

# Tema 3: Tecnologías para el desarrollo en el servidor

**Servlets y Java Server Pages (JSP)**

**Java Persistence API (JPA)**

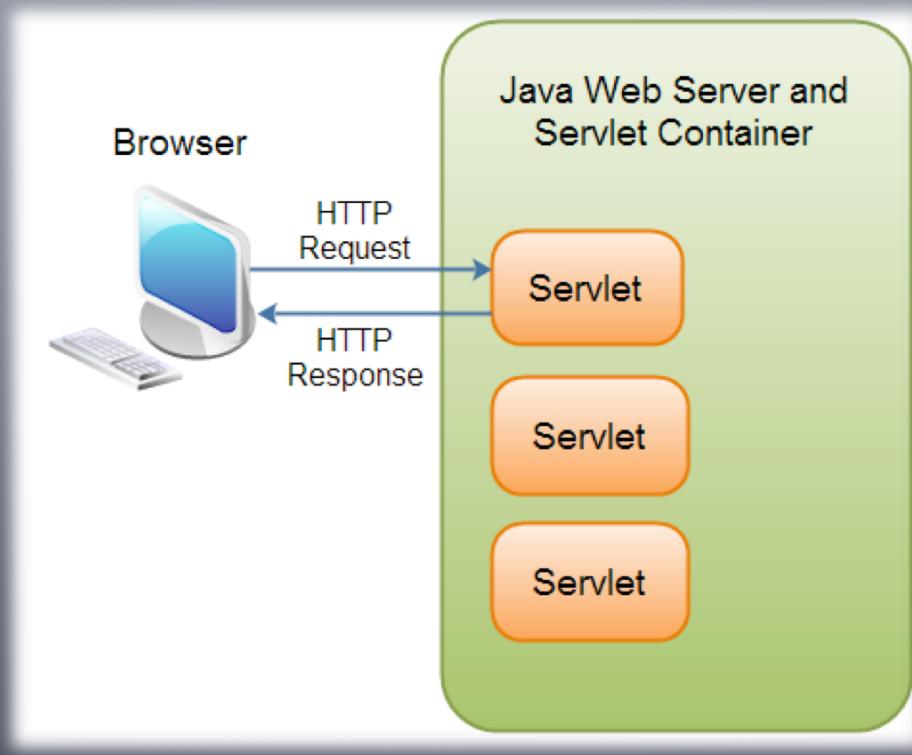
**Java Server Faces (JSF)**

**Enterprise Java Beans (EJB)**

Sistemas de Información para Internet  
3º del Grado de Ingeniería Informática (tres menciones)

Departamento de Lenguajes y Ciencias de la Computación  
Escuela Técnica Superior de Ingeniería Informática  
Universidad de Málaga

# Servlets



# Servlet

## ¿Qué es?

- Un **Servlet** es un objeto Java que se encuentra dentro del servidor de aplicaciones y responde dinámicamente a las peticiones HTTP
- La forma más sencilla de implementar uno es extendiendo la clase **HttpServlet**
- Esta clase define métodos de la forma do<Método>: **doGet**, **doPost**, **doPut**, etc.

# Servlet

## ¿Cómo se identifica?

- Además de extender `HttpServlet`, es necesario añadir una anotación de tipo `@WebServlet`, donde indicaremos el nombre del servlet y los patrones URL a los que responde
- ```
@WebServlet(name="HolaMundoServlet", urlPatterns={"/HolaMundo"})
public class HolaMundoServlet extends HttpServlet {
```
- Se accede con `http://maquina:puerto/contexto/HolaMundo`

# Servlet

## ¿Cómo se identifica?

- También puede declararse en el descriptor `web.xml`
- En versiones antiguas de Java EE era la única forma de hacerlo

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <servlet>
        <servlet-name>HolaMundoServlet</servlet-name>
        <servlet-class>tema3.HolaMundoServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>HolaMundoServlet</servlet-name>
        <url-pattern>/HolaMundo</url-pattern>
    </servlet-mapping>
</web-app>
```

# Servlet

## ¿Cómo se identifica?

- El descriptor tiene prioridad sobre las anotaciones
- Es posible evitar que se analicen las anotaciones desde el descriptor `web.xml` usando el atributo `metadata-complete=true`
- Este descriptor se encuentra en el directorio **WEB-INF**

# Servlet

## Respuesta dinámica

- Cuando un cliente acceda a alguna de las URLs indicadas en la anotación se utilizará el servlet para obtener una respuesta
- El servidor de aplicaciones ejecuta el método do<Método> dependiendo del método que haya usado el cliente
  - Para GET -> doGet
  - Para POST -> doPost

# Servlet

# Respuesta dinámica

## ■ Ejemplo:

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    response.setContentType("text/html;charset=UTF-8");
    PrintWriter out = response.getWriter();
    out.println("<!DOCTYPE HTML "
        + "PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN\"\n"
        + "\"/http://www.w3.org/TR/html4/loose.dtd\"");
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Hola mundo</title>");
    out.println("<meta http-equiv=\"Content-Type\" content=\""
        + response.getContentType()+"\"");
    out.println("</head>");
    out.println("<body>");
    out.println("<h1>Hola mundo!</h1>");
    out.println("<p>Aleatorio: " + Math.random());
    out.print("<p><img src=\"./logoUMA.jpg\" alt=\"logo UMA\">");
    out.println("</body>");
    out.println("</html>");
    out.close();
}
```



# Servlet

## Respuesta dinámica

- El objeto `request` representa la petición
- Contiene (entre otros):
  - La URL que el usuario utilizó
    - `getRequestURL`, `getRequestURI`
  - Los parámetros de la petición
    - `getParameter`, `getParameterValues`
  - Las cabeceras de la petición
    - `getHeader`, `getHeaders`
  - Las cookies que envía el cliente
    - `getCookies`

Una Cookie es un par clave/valor que el navegador envía al servidor cada vez que realiza una petición

# Servlet

## Respuesta dinámica

- El objeto `response` representa la respuesta
- Permite establecer:
  - El contenido del cuerpo
    - `setContentType`, `getWriter`, `getOutputStream`
  - El estado de error y el mensaje
    - `sendError`
  - Las cabeceras de la respuesta
    - `addHeader`
  - Las cookies para futuros accesos
    - `addCookie`

# Servlet

# Respuesta dinámica

## ■ Ejemplo:

```
Map<String, String[]> params = request.getParameterMap();
out.println("<ul>");
for (Entry<String, String[]> e : params.entrySet())
{
    String k = e.getKey();
    for (String v : e.getValue())
    {
        out.println("<li>" + k + "=" + v + "</li>");
    }
}
out.println("</ul>");
```



# Servlet

## Procesar formularios

- Un uso típico de los servlets es procesar un formulario (almacenando información en una BBDD, por ejemplo)
- Vamos a crear un formulario dirigido a nuestro servlet anterior:

Nombre

Apellidos



# Java Server Pages (JSP)



# Java Server Pages

## Motivación

- El contenido estático de la respuesta en un servlet hay que incluirla como cadena de texto en Java
- Esto resulta problemático:
  - No es cómodo para un desarrollador
  - Es propenso a error (los compiladores no analizan el interior de las cadenas y mucho menos si son código (X)HTML válido)
  - Impide que podamos utilizar herramientas de ayuda de escritura de (X)HTML

# Java Server Pages

## Motivación

```
response.setContentType("text/html;charset=UTF-8");
PrintWriter out = response.getWriter();
out.println("<!DOCTYPE HTML "
    + "PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN\""
    + "\n"
    + "\n");
out.println("<html>");
out.println("<head>");
out.println("<title>Hola mundo</title>");
out.println("<meta http-equiv=\"Content-Type\" content=\""
    + response.getContentType()+"\"");
out.println("</head>");
out.println("<body>");
out.println("<h1>Hola mundo!</h1>");
out.println("<p>Aleatorio: " + Math.random());
out.print("<p><img src=\"./logoUMA.jpg\" alt=\"logo UMA\">");
out.println("</body>");
out.println("</html>");
out.close();
```

Respuesta

Servlet

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Hola mundo</title>
<meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
</head>
<body>
<h1>Hola mundo!</h1>
<p>Aleatorio: 0.9130131472639464
<p></body>
</html>
```

Contenido dinámico

# Java Server Pages

## Motivación

Respuesta

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<title>Hola mundo</title>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
</head>  
<body>  
<h1>Hola mundo!</h1>  
<p>Aleatorio: 0.9130131472639464  
<p></body>  
</html>
```

Contenido dinámico

JSP equiv.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>  
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"  
"http://www.w3.org/TR/html4/loose.dtd">  
<html>  
<head>  
<title>Hola mundo</title>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  
</head>  
<body>  
<h1>Hola mundo!</h1>  
<p>Aleatorio: <%= Math.random() %>  
<p></body>  
</html>
```

Directiva

Contenido dinámico

# Java Server Pages

## Motivación

- JSP permite combinar código de plantilla (típicamente HTML) con código Java
- Esta combinación permite escribir más cómodamente el contenido estático mediante código de plantilla) sin afectar a la capacidad para generar contenido dinámico
- En el fondo, JSP es convertido en un Servlet...

# Java Server Pages

## Motivación

Servlet generado por la página JSP anterior

```
response.setContentType("text/html;charset=UTF-8");
response.setHeader("X-Powered-By", "JSP/2.2");
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;
_jspx_resourceInjector = (org.glassfish.jsp.api.ResourceInjector) application.getAttribute("com.sun.appserv.jsp.resource.injector");

out.write("\n");
out.write("\n");
out.write("\n");
out.write("<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"\n");
out.write("      \"http://www.w3.org/TR/html4/loose.dtd\">\n");
out.write("<html>\n");
out.write("  <head>\n");
out.write("    <title>Hola mundo</title>\n");
out.write("    <meta http-equiv=\"Content-Type\" content=\"text/html; charset=UTF-8\">\n");
out.write("  </head>\n");
out.write("  <body>\n");
out.write("    <h1>Hola mundo!</h1>\n");
out.write("    <p>Aleatorio: ");
out.print( Math.random() );
out.write("\n");
out.write("    <p><img src=\"./logoUMA.jpg\" alt=\"logo UMA\"></body>\n");
out.write("</html>\n");
out.write("\n");
out.write("\n");
```

# Java Server Pages

## Estructura

- En una página JSP pueden aparecer 5 tipos de elementos
  - Elementos de *scripting*
  - Directivas
  - Expresiones EL (*Expression Language*)
  - Acciones
  - Código de plantilla
- Los comentarios van entre <%-- y --%>

# Java Server Pages

## Estructura: elementos de scripting

- Permiten añadir código Java
- Existen tres subtipos
  - Scriptlets
    - Encerrados entre <% y %>
    - Admiten cualquier código Java
    - Dicho código se copia directamente en el servlet generado
    - Muy flexible



```
<h1>Hola mundo!</h1>
<% for (int i=0; i < 10; i++) { %>
<p> Esto se repite 10 veces
<% } %>
```

# Java Server Pages

## Estructura: elementos de scripting

- Existen tres subtipos (cont.)
  - Expresiones
    - Encerrados entre <%= y %>
    - Admiten una expresión Java (sin terminar en punto y coma)
    - El resultado de evaluar dicha expresión se muestra en la respuesta (tras convertirlo a String)

```
<p>Aleatorio: <%= Math.random() %>
```

# Java Server Pages

## Estructura: elementos de scripting

- Existen tres subtipos (cont.)
  - Declaraciones
    - Encerradas entre <%!> y <%>
    - Admiten una declaración de variable o método
    - No generan resultado visual
    - La variable o método formarán parte del servlet generado



```
<%! static int factorial (int i) {  
    int res = 1;  
    while (i > 1) res *= i--;  
    return res;  
} %>
```

```
<p>Factorial 7: <%= factorial(7) %>
```

# Java Server Pages

## Estructura: directivas

- Especifican información sobre la página
- Están encerradas entre <%@ y %>
- Existen tres
  - page
    - Indica el tipo MIME de la respuesta
    - La codificación del conjunto de caracteres
    - Los “import” de Java

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
```

# Java Server Pages

## Estructura: directivas

- Existen tres (cont.)
  - include
    - Permite incluir otro fichero en su posición
    - Buena forma de incluir elementos que deban aparecer en todas las páginas: cabecera, logos, etc.
  - taglib
    - Declara el uso de una biblioteca de acciones

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

```
<%@taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"%>
```

# Java Server Pages

## Estructura: expresiones EL

- Inicialmente se introdujeron con la biblioteca JSTL (*JSP Standard Tag Library*)
- Son de la forma  ***\${expresion}***

***\${param.val}***

- Pueden aparecer en el código (X)HTML o en algunos atributos de acciones
- Siguen las convenciones de JavaBeans
  - ***\${persona.nombre}*** devuelve el resultado de llamar al método `getNombre()` de la var. persona

# Java Server Pages

## Estructura: expresiones EL

- Existen objetos predefinidos para usar con expresiones EL:
  - pageContext header
  - pageScope headerValues
  - requestScope cookie
  - sessionScope initParam
  - applicationScope
  - param
  - paramValues

# Java Server Pages

## Estructura: acciones

- Son etiquetas que realizan alguna función
- Constituyen una forma de modularización de elementos JSP

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

- Los nombres de las etiquetas suelen estar formado por un prefijo, dos puntos (:) y el nombre de la acción
- Las acciones con prefijo `jsp` son estándar de JSP

# Java Server Pages

## Estructura: acciones

- Algunas etiquetas estándar son:
- A estas etiquetas hay que añadir las de la biblioteca JSTL, que ya forma parte de Java EE

### JSTL Tag Libraries

Functional Area	URI	Prefix
core	<a href="http://java.sun.com/jsp/jstl/core">http://java.sun.com/jsp/jstl/core</a>	c
XML processing	<a href="http://java.sun.com/jsp/jstl/xml">http://java.sun.com/jsp/jstl/xml</a>	x
I18N capable formatting	<a href="http://java.sun.com/jsp/jstl/fmt">http://java.sun.com/jsp/jstl/fmt</a>	fmt
relational db access (SQL)	<a href="http://java.sun.com/jsp/jstl/sql">http://java.sun.com/jsp/jstl/sql</a>	sql
Functions	<a href="http://java.sun.com/jsp/jstl/functions">http://java.sun.com/jsp/jstl/functions</a>	fn

```
<jsp:useBean> ...
<jsp:setProperty>
<jsp:getProperty>
<jsp:include> ...
<jsp:forward> ...
<jsp:param> ....
<jsp:plugin> ....
<jsp:params> ...
<jsp:fallback> ...
<jsp:attribute> ...
<jsp:body> .....
<jsp:invoke> ...
<jsp:doBody> ...
<jsp:element> ...
<jsp:text> .....
<jsp:output> ....
```

# Java Server Pages

## Estructura: acciones

- Ejemplo:

```
<c:catch var="error">
<p><fmt:formatNumber value="${param.numero}" pattern="#,.##"/>
</c:catch>

<c:if test="${!empty error}">
<p>Debe escribir un número
</c:if>
```



# Java Server Pages

## Estructura: acciones

### Core

```
<c:catch>
<c:choose>
<c:forEach>
<c:forTokens>
<c:if>
<c:import>
<c:otherwise>
<c:out>
<c:param>
<c:redirect>
<c:remove/>
<c:set>
<c:url>
<c:when>
```

### I18N

```
<fmt:bundle>
<fmt:formatDate/>
<fmt:formatNumber>
<fmt:message>
<fmt:param>
<fmt:parseDate>
<fmt:parseNumber>
<fmt:requestEncoding/>
<fmt:setBundle/>
<fmt:setLocale/>
<fmt:setTimeZone/>
<fmt:timeZone>
```

### XML

```
<x:choose>
<x:forEach>
<x:if>
<x:otherwise>
<x:out/>
<x:param>
<x:parse>
<x:set/>
<x:transform>
<x:when>
```

### SQL

```
<sql:dateParam/>
<sql:param>
<sql:query>
<sql:setDataSource/>
<sql:transaction>
<sql:update>
```

# Java Server Pages

## Estructura: acciones

- fn
- Solo EL
- Ej:

```
 ${fn:length('hola')}
```

boolean	<a href="#">contains( java.lang.String, java.lang.String)</a>
boolean	<a href="#">containsIgnoreCase( java.lang.String, java.lang.String)</a>
boolean	<a href="#">endsWith( java.lang.String, java.lang.String)</a>
java.lang.String	<a href="#">escapeXml( java.lang.String)</a>
int	<a href="#">indexOf( java.lang.String, java.lang.String)</a>
java.lang.String	<a href="#">join( java.lang.String[], java.lang.String)</a>
int	<a href="#">length( java.lang.Object)</a>
java.lang.String	<a href="#">replace( java.lang.String, java.lang.String, java.lang.String)</a>
java.lang.String[]	<a href="#">split( java.lang.String, java.lang.String)</a>
boolean	<a href="#">startsWith( java.lang.String, java.lang.String)</a>
java.lang.String	<a href="#">substring( java.lang.String, int, int)</a>
java.lang.String	<a href="#">substringAfter( java.lang.String, java.lang.String)</a>
java.lang.String	<a href="#">substringBefore( java.lang.String, java.lang.String)</a>
java.lang.String	<a href="#">toLowerCase( java.lang.String)</a>
java.lang.String	<a href="#">toUpperCase( java.lang.String)</a>
java.lang.String	<a href="#">trim( java.lang.String)</a>

# Java Server Pages

## Estructura: código de plantilla

- Todo lo que no sea nada de lo anterior es código de plantilla
- Es la parte estática de la página
- Típicamente código (X)HTML
- Pero puede ser cualquier otro formato de texto
- Por ejemplo: una imagen **SVG**



# Para ampliar conocimientos

- Especificación Servlets 3.1 (JSR 340)  
<https://jcp.org/en/jsr/detail?id=340>
- Especificación de JSP 2.3 (JSR 245):  
<https://jcp.org/en/jsr/detail?id=245>
- Especificación de JSTL 1.2 (JSR 52):  
<https://jcp.org/en/jsr/detail?id=52>
- Giuio Zambon, “Beginning JSP, JSF and Tomcat”, Apress (cap. 2 y 4)
- Abraham Otero, “Tutorial básico de Java EE” (disponible en campus virtual)
- Oracle Java EE 7 Tutorial (caps. 6, 9, 17)

