



Sistemas Distribuidos e Internet

Tema 1
Introducción a JEE

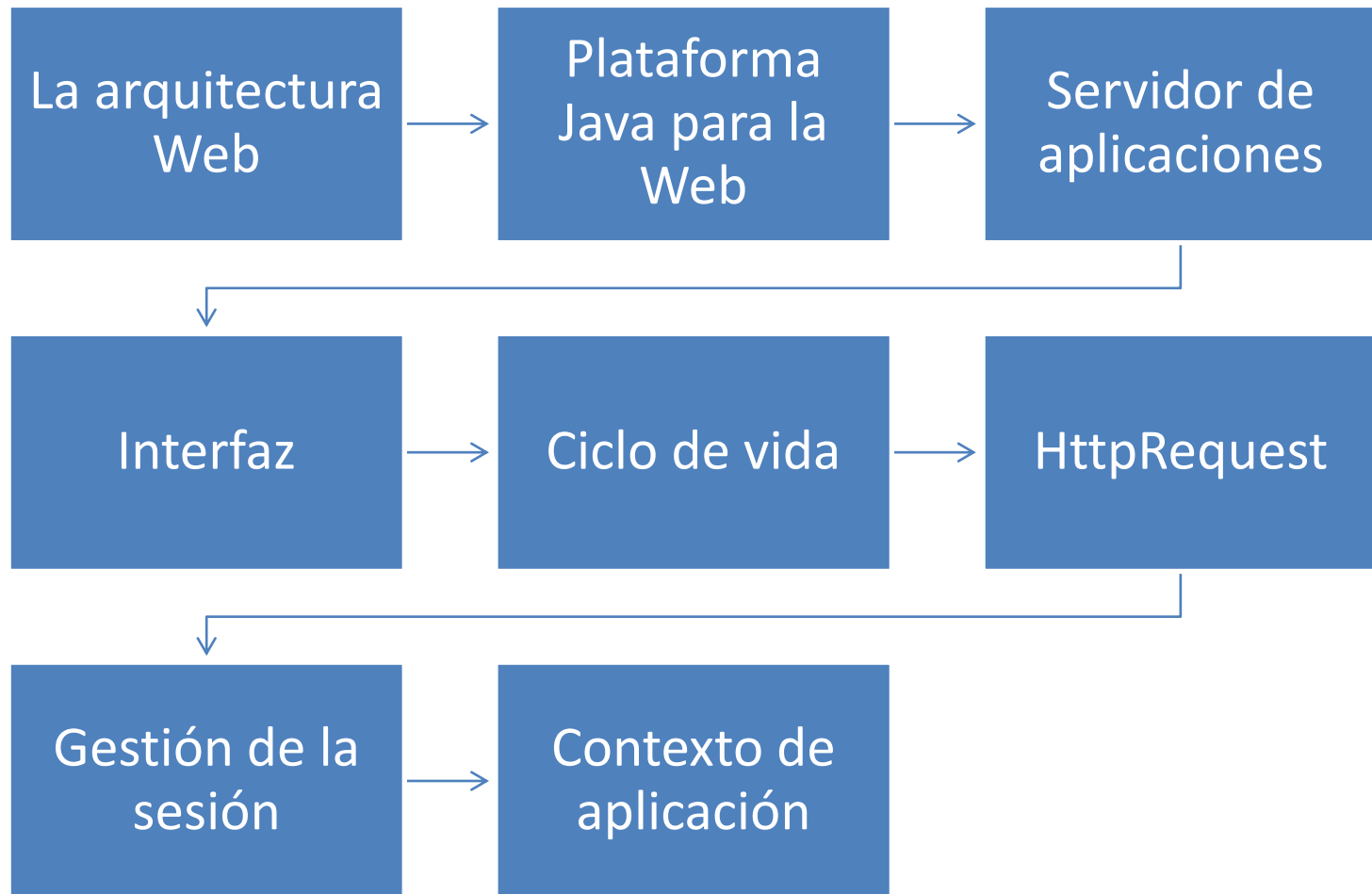


Índice General

- Servlets
- JSPs
- Encadenamiento de Servlets y JSPs



Contenidos



La arquitectura Web : Un formulario

```
<html lang="en">
<head>
  <title>Servlets</title>
  <meta charset="utf-8"/>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</head>

<body>

  <!-- Contenido -->
  <div class="container" id="contenedor-principal">
    <h2>Formulario GET Saludar</h2>

    <form action="GreetingServlet" method="get">
      <div class="form-group">
        <label for="name-get">Nombre</label>
        <input type="text" class="form-control" name="name" id="name-get">
      </div>
      <button type="submit" class="btn btn-primary">Submit</button>
    </form>

    <h2>Formulario POST</h2>

    <form action="GreetingServlet" method="post">
      <div class="form-group">
        <label for="name-post">Nombre</label>
        <input type="text" class="form-control" name="name" id="name-post">
      </div>
      <button type="submit" class="btn btn-primary">Submit</button>
    </form>
  </div>
</body>
</html>
```

```
package com.uniovi.sdi;

import ...

@WebServlet(name = "GreetingServlet", value = "/GreetingServlet")
public class GreetingServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    int contador = 0;

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HTML>");
        out.println("<HEAD><TITLE>Hello World!</TITLE></HEAD>");
        out.println("<BODY>");
        String name = request.getParameter("name");
        if (name != null) {
            out.println("Hello " + name + "<br>");
        }
        out.println("</BODY></HTML>");

        try {
            Thread.sleep(15000);
        } catch (InterruptedException e) {}

        out.println("Thread ID: " + Thread.currentThread().getId() + "<br>");
        contador++;
        out.println("Visits: " + contador + "<br>");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }
}
```

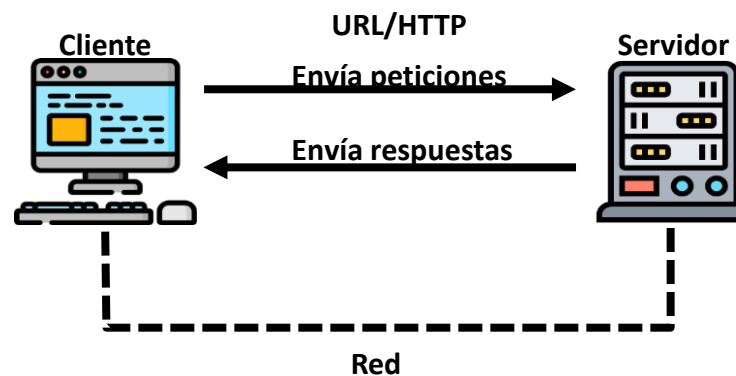
Action = "GreetingServlet"

La arquitectura Web básica

- Está basado en el **Modelo Cliente/Servidor**.

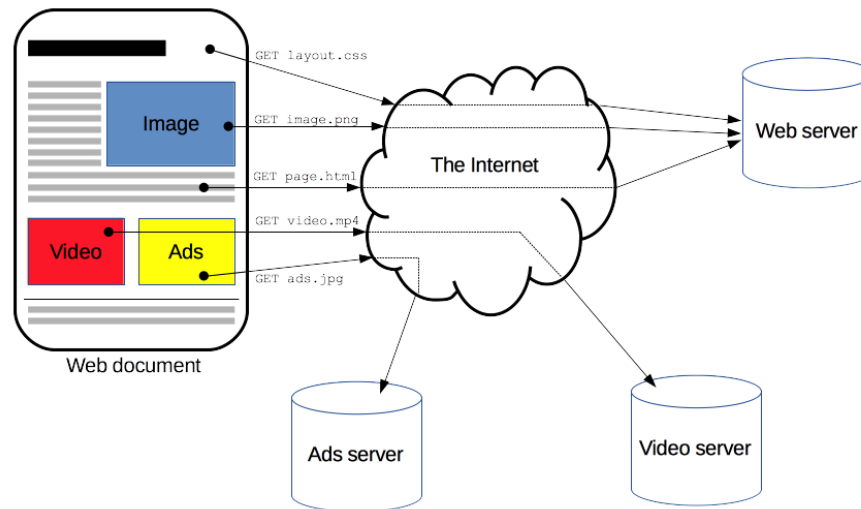
*Un cliente (**Navegador**) envía una petición de un recurso (**URL**) a un servidor (**Servidor HTTP**) y éste le responde vía **HTTP**.*

- **URL (Unique Resource Locator):** Nomenclatura para un recurso web ([Enlace](#))
- **HTTP (HyperText Transfer Protocol)** es el **protocolo de aplicación** utilizado para el intercambio de información en la Web mediante el **paso de mensajes (Cabeceras HTTP)**.



La arquitectura Web

- Cuando el navegador solicita una página web, **recibe la página Y:**
 - Desencadena **una petición para cada uno de los recursos asociados a la misma.**



Fuente: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>

Plataformas Java

Java SE (Java Platform, Standard Edition)

- Para aplicaciones y applets, núcleo de la especificación de Java 2, máquina virtual, herramientas y tecnologías de desarrollo, librerías, ...

Java EE (Java Platform, Enterprise Edition)

- Se apoya en Java SE; con el paso del tiempo, algunas APIs de Java EE se pasaron (y quizás se sigan pasando) a Java SE
- Incluye las especificaciones para **Servlets, JSP, JSF, Beans, ...**: (<http://www.oracle.com/technetwork/java/javaee/tech/index.html>)

Java ME (Java Platform, Micro Edition)

- Subconjunto de Java SE para pequeños dispositivos (móviles, PDAs, ...)

JavaFX (JavaFX Script)

- API para aplicaciones cliente. Permite incluir gráficos, contenidos multimedia, embeber páginas web, controles visuales, ...

JDBC (Java SE)

- API para acceso a bases de datos relacionales
- El programador puede lanzar queries (consulta, actualización, inserción y borrado), agrupar queries en transacciones, ...

Introducción: JEE

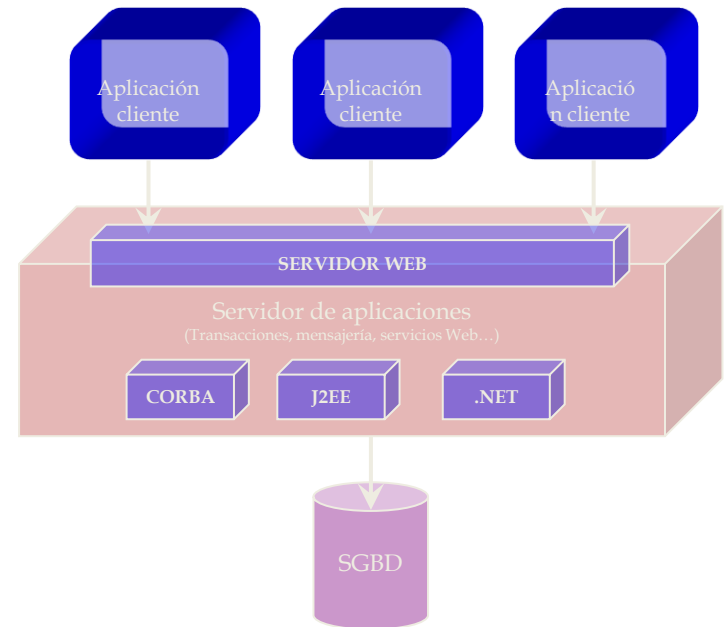
Definición: Java EE = Java Enterprise Edition

- Especificación de Sun para una plataforma basada en APIs de Java (Java SE) que permiten construir aplicaciones empresariales.
- Las especificaciones suelen ser interfaces y clases abstractas.
- Existen múltiples implementaciones de diversos fabricantes incluso OpenSource: IBM WebSphere, Oracle Glasfish, OraclexAS: [Oracle](#)
- Una aplicación Java EE no depende de una implementación particular
- Sitio central de JEE: [Oracle](#)



Servidores de aplicaciones

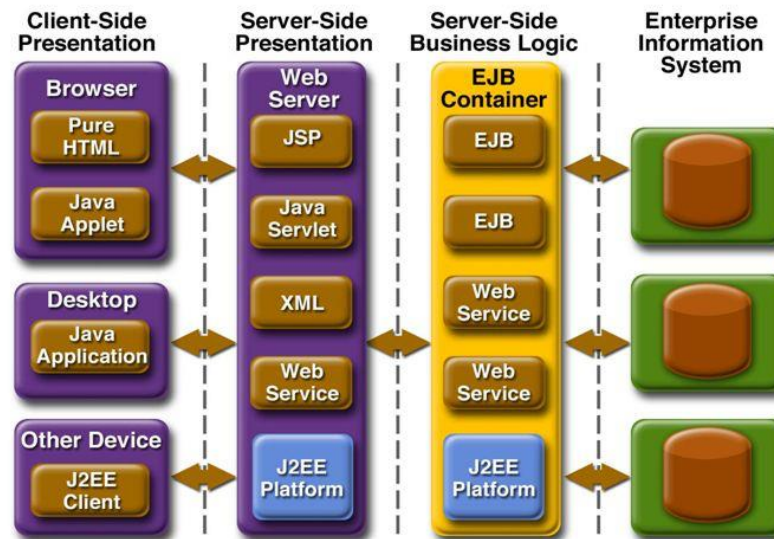
- Programa que provee la infraestructura necesaria para aplicaciones web empresariales
 - Los programadores van a poder dedicarse casi en exclusiva a implementar la lógica del dominio
 - Servicios como seguridad, persistencia, transacciones, etc. son proporcionados por el propio servidor de aplicaciones
 - Pieza clave para cualquier empresa de comercio electrónico
- Es una capa intermedia (*middleware*) que se sitúa entre el servidor web y las aplicaciones y bases de datos subyacentes



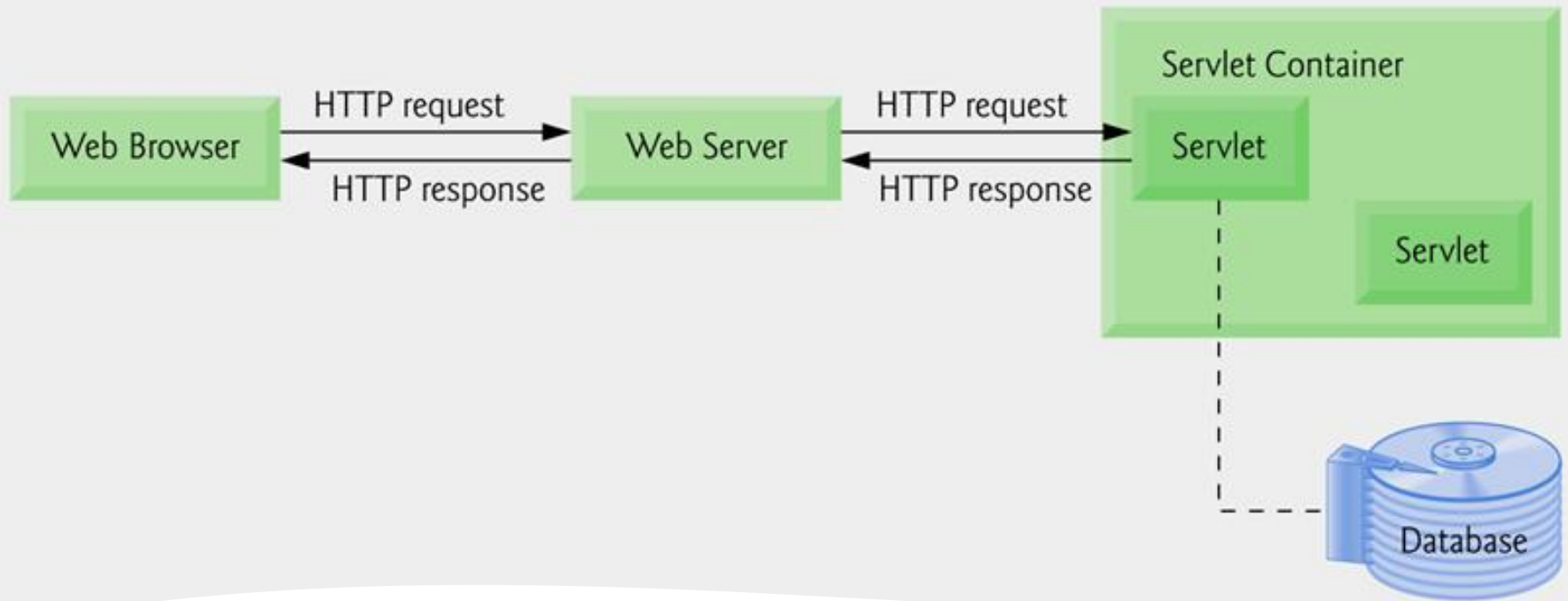
Servidores de aplicaciones

- Tecnología JEE

Arquitectura J2EE



Introducción a la arquitectura J2EE con ejemplos prácticos

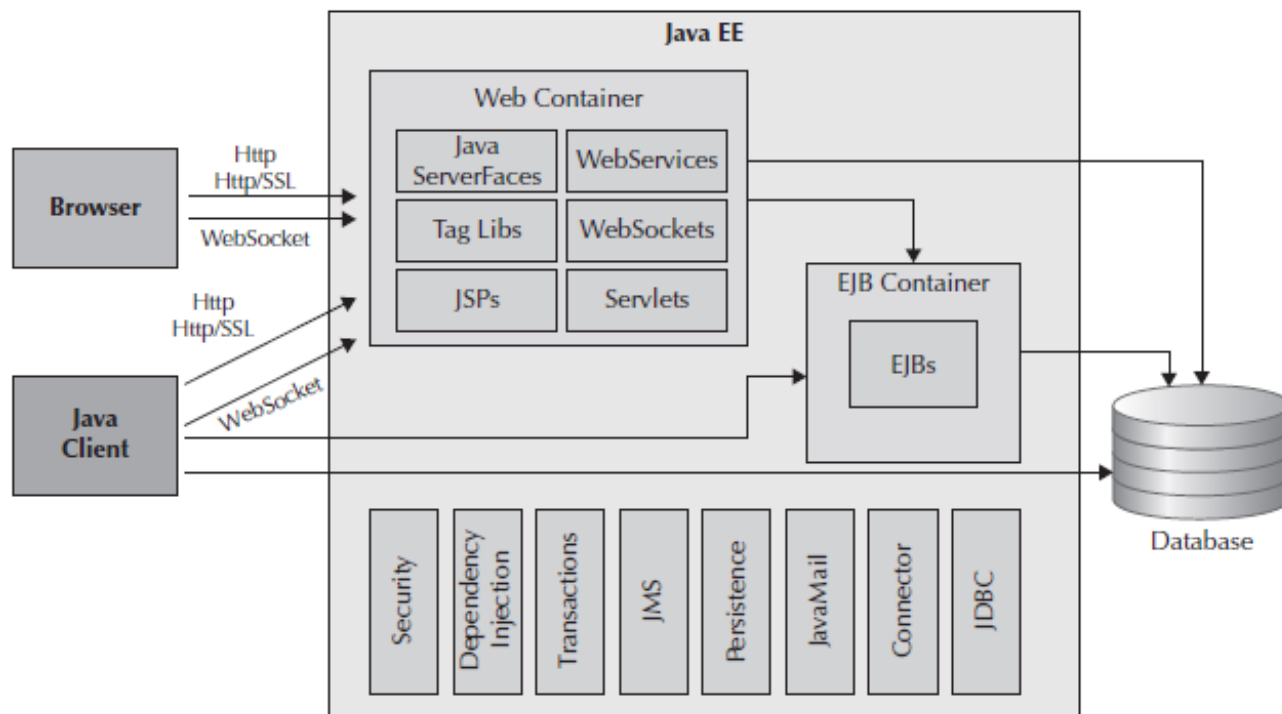


Qué es un servlet

- Es una clase Java que hereda de la clase JEE HTTPServlet y que:
 - Acepta peticiones de cualquier método HTTP (get, post, put, delete, head, trace, ...)
 - Responde también usando el protocolo HTTP
 - Se ejecuta dentro de un **contenedor de Servlets** que a su vez está dentro de un **servidor de aplicaciones JEE**

Contenedor de servlets/Web container

- Un contenedor define un ambiente estandarizado de ejecución que provee servicios específicos a los servlets
 - Por ejemplo, dan servicio a las peticiones de los clientes, realizando un procesamiento y devolviendo el resultado
- Los servlets tienen que cumplir un contrato con el contenedor para obtener sus servicios
 - Los contratos son interfaces Java
 - Por ejemplo, la interfaz *Servlet*



Ejemplo Servlet “HolaMundo.java”

```
import java.io.*;           // Necesario importar estos tres paquetes
import javax.servlet.*;
import javax.servlet.http.*;
```

```
public class HolaMundo extends HttpServlet { // Herencia de HttpServlet
```

```
    // sustituir métodos doGet() y/o doPost().
```

```
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws IOException, ServletException {
        // Incluir excepciones generadas por doGet() y doPost()
```

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

```
        out.println("<html>"); out.println("<head>");
        out.println("<title> Servlet Hola Mundo </title>");
        out.println("</head>");
```

```
        out.println("<body>");
        out.println("<h1> Hola Mundo!</h1>");
        out.println("</body>");
        out.println("</html>");
```

```
    }
}
```

z HttpServletRequest (Datos de entrada)

- Variables de Formulario
- Cabeceras de solicitud HTTP (CGI)
- Datos del servidor (CGI)

z HttpServletResponse (Datos de salida)

- Códigos de estado
- Encabezados de respuesta: Content-type, Set-Cookie...
- Obtención de un objeto PrintWriter para la respuesta

Servlet API

I

Servlet

- `init(ServletConfig)`
- `getServletConfig()`
- `service(ServletRequest, ServletResponse)`
- `getServletInfo()`
- `destroy()`

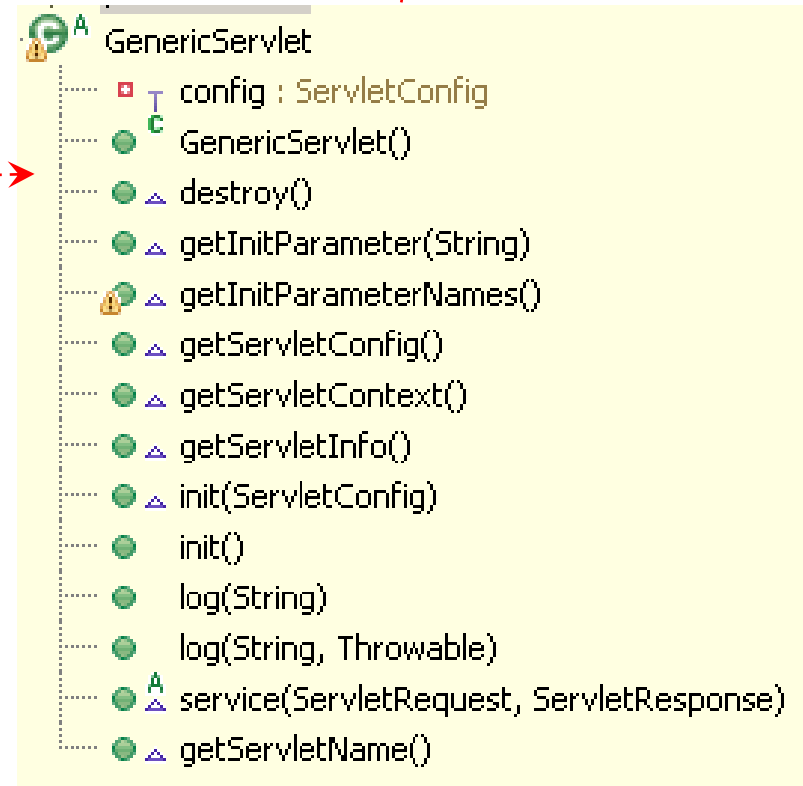
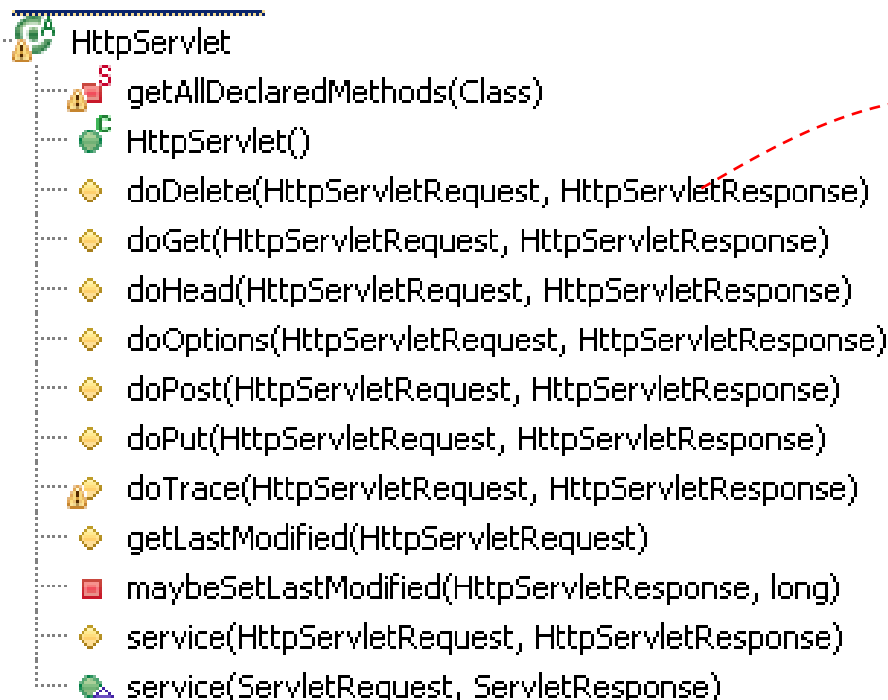
```
import java.io.IOException;
import javax.servlet.*;

public class MyServlet extends GenericServlet {

    public void service (
        ServletRequest request,
        ServletResponse response)
        throws ServletException, IOException {
        //...
    }
    //...
}
```

SDI - Introducción a JEE

Métodos y herencia en Servlet



Servlets: Ciclo de vida

INICIALIZACIÓN:

- Una única llamada al metodo “init” por parte del contenedor de servlets

public void **init**([ServletConfig](#) config) throws [ServletException](#)

- Se pueden recoger unos parametros concretos con “getInitParameter” de “ServletConfig”. Estos parámetros se especifican en el descriptor de despliegue de la aplicación: **web.xml**

PETICIONES

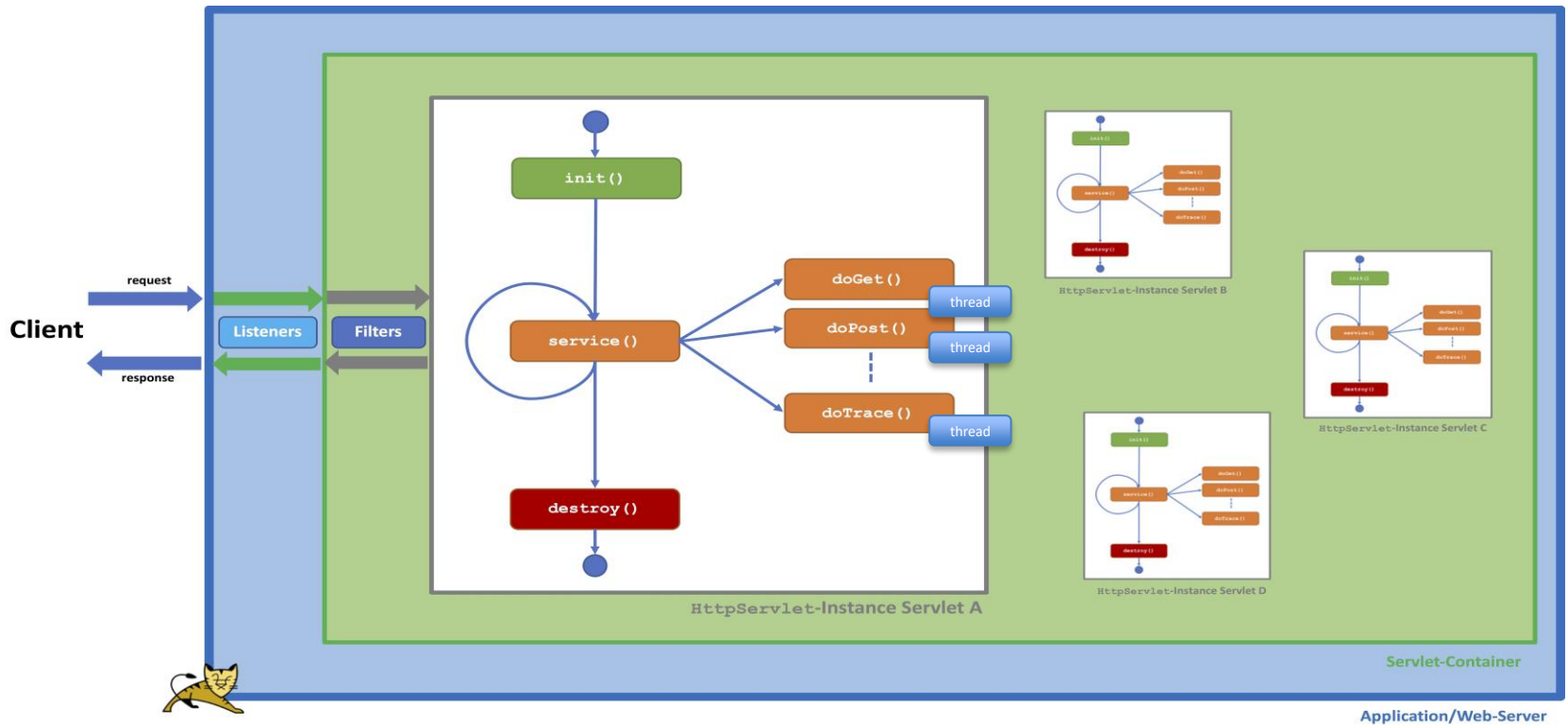
- La primera petición a init se ejecuta en un thread que invoca a service
- El resto de peticiones se invocan en un nuevo hilo mapeado sobre service

DESTRUCCIÓN:

- Cuando todas las llamadas desde el cliente cesen o un temporizador del servidor así lo indique. Se deben liberar recursos retenidos desde init()

public void **destroy**()

Servlet Ciclo de vida



Preguntas Vevox

- Crea una cuenta en Vevox: <http://vevox.com>
- Puedes acceder sin crear cuenta:
<https://vevox.app/#/>
- Acceder a la sesión de la asignatura: xxx-xxx-xx (El profesor te suministrará la ID de sesión)

Servlets: Políticas de acceso concurrente (threading)

- Los servlets están diseñados para soportar múltiples accesos simultáneos por defecto
- **¡Ojo!** El problema puede surgir cuando se hace uso de un **recurso compartido**:
 - Ejemplo: abrimos un fichero desde un servlet
 - Solución:
 - Hacer que el recurso sea el que defina la política de acceso concurrente
 - Ejemplo: las bases de datos están preparadas para ello

Tipos de peticiones HTTP:

(Un navegador puede enviar la información al servidor de varias formas)



GET: Paso de parámetros en la propia URL de acceso al servicio o recurso del servidor. Método “doGet” del servlet



POST: Lo mismo que GET pero los parámetros no van en la línea de URL sino en otra línea a parte. El manejo es idéntico. Método “doPost” del servlet.



PUT, ...

```

protected void service(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

    String method = req.getMethod();

    if (method.equals(METHOD_GET)) {
        doGet(req, resp);

    } else if (method.equals(METHOD_HEAD)) {
        doHead(req, resp);

    } else if (method.equals(METHOD_POST)) {
        doPost(req, resp);

    } else if (method.equals(METHOD_PUT)) {
        doPut(req, resp);

    } else if (method.equals(METHOD_DELETE)) {
        doDelete(req, resp);

    } else if (method.equals(METHOD_OPTIONS)) {
        doOptions(req, resp);

    } else if (method.equals(METHOD_TRACE)) {
        doTrace(req, resp);
    }
}

```

Método service() en HttpServlet

- Separa la petición en función del método HTTP

Servlets: Métodos doGet y doPost

- Son llamados desde el método `service()`
- Reciben interfaces instanciadas:
 - “`HttpServletRequest`” canal de entrada con información enviada por el usuario
 - “`HttpServletResponse`” canal de salida (contenido web)

```
protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, java.io.IOException {
    . . .
}

protected void doPost(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, java.io.IOException {
    . . .
}
```

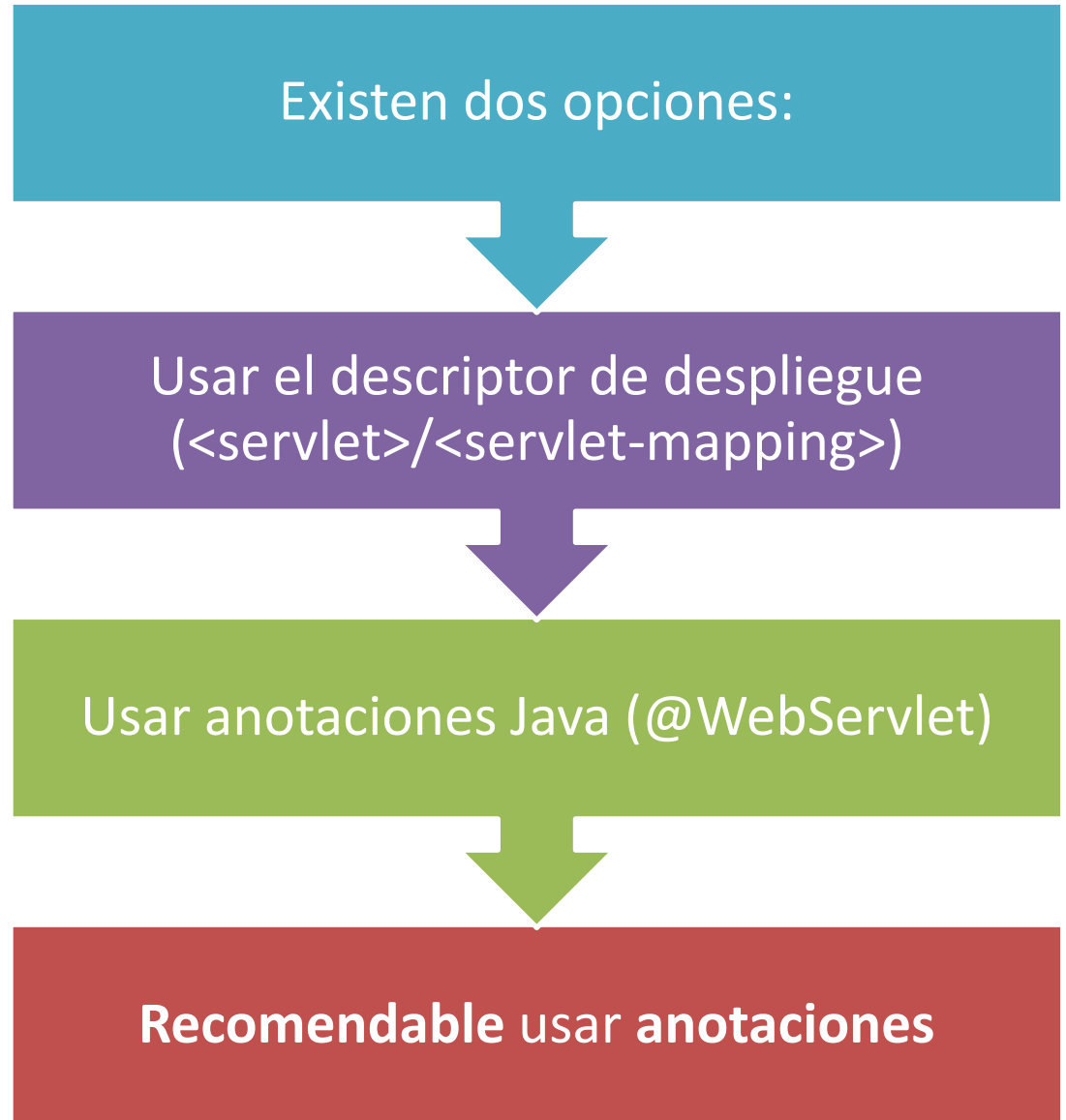
`doGet()` → consulta (no cambia estado, idempotente)

`doPost()` → modifica datos (cambia estado)

Ejemplo de Servlet HolaMundo

```
public class HelloWorld extends HttpServlet {  
  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
  
        response.setContentType("text/html");  
  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>");  
        out.println("<HEAD><TITLE>Hola Mundo!</TITLE></HEAD>");  
        out.println("<BODY>");  
        out.println("Bienvenido a mi primera página Güev!");  
        out.println("</BODY></HTML>");  
    }  
  
    protected void doPost(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
  
        doGet(request, response);  
    }  
}
```


Registro de un Servlet



Opción1: Registro de un Servlet en el **descriptor de despliegue web.xml**

- Insertamos en **web.xml** la declaración del servlet y del servlet-mapping

```
<servlet>
```

```
  <servlet-name>HolaMundo</servlet-name>
```

```
  <servlet-class>uo.sdi.servlet.HolaMundoServlet</servlet-class>
```

```
</servlet>
```

```
<!-- Standard Action Servlet Mapping -->
```

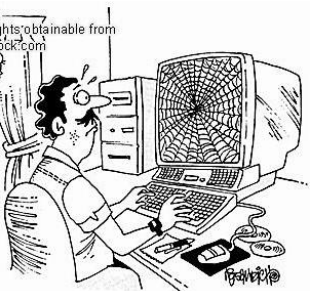
```
<servlet-mapping>
```

```
  <servlet-name>HolaMundo</servlet-name>
```

```
  <url-pattern>/HolaMundoCordial</url-pattern>
```

```
</servlet-mapping>
```

<http://<server>/HolaMundoCordial>



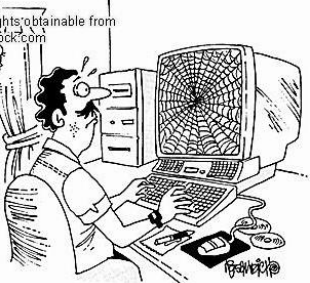
Opción2: Registro de un Servlet usando anotaciones (@WebServlet)

- Insertamos antes de la clase Servlet la anotación con el nombre del servlet y el url de mapeo.

```
@WebServlet(name = "HolaMundo", urlPatterns = { "/HolaMundoCordial" })  
public class HolaMundoServlet extends HttpServlet {  
    private static final long serialVersionUID = 1L;
```

```
/**  
 * @see HttpServlet#HttpServlet()  
 */  
public HolaMundoServlet() {  
    super();  
    // TODO Auto-generated constructor stub  
}
```

.....



<http://<server>/HolaMundoCordial>

HttpServletRequest: Recogiendo información de usuario

- Los parámetros nos llegan en la **request**
 - Request es tipo “*HttpServletRequest*”
 - Método *getParameter(<nombre>)*
- Son los parámetros de la QueryString o de los campos del formulario
- Siempre nos devuelve objetos de tipo **String**
 - Habrá que hacer las conversiones que proceda

Object HttpServletRequest.getParameter(nombre)
devuelve:

- "" (si no hay valor)
- `null` (si no existe el parámetro)
- El valor en caso de haber sido establecido

Formularios HTML y request.getParameter()

```
<form ACTION="http://server/app/HelloWorld" METHOD="POST">  
  Nombre: <INPUT TYPE="TEXT" NAME="NombreUsuario" SIZE="15">  
  <INPUT TYPE="Submit" VALUE="Aceptar">  
</FORM>
```

```
public class HelloWorld extends HttpServlet {  
    protected void doGet(HttpServletRequest request,  
        HttpServletResponse response) throws ServletException, IOException {  
        String nombre = (String) request.getParameter("NombreUsuario");  
  
        response.setContentType("text/html");  
  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>");  
        out.println("<HEAD><TITLE>Hola Mundo!</TITLE></HEAD>");  
        out.println("<BODY>");  
        out.println("Bienvenido " + nombre);  
        out.println("</BODY></HTML>");  
    }  
}
```

Ejemplo Servlet con parámetros:

HolaMundo personalizado

- Creamos index.html, página con un formulario que nos pasa el parámetro **“Nombre”**:

```
<HTML><HEAD> <TITLE>Mi primer formulario</TITLE> </HEAD>
<BODY>
  <FORM ACTION="http://127.0.0.1:8080/sdi/HolaMundoCordial"
    METHOD="POST">
    <CENTER><H1>Saludador</H1></CENTER>
    <HR><BR>
    <TABLE ALIGN="CENTER">
      <TR>
        <TD ALIGN="RIGHT">¿Cacute;mo te llamas?</TD>
        <TD><INPUT TYPE="TEXT" NAME="NombreUsuario"
          ALIGN="LEFT" SIZE="15"></TD>
      </TR>
      <TR>
        <TD><INPUT TYPE="Submit" VALUE="Saluda!"> </TD>
      </TR>
    </TABLE>
  </FORM>
</BODY>
</HTML>
```

Ejemplo Servlet con parámetros:

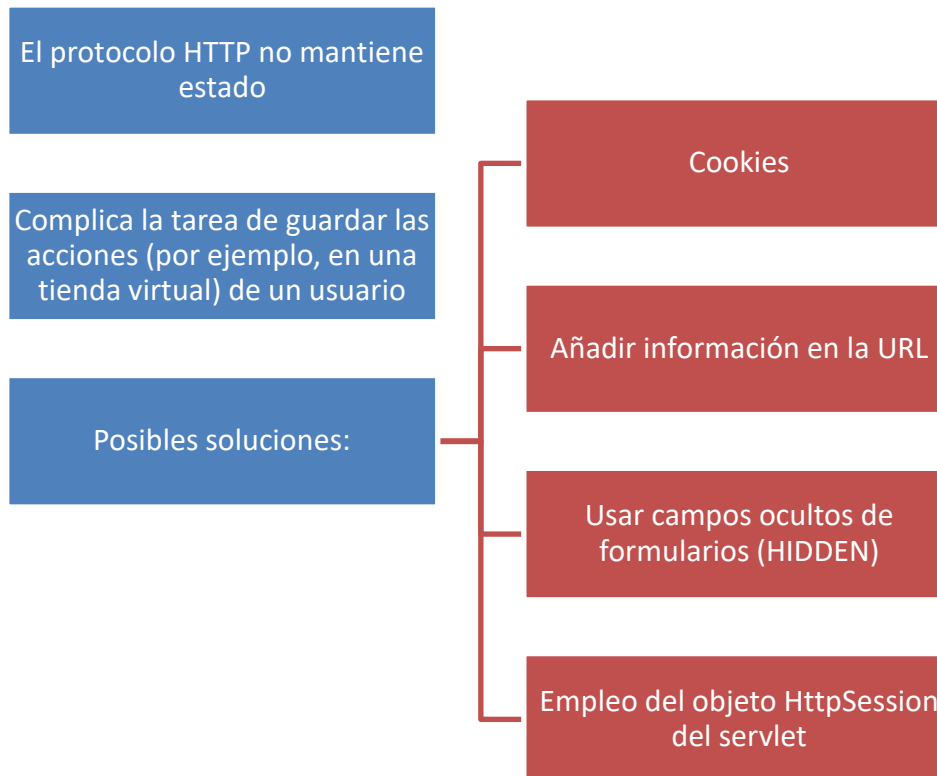
HolaMundo personalizado

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class HolaMundoServlet extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse
        res) throws ServletException, IOException {
        ...
        String nombre = (String) req.getParameter("NombreUsuario");
        ...
        if ( nombre != null )
            out.println("<br>Hola " + nombre + "<br>");
        ...
    }
}
```

Gestión de la Sesión. Mantenimiento del estado de la sesión.



Servlets: Seguimiento de sesión

- Los servlets proporcionan una solución técnica
 - La API **HttpSession**
- Una interfaz de alto nivel construida sobre las cookies y la reescritura de URLs
 - Pero transparente para el desarrollador
- Permite almacenar objetos

Servlets: Seguimiento de sesión

- Para buscar el objeto HttpSession asociado a una petición HTTP:
 - Se usa el método “getSession()” de “HttpServletRequest” que devuelve null si no hay una sesión asociada
 - En este último caso podríamos crear un objeto HttpSession
 - Pero, al ser una tarea sumamente común, se puede pasar al método un argumento con valor true y él mismo se encarga de crear un objeto HttpSession si no existe

Interfaz HttpSession

HttpSession	
	<code>getCreationTime()</code>
	<code>getId()</code>
	<code>getLastAccessedTime()</code>
	<code>getServletContext()</code>
	<code>setMaxInactiveInterval(int)</code>
	<code>getMaxInactiveInterval()</code>
	<code>getSessionContext()</code>
	<code>getAttribute(String)</code>
	<code>getValue(String)</code>
	<code>getAttributeNames()</code>
	<code>getValueNames()</code>
	<code>setAttribute(String, Object)</code>
	<code>putValue(String, Object)</code>
	<code>removeAttribute(String)</code>
	<code>removeValue(String)</code>
	<code>invalidate()</code>
	<code>isNew()</code>



Deprecated methods

Servlets: Seguimiento de sesión

- Añadir y recuperar información de una sesión
 - **getAttribute("nombre_variable")**
 - Devuelve una instancia de `Object` en caso de que la sesión ya tenga *algo* asociado a la etiqueta **nombre_variable**
 - `null` en caso de que no se haya asociado nada aún
 - **setAttribute("nombre_variable", referencia)**
 - Coloca el objeto referenciado por **referencia** en la sesión del usuario bajo el nombre **nombre_variable**. A partir de este momento, el objeto puede ser recuperado por este mismo usuario en sucesivas peticiones. Si el objeto ya existiera, **lo sobrescribe**
 - **getAttributeNames()**
 - Retorna una *Enumeration* con los nombres de todos los atributos establecidos en la sesión del usuario

Servlets: Seguimiento de sesión

- **getId()**
 - Devuelve un identificador único generado para cada sesión
- **isNew()**
 - True si el cliente (navegador) nunca ha visto la sesión. False para sesión preexistente
- **getCreationTime()**
 - Devuelve la hora, en milisegundos desde 1970, en la que se creó la sesión
- **getLastAccessedTime()**
 - La hora en que la sesión fue enviada por última vez al cliente

Servlets: Seguimiento de sesión

- Caducidad de la sesión
 - Peculiaridad de las aplicaciones web
 - **No sabemos cuándo se desconecta el usuario del servidor**
 - Automáticamente el servidor web invalida la sesión tras un periodo de tiempo (p.e., 30') sin peticiones o manualmente usando el método `invalidate`

¡OJO!

¡SOBRECARGAR LA SESIÓN ES PELIGROSO!

Los elementos almacenados no se liberan hasta que no salta el timeout o `session.invalidate()`

Servlets: Contexto de la aplicación

- Se trata de un saco “**común**” a todas las sesiones de usuario activas en el servidor
- Nos permite compartir información y objetos entre los distintos usuarios
- Se accede por medio del objeto “`ServletContext`”

```
public ServletContext getServletContext()
```

(Ejemplo: Contador de Visitas)



Se hereda de `GenericServlet`

Interfaz ServletContext()

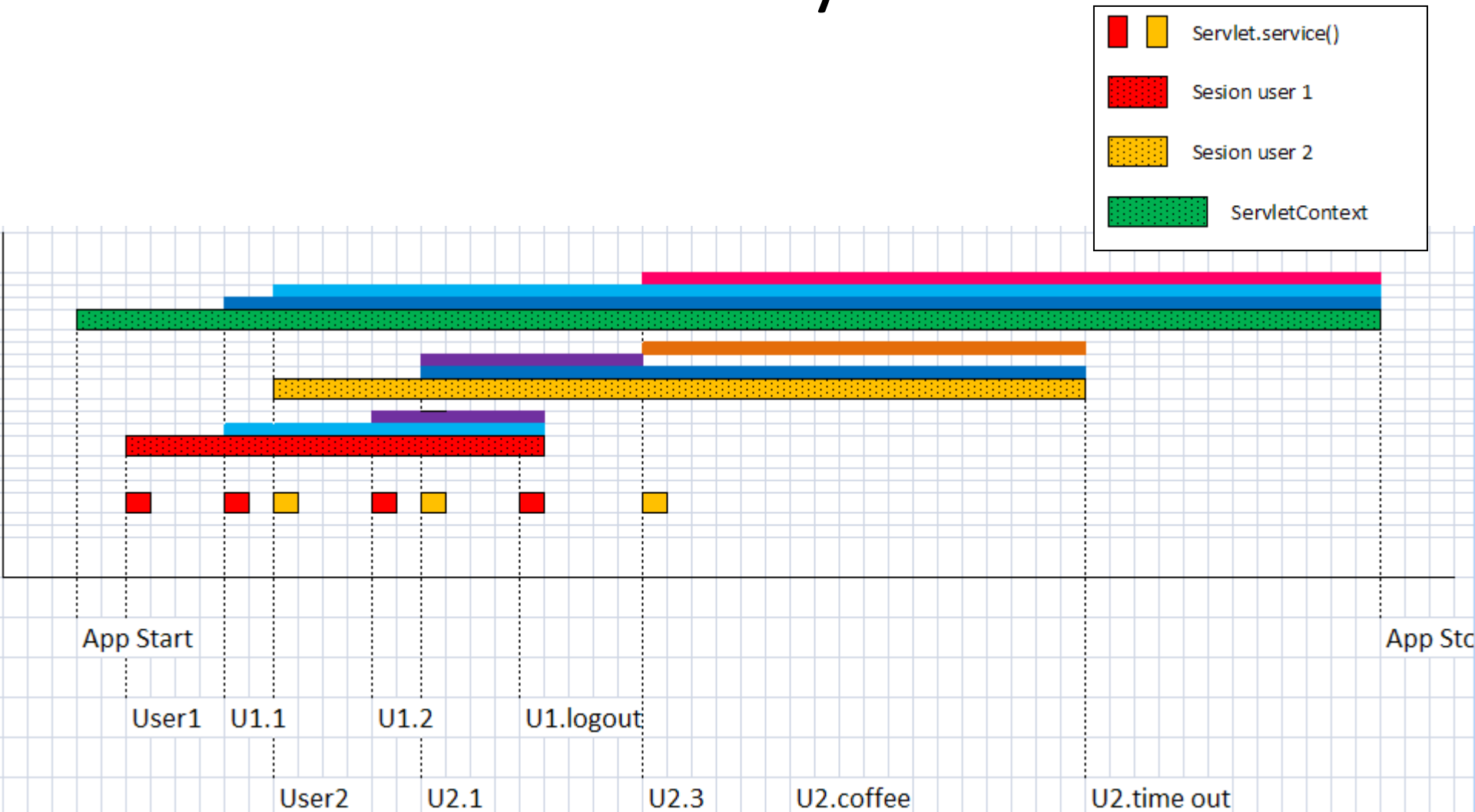
ServletContext	
●	getContext(String)
●	getContextPath()
●	getMajorVersion()
●	getMinorVersion()
●	getMimeType(String)
●	getResourcePaths(String)
●	getResource(String)
●	getResourceAsStream(String)
●	getRequestDispatcher(String)
●	getNamedDispatcher(String)
●	getServlet(String)
●	getServlets()
●	getServletNames()
●	log(String)
●	log(Exception, String)
●	log(String, Throwable)
●	getRealPath(String)
●	getServerInfo()
●	getInitParameter(String)
●	getInitParameterNames()
●	getAttribute(String)
●	getAttributeNames()
●	setAttribute(String, Object)
●	removeAttribute(String)
●	getServletContextName()

Servlets: Contexto de la aplicación

- Para colocar o recuperar objetos del contexto...
 - **Añadir un atributo**
 - Se usa el método “`setAttribute`” de “`ServletContext`”
 - El control sobre varios servlets manipulando un mismo atributo es responsabilidad del desarrollador
 - **Recuperar un atributo**
 - Se usa el método “`getAttribute`” de “`ServletContext`”
 - Hay que convertir el objeto que devuelve al tipo requerido (retorna un `Object`)

Ejemplo: Contador de Visitas

Duración de Session y ServletContext



Preguntas Vevox

- Crea una cuenta en Vevox: <http://vevox.com>
- Puedes acceder sin crear cuenta:
<https://vevox.app/#/>
- Acceder a la sesión de la asignatura: xxx-xxx-xx (El profesor te suministrará la ID de sesión)



Contenidos

Introducción

Elementos de JSP

- Elementos de secuencias/Scripting
- Directivas
- Acciones

¿Qué es JSP?



JSP = Java Server Pages



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 presentación de los Servlets

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>
```

Ejemplo en JSP compilado a servlet

```
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>");
}
```

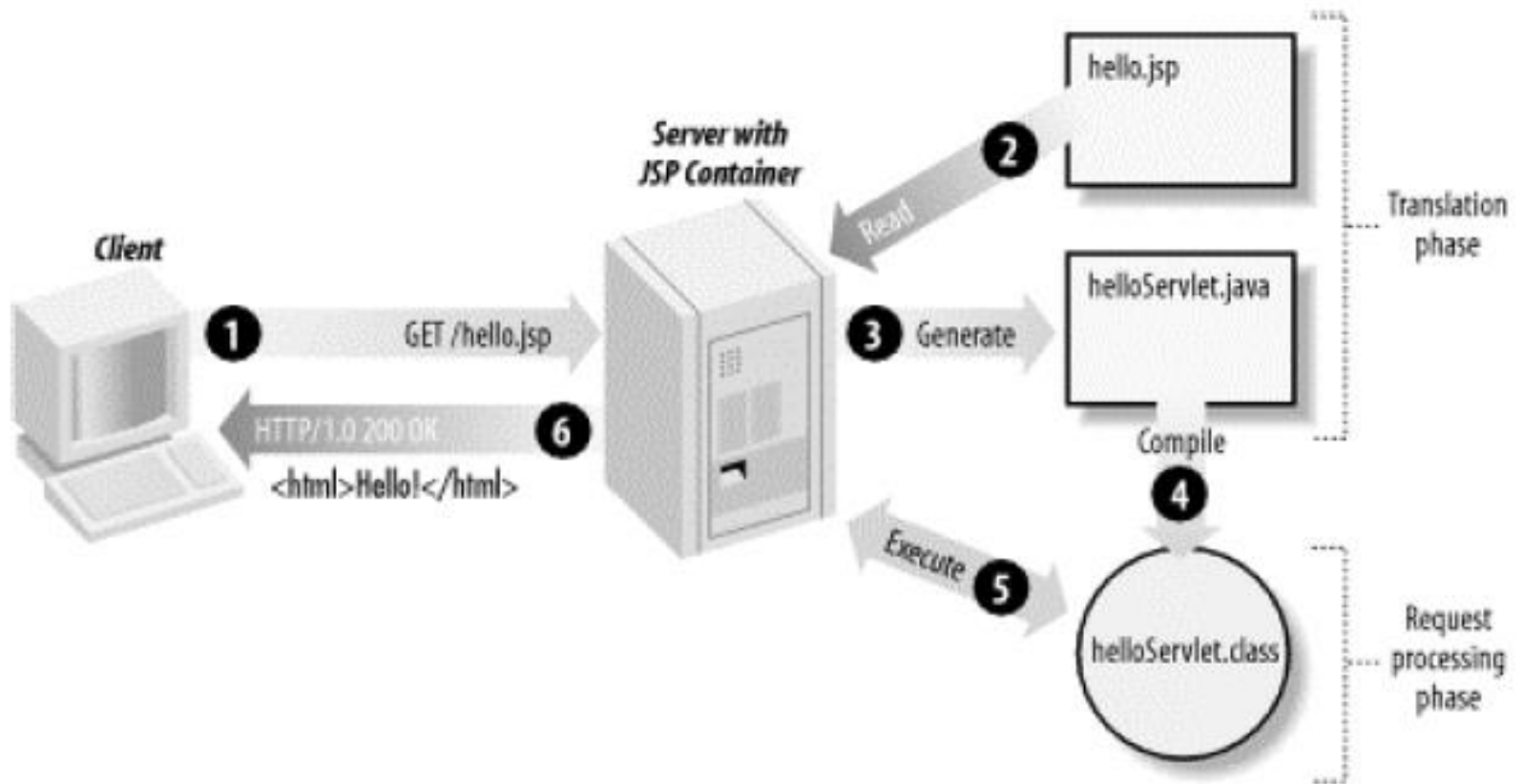

¿Beneficio?

- 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 (Serv. JEE)

```
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 {

    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("<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);
    }
}
```

<HTML>

<HEAD>

<TITLE>Hola Mundo!</TITLE>

</HEAD>

<BODY>

<center>Bienvenido a mi primera página Web!</center>

</BODY>

</HTML>

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

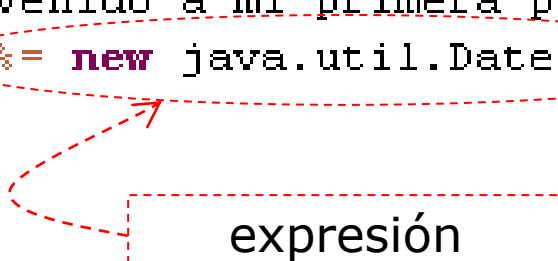
Un ejemplo más “dinámico”

- 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**

Se puede acceder a ellos directamente desde los “scriptlets”, declaraciones y expresiones

Directivas

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

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

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

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>
```

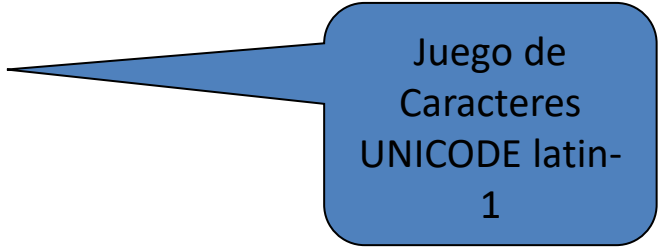
```
<BODY>
```

```
<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.

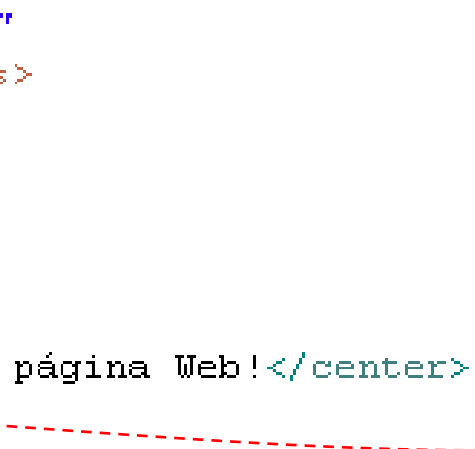
```
<%@ include file="copyright.html" %>
```

Ejemplo: directiva include

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

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

```
<%@ 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>
```



JSP: Acciones

- 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.
- Tipos de acciones
 - Estándar
 - A medida
 - JSTL

Nota: Los elementos XML, al contrario que los HTML, **son sensibles a las mayúsculas**

Acciones

- Elementos XML que realizan determinadas acciones
 - JSP define las siguientes (estándar):

Elemento	Descripción
<code><jsp:useBean></code>	Permite usar un <i>JavaBean</i> desde la página
<code><jsp:getProperty></code>	Obtiene el valor de una propiedad de un componente <i>JavaBean</i> y lo añade a la respuesta
<code><jsp:setProperty></code>	Establece el valor de una propiedad de un <i>JavaBean</i>
<code><jsp:include></code>	Incluye la respuesta de un servlet o una página JSP
<code><jsp:forward></code>	Redirige a otro servlet o página JSP
<code><jsp:param></code>	Envía un parámetro a la petición redirigida a otro servlet o JSP mediante <code><jsp:include></code> o <code><jsp:forward></code>
<code><jsp:plugin></code>	Genera el HTML necesario para ejecutar un <i>applet</i>

JSP: Acción useBean

- Permite cargar y utilizar un JavaBean en la página JSP y así utilizar la reusabilidad de las clases Java

```
<jsp:useBean id="name" class="package.class" />
```

- Esto significa: "usa un objeto de la clase especificada por class, y únelo a una variable con el nombre especificado por id"
- Ahora podemos modificar sus propiedades mediante `<jsp:setProperty>`, o usando un scriptlet y llamando a un método del objeto referenciado por id
 - Para recoger una propiedad se usa `<jsp:getProperty>`

JSP: Acción useBean

- **id**
 - Da un nombre a la variable que referenciará el bean. Se usará un objeto bean anterior en lugar de instanciar uno nuevo si se puede encontrar uno con el mismo id y ámbito (scope)
- **class**
 - Designa el nombre cualificado completo del bean
- **scope**
 - Indica el contexto en el que el bean debería estar disponible. Hay cuatro posibles valores: `page`, `request`, `session`, y `application`
- **type**
 - Especifica el tipo de la variable a la que se referirá el objeto
- **beanName**
 - Da el nombre del bean, como lo suministraríamos en el método `instantiate` de Beans. Está permitido suministrar un `type` y un `beanName`, y omitir el atributo `class`

Ejemplo de uso <jsp:xxx

```
<%@ page contentType="text/html" pageEncoding="iso-8859-1"
    isELIgnored="false"%>

<jsp:useBean id="date" class="uo.dasdi.beans.DateBean" />

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

<%
    if (date.getHour() > 12) {
        Son mas de las 12, son ya las las <%=date.getHour() %>
    }
%>

</BODY>
</HTML>
```

A red dashed line connects the `id="date"` attribute of the `<jsp:useBean>` tag to the `date` object in the `if (date.getHour() > 12)` statement. Another red dashed line connects the `date.getHour()` expression in the `if` statement to the `<%=date.getHour() %>` expression in the output line.

Ejemplo de uso <jsp:xxx código del bean

```
package uo.dasdi.beans;

import java.util.Calendar;

public class DateBean {
    private Calendar calendar = Calendar.getInstance();

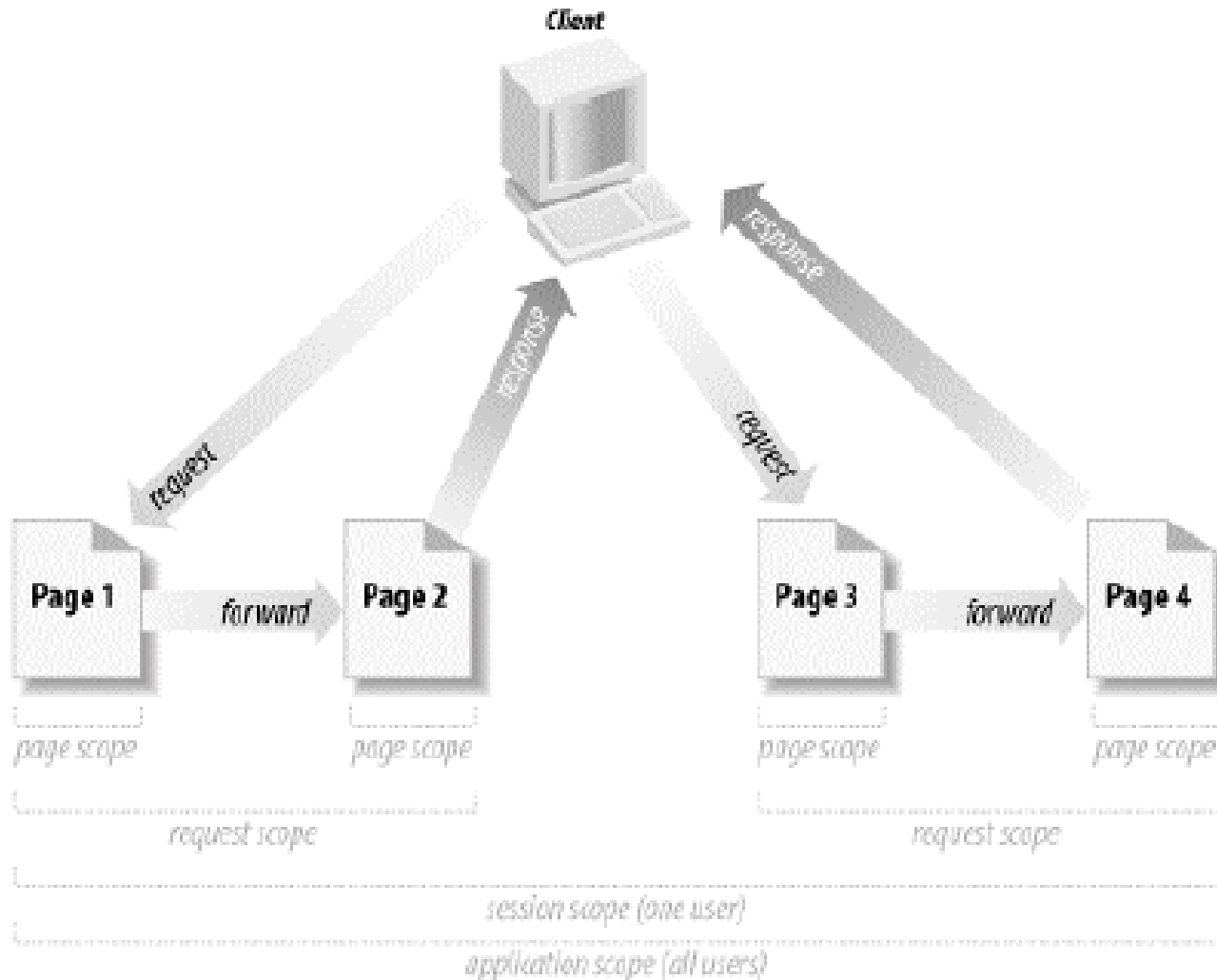
    public Date getDate(){
        return calendar.getTime();
    }

    public int getHour(){
        return calendar.get(Calendar.HOUR_OF_DAY);
    }

    public String getGreeting(){
        int hour = calendar.get(Calendar.HOUR_OF_DAY);
        String greeting;
        if (hour < 13) {
            greeting = "Buenos días";
        } else if (hour < 20) {
            greeting = "Buenas tardes";
        } else {
            greeting = "Buenas noches";
        }
        return greeting;
    }
}
```

Comunicación entre jsp

Posibles ámbitos



Acciones JSTL, ejemplo

```
<%@ page contentType="text/html" pageEncoding="iso-8859-1" isELIgnored="false"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<jsp:useBean id="date" class="uo.dasdi.beans.DateBean" />

<HTML>
<HEAD>
<TITLE>Hola Mundo!</TITLE>
</HEAD>
<BODY>
<center>Bienvenido a mi primera página Web!</center>
<c:if test="${date.hour > 12}">
    <c:out value="Son mas de las 12, son las ${date.hour}" />
</c:if>
</BODY>
</HTML>
```

Con tag-libs no hay scriptlet



Encadenamiento de Servlets y JSPs

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 =`
`getContext().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

Cómo incluir contenido estático o dinámico

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

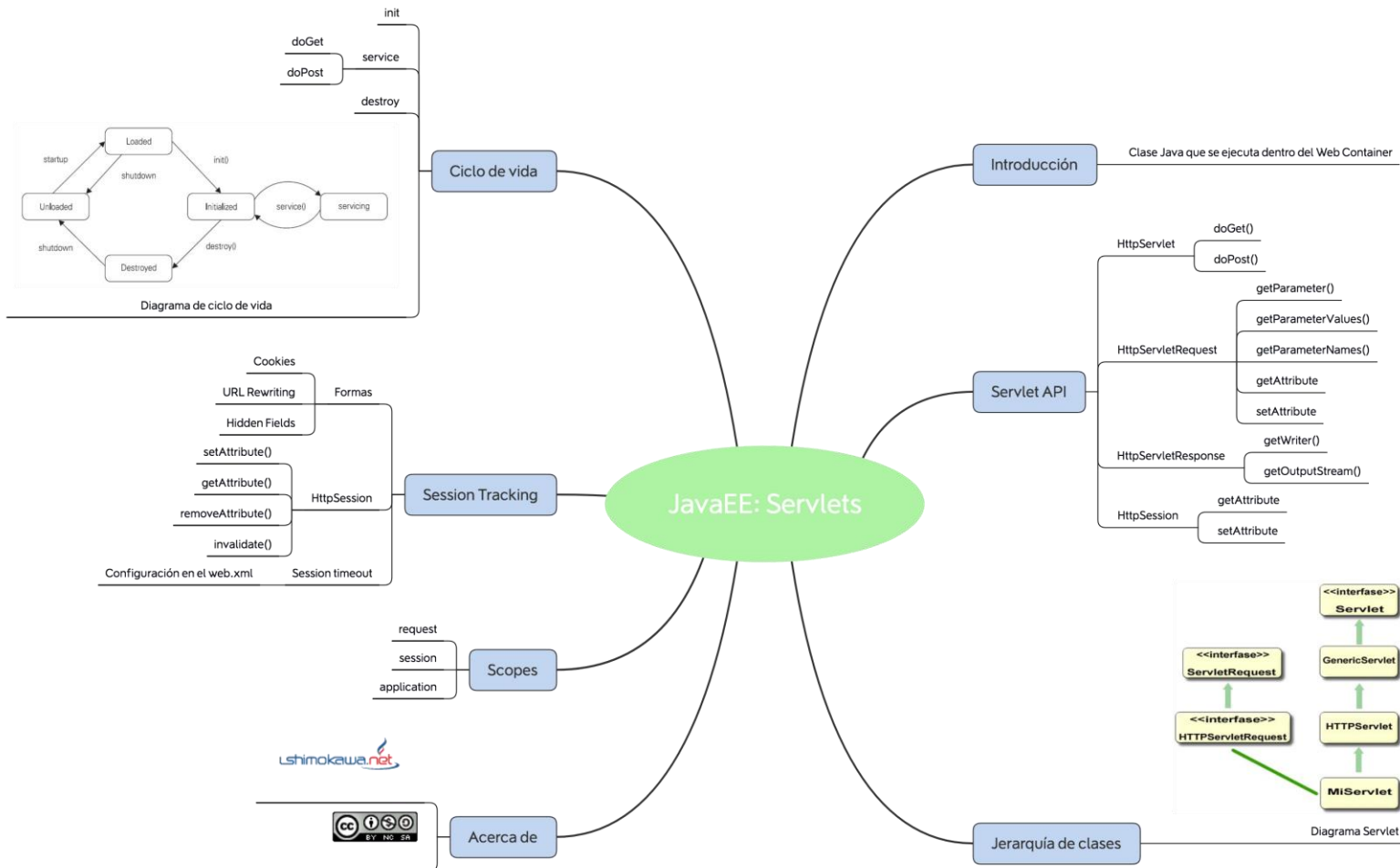
```
out.println("...");
requestDispatcher dispatcher = getServletContext().getRequestDispatcher(address);
dispatcher.include(request, response);
out.println("...");
```

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

MindMap JEE



Bibliografía

- **Un Best sheller en JEE – Marty Hall**
 - <http://www.coreservlets.com>
- **JEE 7**
 - <https://docs.oracle.com/javaee/7/api/>
- **Java 7 API**
 - <http://docs.oracle.com/javase/7/docs/api/>