos en JSP:
  - Scripting
    - Permiten insertar código java que será ejecutado **en el momento de la petición**
  - Directivas
    - Permiten especificar información acerca de la página que permanece constante para todas las peticiones
      - Requisitos de buffering
      - Página de error para redirección, etc.
  - Acciones
    - Permiten ejecutar determinadas acciones sobre información que se requiere en el momento de la petición de la JSP
      - Acciones estándar
      - Acciones propietarias (Tag libs)

Elementos de *"scripting"*

Elemento	Descripción
<% . . . %>	**Scriptlet**. Encierra código Java
<%= . . . %>	**Expresión**. Permite acceder al valor devuelto por una expresión en Java e imprimirlo en OUT
<%! . . . %>	**Declaración**. Usada para declarar variables y métodos en la clase correspondiente a la página
<%-- ... --%>	**Comentario**. Comentario ignorado cuando se traduce la página JSP en un servlet. (comentario en el HTML <!-- comment →)

Un ejemplo más “dinámico”

- Hagamos una página JSP que, en función de la hora, muestre un saludo diferente (buenos días, buenas tardes o buenas noches)
  - Y que muestre también la hora actual

Ejemplo: expresión, saludo.jsp

```
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
    <center>Bienvenido a mi primera página Web!</center>
    <p>Son las <%= new java.util.Date() %></p>
</BODY>
</HTML>
```

<%=

expresión

Nota: Si se necesita usar los caracteres "%>" dentro de un scriptlet, hay que usar "%\>" y "<\%"

Ejemplo: scriptlet, saludo.jsp

```
<%
    String saludo;
    java.util.Date hora = new java.util.Date();

    if (hora.getHours() < 13){
        saludo = "Buenos días";
    }
    else if (hora.getHours() < 20){
        saludo = "Buenas tardes";
    }
    else {
        saludo = "Buenas noches";
    }
%>
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
    <center>Bienvenido a mi primera página Web!</center>
    <p>Son las <%= hora %>, <%= saludo %></p>
</BODY>
</HTML>
```

<%

Ejemplo: declaración, saludo.jsp

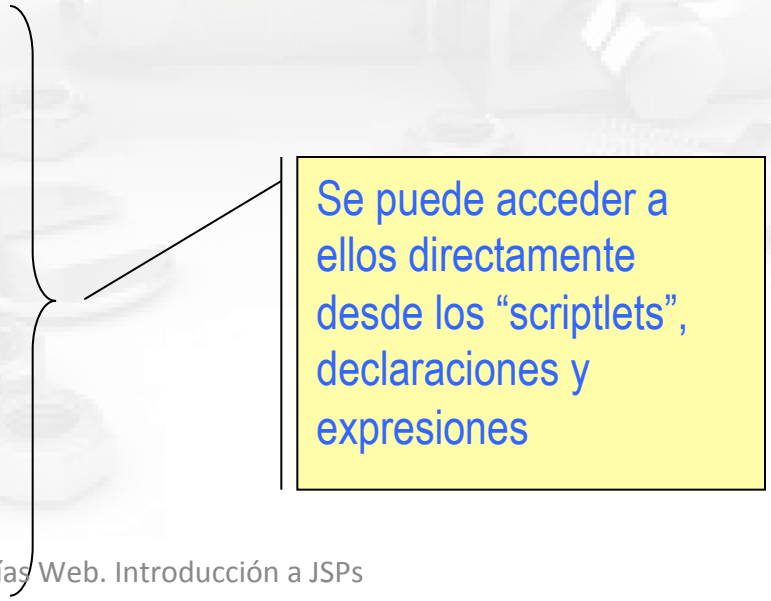
```
<%!
private java.util.Date hora = new java.util.Date();

java.util.Date getHora(){
    return hora;
}

String getSaludo() {
    String saludo;
    if (hora.getHours() < 13) {
        saludo = "Buenos días";
    } else if (hora.getHours() < 20) {
        saludo = "Buenas tardes";
    } else {
        saludo = "Buenas noches";
    }
    return saludo;
}
%>
```

<%!

```
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
<center>Bienvenido a mi primera página Web!</center>
<p>Son las <%=getHora() %>, <%=getSaludo()%></p>
</BODY>
</HTML>
```

JSP: objetos predefinidos

- El código java incrustado en JSP tiene acceso a los mismos objetos predefinidos que tenían los servlets
- Aquí se denominan:
  - `request`
  - `response`
  - `pageContext`
  - `session`
  - `application`
  - `out`
  - `config`
  - `page`



JSP: Objetos predefinidos

- **request:**
  - HttpServletRequest asociado con la petición
  - Permite acceder a:
    - Los parámetros de la petición (mediante `getParameter`)
    - El tipo de petición (GET, POST, HEAD, etc.)
    - Las cabeceras HTTP entrantes (cookies, 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

JSP: Objetos predefinidos

- **response:**
  - Objeto de clase HttpServletResponse asociado con la respuesta al cliente
  - Como el stream de salida 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

JSP: Objetos predefinidos

- **out:**
  - PrintWriter usado para enviar la salida al cliente
  - Esta es una versión con buffer de PrintWriter llamada JspWriter
    - Podemos ajustar el tamaño del buffer, o incluso desactivarlo, usando el atributo buffer de la directiva page
  - 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*

JSP: Objetos predefinidos

- **session:**
  - HttpSession asociado con la petición
  - 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

JSP: Objetos predefinidos

- **application:**
  - El `ServletContext` obtenido mediante `getServletConfig().getContext()`
- **config:**
  - El objeto ServletConfig
- **pageContext:**
  - PageContext es un nuevo contexto, además de Session y ServletContext
  - Permite acceder a writer especifico para JSP y establecer atributos en ámbito de página

Índice

- Introducción
- Elementos de JSP
  - Elementos de secuencias/Scripting
  - Directivas
  - Acciones

Directivas

- Las directivas son mensajes para el contenedor de JSP
- Ejemplos:

Elemento	Descripción
`<%@ page ... %>`	Permite importar clases Java, especificar el tipo de la respuesta (“text/html” por omisión), etc.
`<%@ include ... %>`	Permite incluir otros ficheros antes de que la página sea traducida a un servlet
`<%@ taglib ... %>`	Declara una biblioteca de etiquetas con ***acciones*** personalizadas para ser utilizadas en la página

Forma general: <%@ directive { attr=”value” }* %>

JSP: Directiva “page”

- **import**="package.class“ o import="packg.class1,...,packg.classN".
  - Permite especificar los paquetes que deberían ser importados. El atributo import es el único que puede aparecer múltiples veces
- **ContentType** = "MIME-Type" o contentType = "MIME-Type; charset = Character-Set"
  - Especifica el tipo MIME de la salida. El valor por defecto es “text/html”. Tiene el mismo valor que el scriptlet usando “response.setContentType”
- **isThreadSafe** = "true|false".
  - True (por defecto) indica procesamiento del servlet normal, múltiples peticiones pueden procesarse simultáneamente con un sólo ejemplar del servlet (el autor sincroniza los recursos compartidos). Un valor de false indica que el servlet debería implementar SingleThreadModel.

JSP: Directiva “page”

- **session** = "true|false"
  - True (por defecto) indica que la variable predefinida session (del tipo HttpSession) debería unirse a la sesión existente si existe una; si no existe se debería crear una nueva sesión para unirla. False indica que no se usarán sesiones, y los intentos de acceder a la variable session resultarán en errores en el momento en que la página JSP sea traducida a un servlet
- **buffer** = "sizekb|none"
  - Esto especifica el tamaño del buffer para el JspWriter out. El valor por defecto es específico del servidor y debería ser de al menos 8kb.
- **autoflush, extends, info, errorPage, isErrorPage, language**

Ejemplo directiva “page”

- Modificar la página de saludo para que en lugar de hacer referencia directamente a java.util.Date importe el paquete java.util

```
<%@ page import="java.util.Date"
    contentType="text/html"
    pageEncoding="iso-8859-1" %>
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
<center>Bienvenido a mi primera página Web!</center>
<p>Son las <%= new Date() %></p>
</BODY>
</HTML>
```

Juego de Caracteres UNICODE latin-1

JSP: Directiva “include”

- Permite incluir ficheros en el momento en que la página JSP es traducida a un servlet

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

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

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

Ejemplo: directiva include

```jsp
<%@ page import="java.util.Date"
    contentType="text/html"
    pageEncoding="iso-8859-1"  %>

<p>Son las <%= new Date() %></p>
```

```jsp
<%@ page contentType="text/html"
    pageEncoding="iso-8859-1"  %>
<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
<center>Bienvenido a mi primera página Web!</center>
<%@ include file="date.jsp" %>
</BODY>
</HTML>
```

Directiva “taglib”

- Declara que la página usa una librería de tags (acciones)

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

  - Identifica la librería por su URI
  - Le asocia un prefijo para usar en el código de la JSP

Syntax

<%@ taglib (uri=”*tagLibraryURI*” | *tagdir=”tagDir”) prefix=”tagPrefix” %>*

Directiva “taglib”

- **uri** = “taglib.tld”
  - Absoluto o relativo. Permite especificar el fichero de descripción de la librería (.tld)
- **tagdir** = “myTags”
  - Especifica el directorio bajo /WEB-INF/tags/ en el cual hay ficheros de tags (.tag ó tagx) que no están empaquetados en bibliotecas de tags
- **prefix**= “c” (“fmt”, etc.)
  - Prefijo que permite distinguir acciones cuando se importan varias bibliotecas