

Bienvenidos





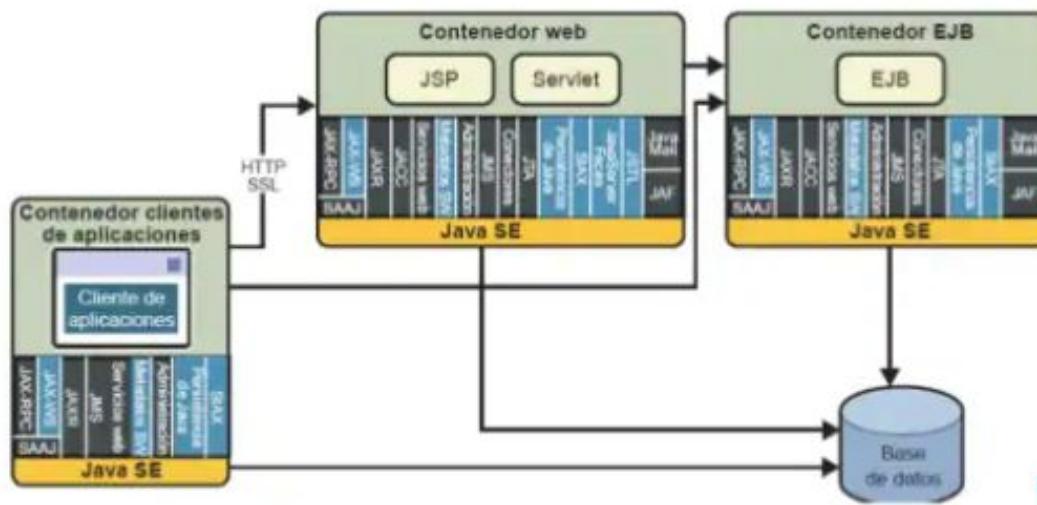
- Introducción JEE
- Qué es un servidor de aplicaciones?
- Hablemos de Java Specification Request
- Qué es Jakarta EE?
- Java EE and Spring Framework

Introducción a Java Enterprise

- ¿Qué es Java EE?

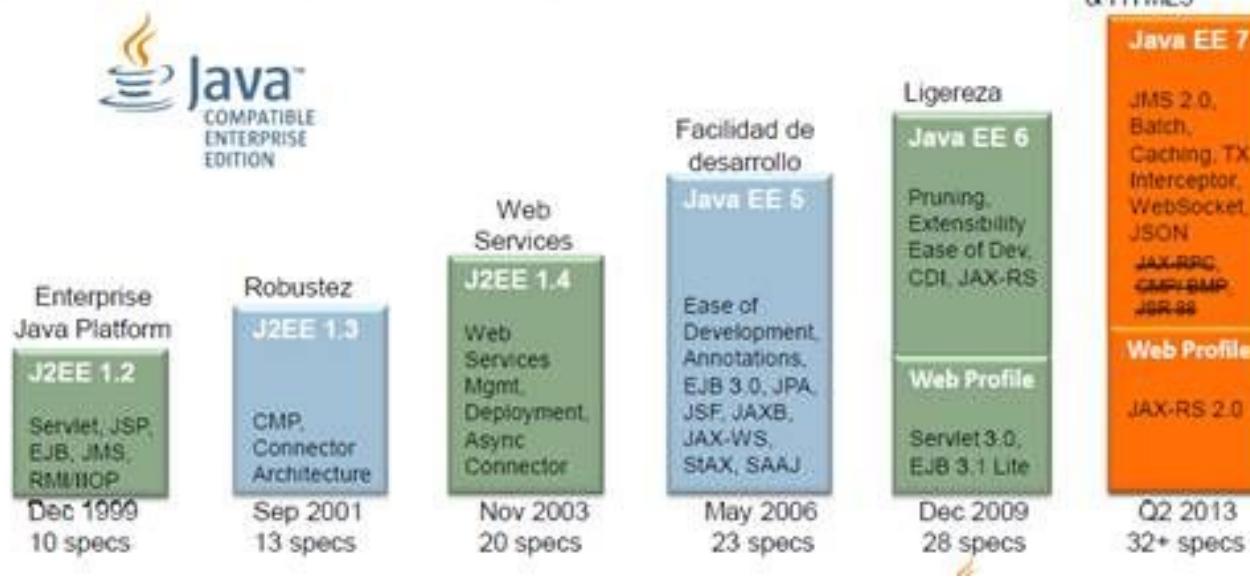
Java Platform, Enterprise Edition (antes J2EE, ahora Java EE) es un estándar para el desarrollo de aplicaciones empresariales (portables, robustas, escalables y seguras) usando tecnología Java.

Java EE es una especificación, no un producto. Los productos que cumplen con la especificación son realizados por terceras empresas u organizaciones.



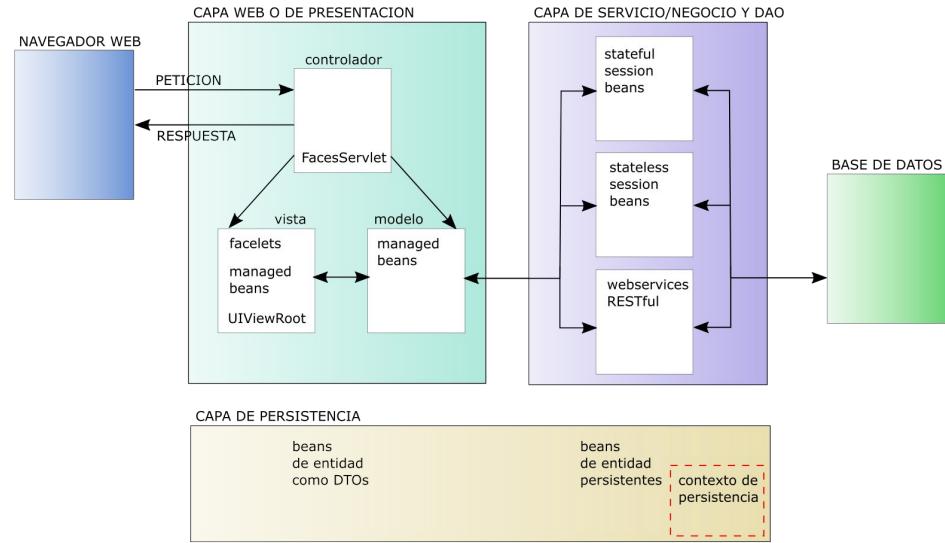
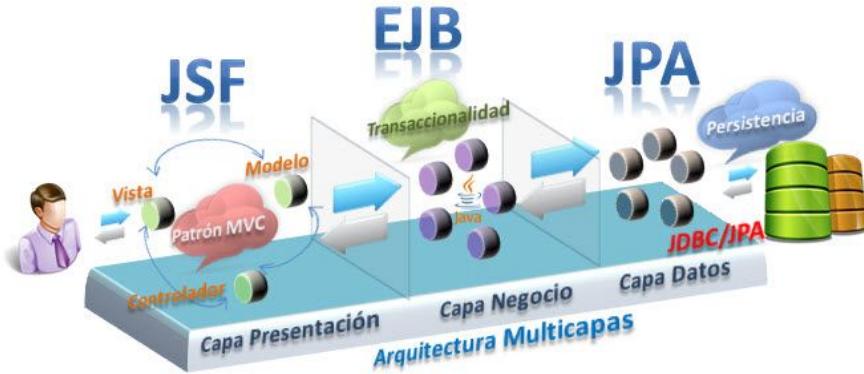


Java EE pasado, presente, & futuro



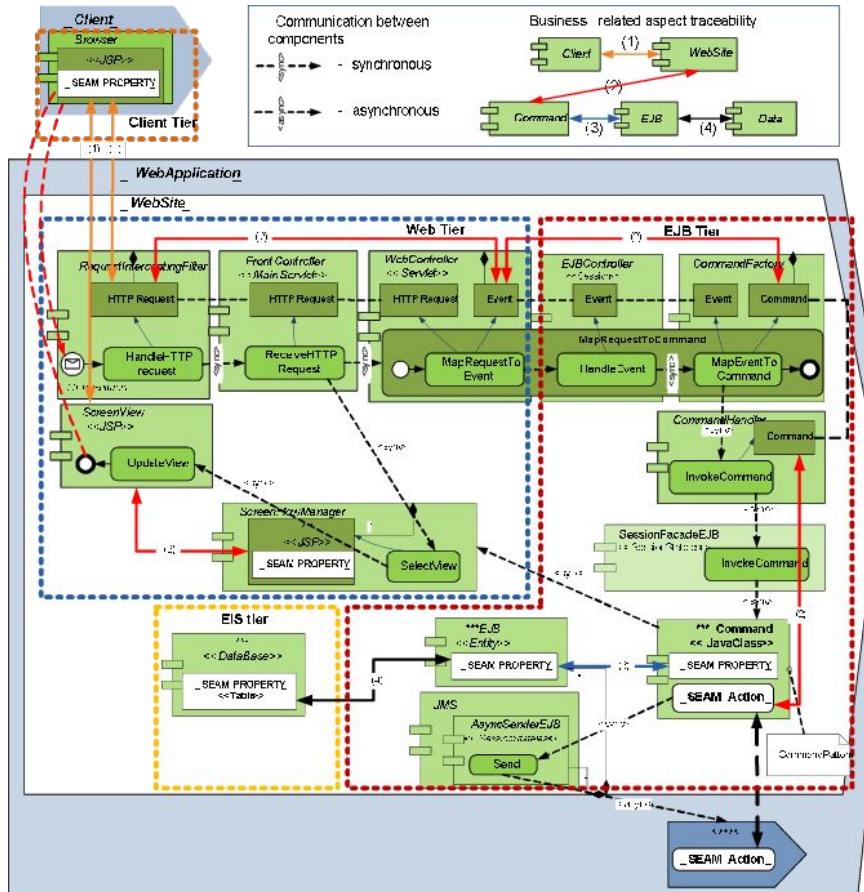
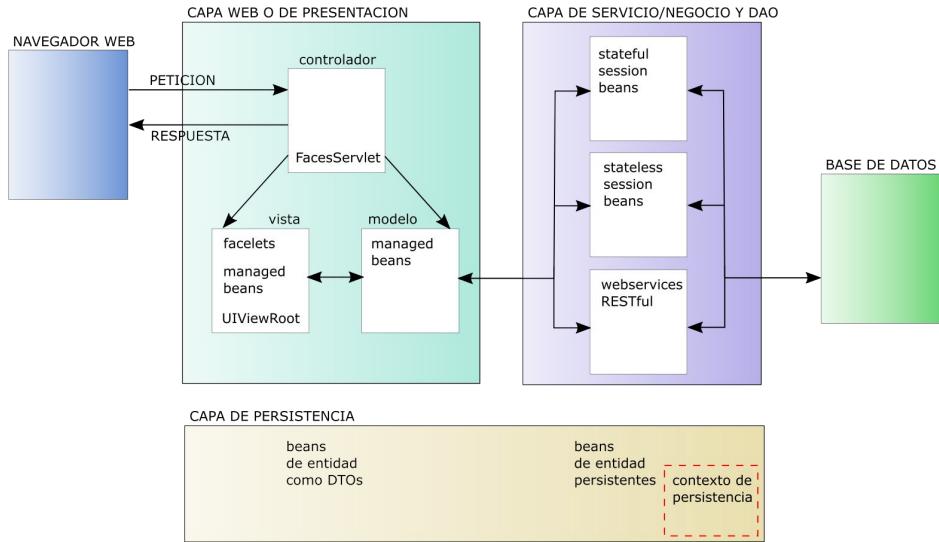
Introducción a Java Enterprise

Modelo MVC



Introducción a Java Enterprise

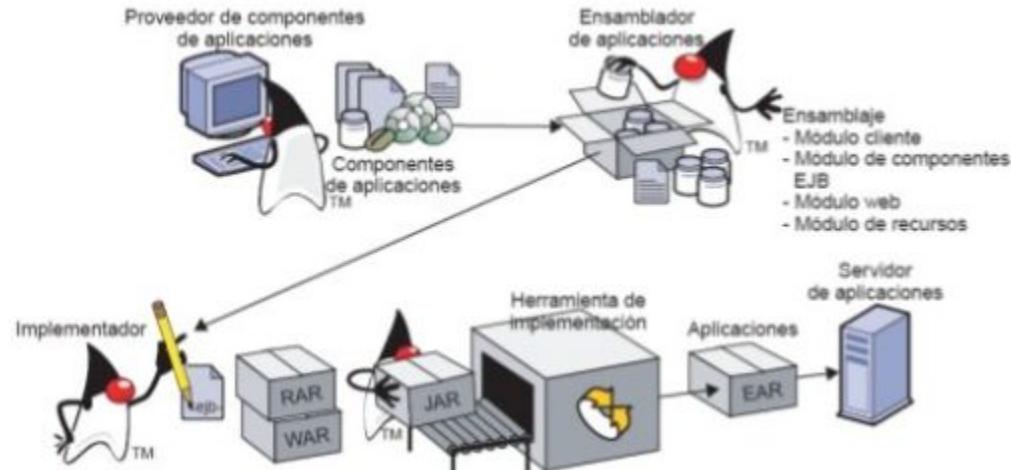
Modelo MVC



Introducción a Java Enterprise

Fases del Desarrollo

- Diseño
- Codificación
- Creación de descriptores de implementación
- Empaquetado
- Ensamblaje
- Despliegue



- Archivos de almacenamiento web (WAR)
- Archivos de almacenamiento Java (JAR)
- Archivos de almacenamiento de recursos (RAR)
- Archivos de almacenamiento empresariales (EAR)

Introducción a Java Enterprise

Servidores de Aplicaciones

Un servidor de aplicaciones Java EE es una plataforma que da soporte a los servicios definidos en la especificación, desde aplicaciones web con servlets y JSP, hasta aplicaciones distribuidas con soporte transaccional utilizando Enterprise JavaBeans (EJB) o servicios web.



<https://www.oracle.com/java/technologies/java-ee-glance.html>

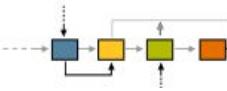
Introducción a Java Enterprise

Java Specifications Request

Los JSR son documentos formales que describen las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma Java.



Java
Community
Process



Community Development of Java Technology Specifications

<https://jcp.org/en/jsr/all>

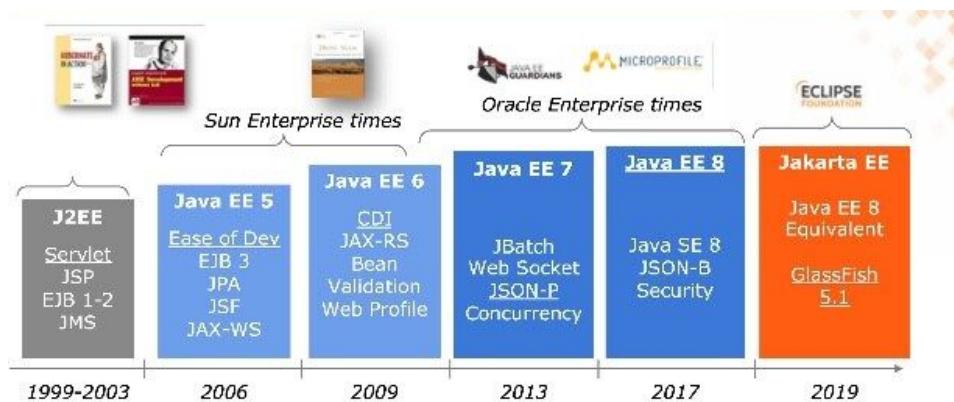
JSR #	Especificación o Tecnología
30	Connected Limited Device Configuration (CLDC) 1.0 for Java ME
37	Mobile Information Device Profile (MIDP) 1.0 for Java ME
54	Java Database Connectivity (JDBC) 3.0
68	Java Platform, Micro Edition (Java ME) 1.0
75	PDA Optional Packages for the J2ME Platform
80	Java USB API
82	Java APIs for Bluetooth
116	SIP Servlet API 1.0
118	Mobile Information Device Profile (MIDP) 2.0 for Java ME
120	Wireless Messaging API (WMA)
135	Java Mobile Media API (MMAPI) for Java ME
139	Connected Limited Device Configuration (CLDC) 1.1 for Java ME
176	Java 2 Platform, Standard Edition (J2SE) 5.0 (Tiger)
177	Security and Trust Services API for J2ME (SATSA)
179	Location API 1.0 for Java ME
180	Session Initiation Protocol (SIP) API for Java ME
184	Mobile 3D Graphics API for Java ME 1.0 and 1.1
185	Java Technology for the Wireless Industry (JTWI)
187	Instant messaging (Java ME and Java SE)
203	More New I/O APIs for the Java Platform (NIO2)
205	Wireless Messaging API 2.0 (WMA) 2.0
211	Content Handler API
218	Connected Device Configuration (CDC) 1.1 for Java ME
221	Java Database Connectivity (JDBC) 4.0
226	Scalable 2D Vector Graphics API for Java ME
234	Advanced Multimedia Supplements API for Java ME
253	Mobile Telephony API (MTA)
256	Mobile Sensor API
271	Mobile Information Device Profile (MIDP) 3.0 for Java ME
282	Real-Time Specification for Java (RTSJ) 1.1
293	Location API 2.0 for Java ME
912	Java 3D API 1.3

Introducción a Java Enterprise

Jakarta EE

Jakarta EE pasa a ser la nueva plataforma opensource de Java EE gestionada por Eclipse Foundation.

Además los paquetes de Jakarta EE , que en **Jakarta EE 8.0** se mantuvieron iguales, han pasado a ser **jakarta.*** en **Jakarta EE 9.0**. Desapareciendo los **javax.***



Introducción a Java Enterprise

Java EE vs Spring Boot



Introducción a Java Enterprise

Java EE vs Spring Boot

What Spring can do



Microservices

Quickly deliver production-grade features with independently evolvable microservices.



Reactive

Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.



Cloud

Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.



Web apps

Frameworks for fast, secure, and responsive web applications connected to any data store.



Serverless

The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.



Event Driven

Integrate with your enterprise. React to business events. Act on your streaming data in realtime.



Batch

Automated tasks. Offline processing of data at a time to suit you.

Introducción a Java Enterprise



Java Server Faces

- **Introducción a JSP y Servlets**
- **Introducción a Java Server Pages**
- **Patrón MVC usando JSF**
- **Managed Beans**
- **Navegación**
- **Etiquetas**
- **Manejo de eventos**



Servlets

Modelo:

Implementa el núcleo de la funcionalidad de la aplicación.
Encapsula el estado de la aplicación.

Totalmente independiente de la vista y el controlador.

Vista:

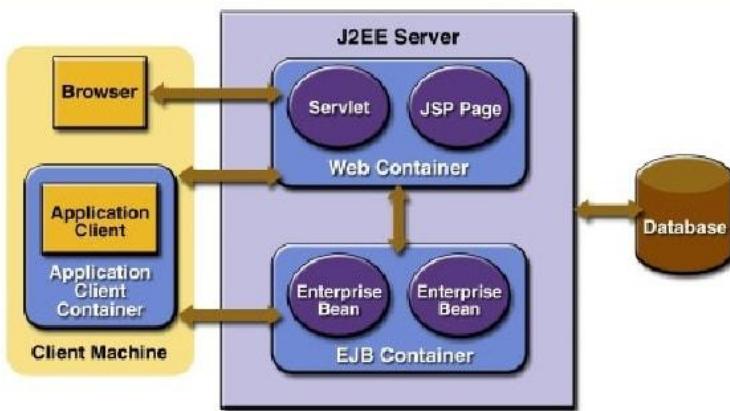
Representación del modelo.

Puede acceder al estado del modelo, pero no puede modificarlo.

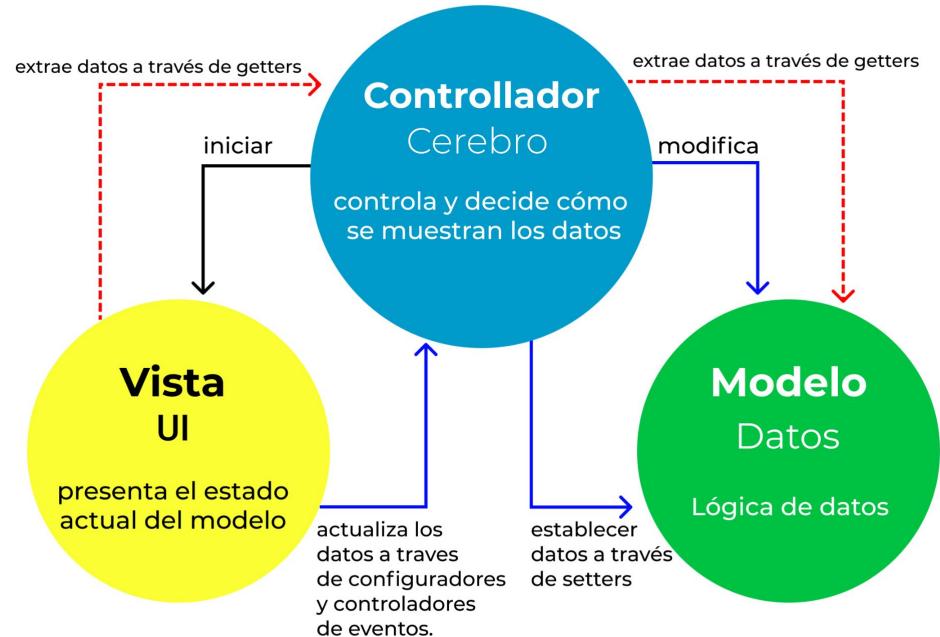
Puede recibir notificaciones de cambios en el estado del modelo.

Controlador:

Responde a las peticiones de cliente cambiando la presentación de las vistas y/o actualizando el estado del modelo



Patrones de Arquitectura MVC

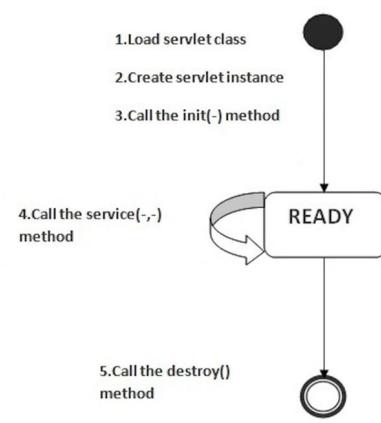
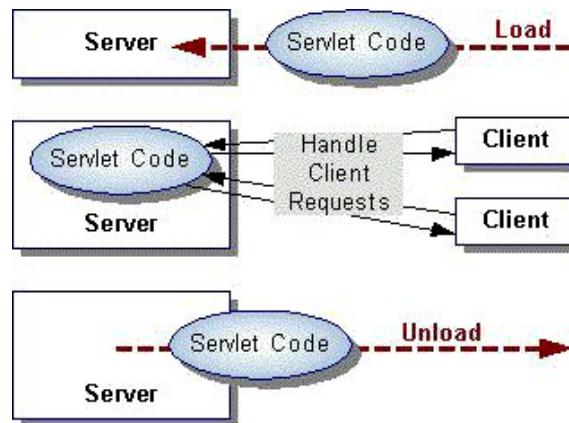


Servlets

Un **servlet** es un programa Java que se ejecuta en un servidor Web y construye o sirve páginas web. De esta forma se pueden construir páginas dinámicas, basadas en diferentes fuentes variables: datos proporcionados por el usuario, fuentes de información variable (páginas de noticias, por ejemplo), o programas que extraigan información de bases de datos.

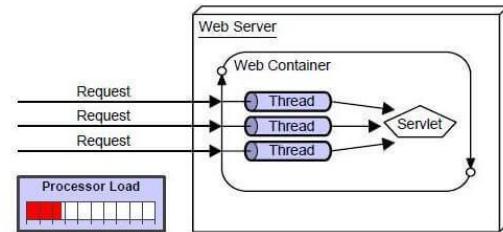
Comparado con un CGI, un servlet es más sencillo de utilizar, más eficiente (se arranca un hilo por cada petición y no un proceso entero), más potente y portable. Con los servlets podremos, entre otras cosas, procesar, sincronizar y coordinar múltiples peticiones de clientes, reenviar peticiones a otros servlets o a otros servidores, etc.

Ciclo de vida



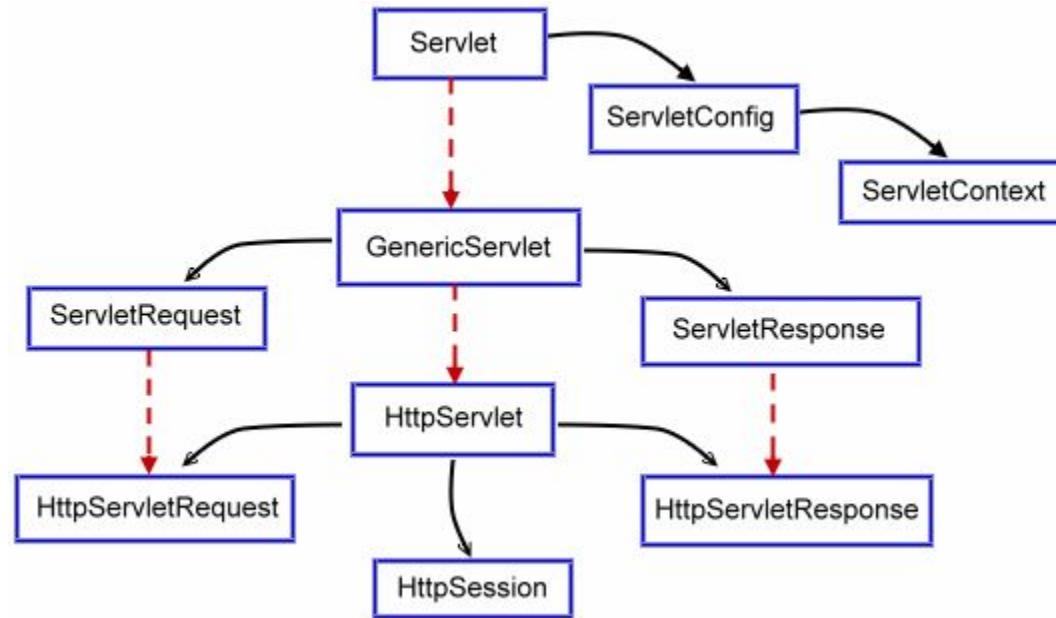
Propiedades

- **Manejo de Sesiones:** Se puede hacer seguimiento de usuarios a través de distintos servlets a través de la creación de sesiones.
- **Utilización de Cookies:** Las *cookies* son pequeños datos en texto plano que pueden ser guardados en el cliente. La API de servlets permite un manejo fácil y limpio de ellas.
- **Multi-thread:** Los servlets soportan el acceso concurrente de los clientes, aunque hay que tener especial cuidado con las variables compartidas a menos que se utilice la interfaz **SingleThreadModel**.
- **Programación en Java:** Se obtienen las características de multiplataforma o acceso a APIs como JDBC, RMI, etc.



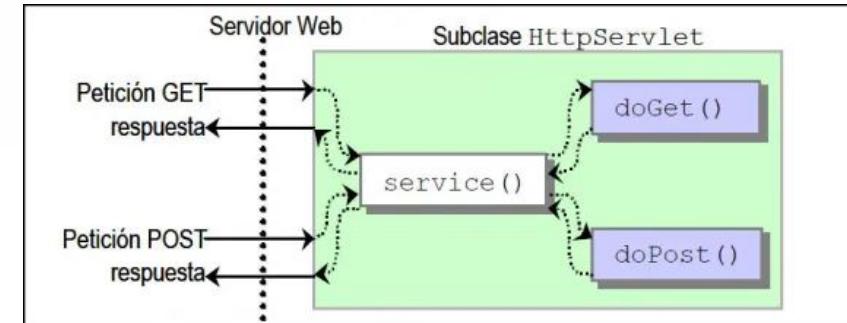
Servlets

javax.servlet ----- jakarta.servlet



Estructura de Clases para los Servlets

Método Service



Que puede hacer un Servlet

- Leer los datos enviados por un usuario
 - Usualmente de formularios en páginas Web
 - Pueden venir de applets de Java o programas cliente HTTP.
- Buscar cualquier otra información sobre la petición que venga incluida en esta
 - Detalles de las capacidades del navegador, cookies, nombre del host del cliente, etc.
- Generar los resultados
 - Puede requerir consultas a Base de Datos, invocar a otras aplicaciones, computar directamente la respuesta, etc.
- Dar formato a los resultados en un documento
 - Incluir la información en una página HTML
- Establecer los parámetros de la respuesta HTTP
 - Decirle al navegador el tipo de documento que se va a devolver, establecer las cookies, etc.
- Enviar el documento al cliente

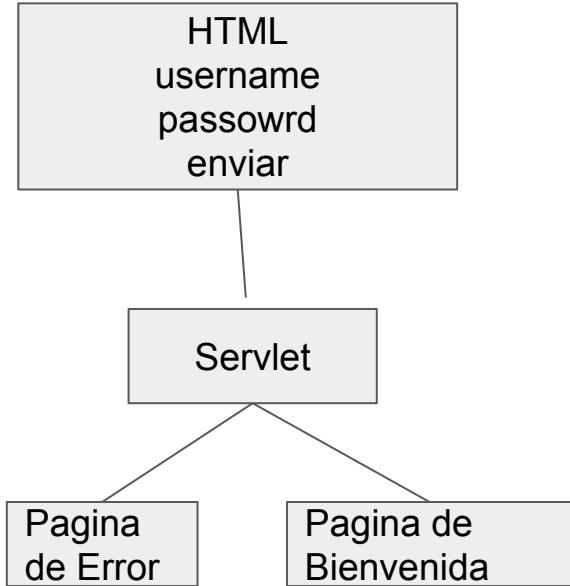
Cómo Invocar a un Servlet

Desde la barra de URL del navegador
A través del action en un Formulario

Servlets

```
32     @Override  
33     protected void doPost(HttpServletRequest request,  
34         HttpServletResponse response)  
35     throws ServletException, IOException {  
36         response.setContentType("text/html;charset=UTF-8");  
37         PrintWriter out = response.getWriter();  
38         try {  
39             //TODO output your page here  
40             out.println("<html>");  
41             out.println("<head>");  
42             out.println("<title>Servlet0</title>");  
43             out.println("</head>");  
44             out.println("<body><center>");  
45             out.println("<h1>Respuesta del Servlet0 en " +  
46                         request.getContextPath () + "</h1>");  
47             out.println("</center></body>");  
48             out.println("</html>");  
49         } finally {  
50             out.close();  
51         }  
52     }
```

Ejemplo de Servlet



Servlets

```
<?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>aprendejava</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>aprendejava</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/aprendejava-servlet.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

```

web.xml

Anotaciones desde JEE 6

```
@WebServlet(name = "TestServlet", urlPatterns = {"/search"})
```

web.xml / annotations

```
<servlet>
    <servlet-name>Example</servlet-name>
    <servlet-class>com.servlets</servlet-class>

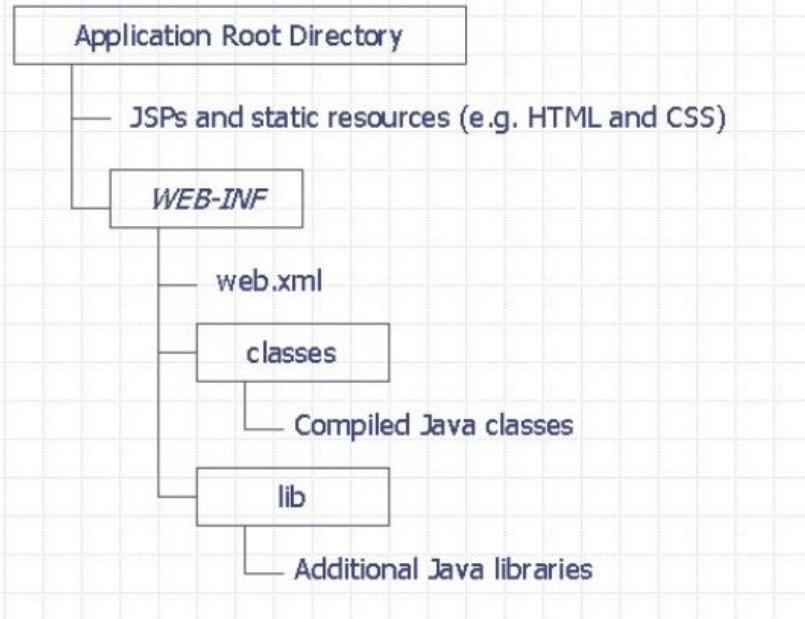
    <init-param>
        <param-name>path</param-name>
        <param-value>/files/</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>welcome</servlet-name>
</servlet-mapping>
```

import javax.servlet.annotation.WebServlet;

```
@WebServlet(name = "welcome", urlPatterns = "/files/*", initParams = {@WebInitParam{ name = "path", value = "/files/" }})
public class Download extends HttpServlet {
```

Preferred method

Servlets



```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/Simple")
public class Simple extends HttpServlet {
    private static final long serialVersionUID = 1L;

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

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

        out.print("");
        out.print("Hello Servlet");
        out.print("");

    }
}
```

Servlets

Interface HttpServletRequest

- Acceso a datos del cliente :
 - El método **getParameter()** retorna el valor de un parámetro y lo almacena como String.
 - El método **getParameterValues()** retorna los valores de los varios parámetros (checkboxes, listas desplegables múltiples) y lo almacena en un String[].
 - El método **getParameterNames()** provee los nombres de los parámetros y lo almacena un java.util.Enumeration.

Interface ServletResponse

- La interface ServletResponse provee los métodos para contestar al cliente:
 - Suministra un flujo de salida ServletOutputStream y un Writer a través del cual el Servlet puede enviar datos al cliente.
 - Permite al Servlet configurar el tipo MIME (Multipurpose Internet Mail Extension) a enviar.
- Las interfaces que se extienden de la interface ServletResponse permiten aumentar las capacidades de un protocolo específico.
 - La interface **HttpServletResponse** contiene métodos que permiten manipular la información de cabecera del HTTP.

Interface HttpServletResponse

- El objeto HttpServletResponse provee dos formas de enviar datos al cliente :
 - El método **getWriter()** que retorna un objeto PrintWriter. Este objeto le permite al Servlet enviar texto plano (como HTML) al cliente.
 - El método **getOutputStream()** que retorna un objeto ServletOutputStream. Este objeto le permite al Servlet enviar datos en binario (doc, pdf, exe, ppt, zip, entre otros) al cliente.

Servlets

Recuperando datos desde un Formulario

Metodo	Descripción
<code>String getParameter (String paramName)</code>	Nos devuelve la primera ocurrencia de la URL.
<code>String[] getParameterValues (String paramName)</code>	Nos devuelve un array con los valores de todas las ocurrencias de name en la URL.
<code>Enumeration getParameterNames() o getParamaterMap()</code>	Nos devuelve o un Enumeration o mapa de todos los parámetros de la petición para recorrerlos.

HOBBIES

Playing Cricket

Watching Movies

Listening Songs

Playing Basketball

```
String[] favoriteFruits = request.getParameterValues("favoriteFruit");
```

```
String nom = request.getParameter("nom");
String ape = request.getParameter("ape");
```

```
Enumeration paramNames = request.getParameterNames();
while(paramNames.hasMoreElements()) {
    String paramName = (String)paramNames.nextElement();
    out.print("<TR><TD>" + paramName + "\n<TD>");
    String[] paramValues = request.getParameterValues(paramName);
    if (paramValues.length == 1) {
        String paramValue = paramValues[0];
        if (paramValue.length() == 0) out.println("<I>No Value</I>");
        else out.println(paramValue);
    } else {
        out.println("<UL>");
        for(int i=0; i<paramValues.length; i++) {out.println("<LI>" +
+paramValues[i]); }
        out.println("</UL>");
    }
}
out.println("</TABLE>\n</BODY></HTML>");}
```

Servlets

Leyendo la Cabecera HTTP

Ejemplo típica cabecera HTTP

GET /servlet/Search?keywords=servlets+jsp HTTP/1.1

Accept: image/gif, image/jpg, */*

Accept-Encoding: gzip

Connection: Keep-Alive

Cookie: userID=id456578

Host: www.somebookstore.com

Referer: http://www.somebookstore.com/findbooks.html

User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)

```
for(Enumeration e = request.getHeaderNames();e.hasMoreElements() ;){  
    headername = (String)e.nextElement();  
    out.println(request.getHeader(headername) + "<br/>");  
}
```

Metodo	Descripción
<i>String getHeader (String headerName)</i>	Devuelve uno de los valores asociados a la cabecera <i>headerName</i>
<i>Enumeration getHeaders (String headerName)</i>	Devuelve en forma de Enumeration todos los valores asociados a la cabecera <i>headerName</i>
<i>Enumeration getHeaderNames()</i>	Devuelve todas las nombres de cabeceras. Esto es útil para obtener los valores asociados cuando no conocemos los nombres de las propias cabeceras.

Servlets

Usando las Cookies

```
String cookieName = "userID";
Cookie[] cookies = request.getCookies();
if (cookies != null) {
    for(Cookie cookie: cookies) {
        if (cookieName.equals(cookie.getName())) {
            doSomethingWith(cookie.getValue());
        }
    }
}
```

Servlets

Colaboración entre Servlets

- Obtener el objeto RequestDispatcher:
 - Se obtiene el objeto **RequestDispatcher** usando el método **getRequestDispatcher** del objeto **ServletContext**.
- Reenviar el requerimiento del cliente:
 - Se usa el método **forward** del objeto **RequestDispatcher**.
 - Si se accede antes al objeto **PrintWriter** o **ServletOutputStream** no se podrá usar el método **forward**.

- Obtener el objeto RequestDispatcher:
 - **RequestDispatcher dispatcher = getServletContext().getRequestDispatcher("/respuesta.jsp");**
- Reenviar el requerimiento del cliente :
 - **dispatcher.forward(request,response);**

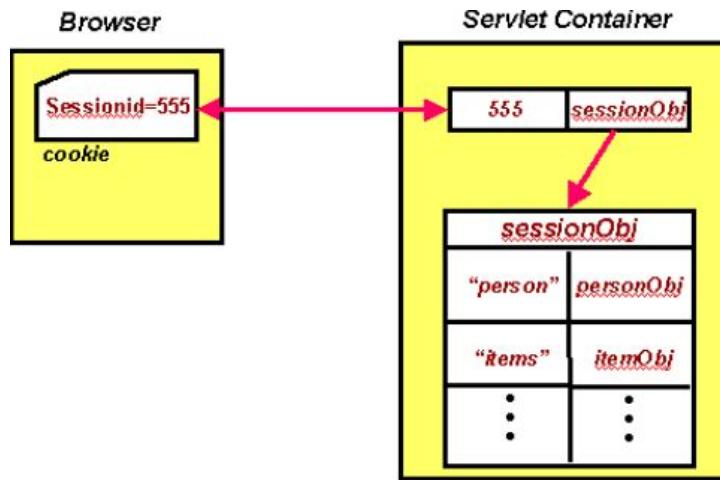
```
if(userid.equals("admin") && password.equals("adminjuga")) {  
    RequestDispatcher dispatch = request.getRequestDispatcher("/Welcome");  
    dispatch.forward(request, response);  
}  
else {  
    RequestDispatcher dispatch = request.getRequestDispatcher("/Login");  
    dispatch.include(request, response);  
    out.println("Login Salah");  
}
```



Servlets

Colaboración entre Servlets

- Los servlets de una aplicación pueden compartir recursos usando alguno de los ambientes o scopes:
 - Context (o application)
 - Session
 - Request
 - Page



- Estos ambientes o scopes tienen los siguientes métodos:
 - setAttribute(nombre, valor)
 - getAttribute(nombre)
 - removeAttribute(nombre)

HttpSession

El Servlet API trae el soporte de sesiones en la interfase `javax.servlet.http.HttpSession`.

El contenedor de Servlet crea un nuevo objeto HttpSession cuando inicia una sesión para un cliente.

Además de representar la sesión, este objeto actúa como un contenedor para la información relacionada a la sesión.

Normalmente, se necesitan hacer 3 acciones con una sesión HTTP:

- Recuperar la sesión asociada con la petición.
- Agregar o remover atributos de sesión.
- Cerrar o invalidar la sesión si es necesario.

Servlets

Colaboración entre Servlets

HttpSession

Recuperar la sesión:

- HttpSession session = request.getSession();

Agregar un objeto a la sesión:

- session.setAttribute(NOMBRE, OBJETO);

Recuperar un objeto de la sesión:

- Object lista =
(Object)session.getAttribute(NOMBRE);

Invalidando una sesión:

- session.invalidate();

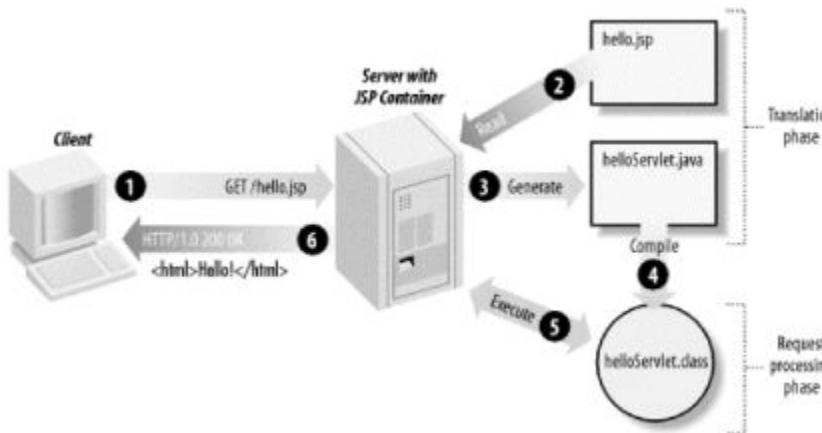
Tiempo de expiración de la sesión

La configuración se realiza en el web.xml. El tiempo establecido está en MINUTOS. El valor 0 indicaría que la sesión nunca expirará.

```
<web-app>
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>
</web-app>
```



- Una tecnología que permite combinar código HTML estático con código generado dinámicamente en un mismo fichero.
- Ventajas:
 - Separación de datos estáticos/dinámicos.
 - Independencia de formato/plataforma.
 - Sencillez (sabiendo servlets)
- Los JSP nos permiten separar la parte dinámica de nuestras páginas Web del HTML estático. Simplemente escribimos el HTML regular de la forma normal y encerramos el código de las partes dinámicas en unas etiquetas especiales, la mayoría de las cuales empiezan con "<%>" y terminan con "%>".
- Damos extensión .jsp.
- Aunque el código parezca mas bien HTML, el servidor lo traduce a un servlet en la primera petición.
- 3 elementos en un JSP:
 - Elementos script (scriptlets)
 - Directivas
 - Acciones



Expresión JSP

```
<%= expression %>;
```

La Expresión es evaluada y situada en la salida.

El equivalente XML es

```
<jsp:expression> expression </jsp:expression>
```

Las variables predefinidas son `request`, `response`, `out`, `session`, `application`, `config`, y `pageContext`.

Scriptlet JSP

```
<% code %>;
```

El código se inserta en el método service.

El equivalente XML es:

```
<jsp:scriptlet> code </jsp:scriptlet>.
```

Declaración JSP

```
<%! code %>
```

El código se inserta en el cuerpo de la clase del servlet, fuera del método service. Se utiliza para declarar variables y métodos.

El equivalente XML es:

```
<jsp:declaration> code </jsp:declaration>.
```

Directivas JSP

Afectan a la estructura general de la clase servlet. Normalmente tienen la siguiente forma:

```
<%@ directive att="val" %>
```

También podemos combinar múltiples selecciones de atributos para una sola directiva:

```
<%@ directive attribute1="value1"
           attribute2="value2"
           ...
           attributeN="valueN" %>
```

Comentario JSP

```
<!-- comment -->
```

Comentario ignorado cuando se traduce la página JSP en un servlet. Si queremos un comentario en el HTML resultante, usamos la sintaxis de comentario normal del HTML <!-- comment -->.

Acciones jsp

```
<jsp:nombre-de-la-acción att=valor att2=valor2 .../>
```

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

SCRIPTLETS: <% código %> que se insertan dentro del método service del servlet.

- Tienen acceso a las mismas variables que las expresiones.
- El código dentro de un scriptlet se insertará **exactamente** como está escrito, y cualquier HTML estático (plantilla de texto) anterior o posterior al scriptlet se convierte en sentencias print.

Ejemplo:

```
<html>
<% for ( int i = 0 ; i < 10 ; i ++ )
    out.println("<br>" + i); %>
</html>
```

EXPRESIONES: <%= expresión %> Se evalúan y se insertan en la salida.

- Se tiene acceso a variables:

- request, el HttpServletRequest
- response, el HttpServletResponse
- session, el HttpSession asociado con el request (si existe)
- out, el PrintWriter (una versión con buffer del tipo JspWriter) usada para enviar la salida al cliente.

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

Ejemplo: ¿Como visualizar un contador de visitas en una JSP?

- El equivalente en XML es usar una sintaxis alternativa para las expresiones JSP:

```
<jsp:expression>
Expresión Java
</jsp:expression>
```

- Los elementos XML, al contrario que los del HTML, son sensibles a las mayúsculas.

SCRIPTLETS: <% código %> que se insertan dentro del método service del servlet.

- Tienen acceso a las mismas variables que las expresiones.
- El código dentro de un scriptlet se insertará **exactamente** como está escrito, y cualquier HTML estático (plantilla de texto) anterior o posterior al scriptlet se convierte en sentencias print.

Ejemplo:

```
<html>
<% for ( int i = 0 ; i < 10 ; i ++ )
    out.println("<br>" + i); %>
</html>
```

DECLARACIONES: <%! código %> que se insertan en el cuerpo de la clase del servlet, fuera de cualquier método existente.

- Permite insertar métodos, variables...
- No generan salida alguna. Se usan combinadas con scriptlets.

```
<%! private int accessCount = 0; %>
Accesses to page since server reboot:
<%= ++accessCount %>
```

```
<% if (Math.random() < 0.5) { %>
    Have a <B>nice</B> day!
<% } else { %>
    Have a <B>lousy</B> day!
<% } %>
■ Que se traducirá en:
if (Math.random() < 0.5) {
    out.println("Have a <B>nice</B> day!");
else {
    out.println("Have a <B>lousy</B> day!");
```

- Como con los scriptlet, si queremos usar los caracteres "%>", ponemos "%\>".
- El equivalente XML de <%! Código %> es:

```
<jsp:declaration>
    Código
</jsp:declaration>
```

- Afectan a la estructura general de la clase servlet. Normalmente tienen la siguiente forma:

```
<%@ directive attribute="value" %>
```

- También podemos combinar múltiples selecciones de atributos para una sola directiva:

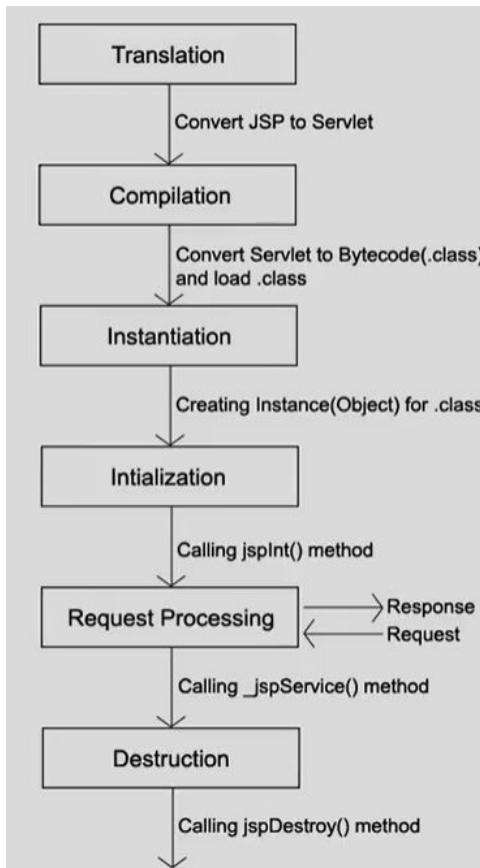
```
<%@ directive attribute1="value1"
               attribute2="value2"
               ...
               attributeN="valueN" %>
```

- session="true|false"**. Un valor de 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. Un valor de 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="true|false"**. Un valor de true (por defecto) indica que el buffer debería descargarse cuando esté lleno. Un valor de false, raramente utilizado, indica que se debe lanzar una excepción cuando el buffer se sobrecargue. Un valor de false es ilegal cuando usamos buffer="none".

PAGE:

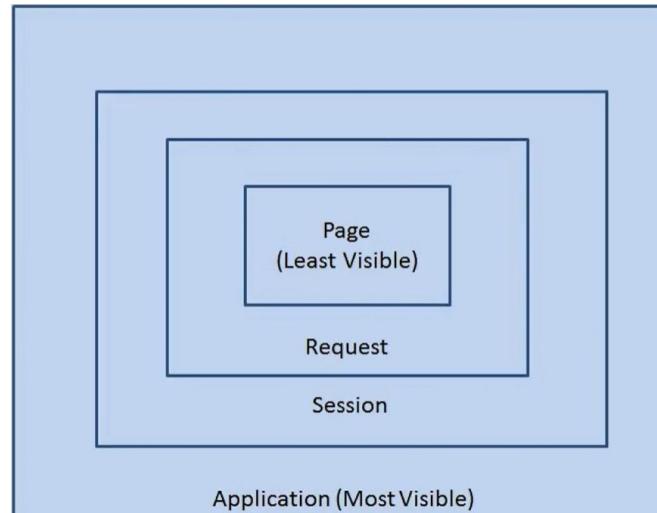
- import="package.class" o import="package.class1,...,package.classN"**. Esto permite especificar los paquetes que deberían ser importados. El atributo import es el único que puede aparecer múltiples veces.
- ContentType = "MIME-Type" o contentType = "MIME-Type; charset = Character-Set"** Esto especifica 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"**. Un valor de true (por defecto) indica un procesamiento del servlet normal, donde múltiples peticiones pueden procesarse simultáneamente con un sólo ejemplar del servlet, bajo la suposición que el autor sincroniza los recursos compartidos. Un valor de false indica que el servlet debería implementar SingleThreadModel.
- extends="package.class"**. Esto indica la superclase del servlet que se va a generar. Debemos usarla con extrema precaución, ya que el servidor podría utilizar una superclase personalizada.
- info="message"**. Define un string que puede usarse para ser recuperado mediante el método getServletInfo.
- errorPage="url"**. Especifica una página JSP que se debería procesar si se lanzará cualquier Throwable pero no fuera capturado en la página actual.
- isErrorPage="true|false"**. Indica si la página actual actúa o no como página de error de otra página JSP. El valor por defecto es false.
- language="java"**. En algunos momentos, esto está pensado para especificar el lenguaje a utilizar. Por ahora, no debemos preocuparnos por él ya que java es tanto el valor por defecto como la única opción legal.

JSP



```
<%@taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<%@taglib prefix="x" uri="http://java.sun.com/jstl/xml" %>
<%@taglib prefix="fmt" uri="http://java.sun.com/jstl/fmt" %>
<%@taglib prefix="sql" uri="http://java.sun.com/jstl/sql" %>
```

```
<h1>Hola mundo!</h1>
<% out.println("Mi primer scriptle"); %>
${ "Esto es un expression languange" }
<c:out value="Usando JSTL" />
```



JSP: Objetos integrados

Objeto	Clase o interfaz
page	java.lang.Object
config	javax.servlet.ServletConfig
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
out	javax.servlet.jsp.JspWriter
application	javax.servlet.ServletContext
pageContext	javax.servlet.jsp.PageContext
exception	java.lang.Throwable
session	javax.servlet.http.HttpSession

JSP

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

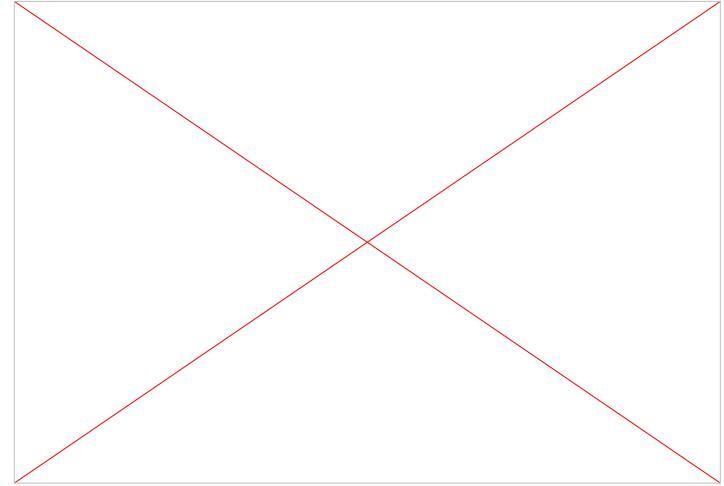
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Redirección con RESPONSE</title>
  </head>
  <body>
    <%response.sendRedirect("/jspTomcat/errorResp.jsp");%>
  </body>
</html>
```

```
<%@page import="java.util.*"%>
<%@page language="Java"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>DESTINO</title>
  </head>
  <body>
    <b><%Date fecha = (Date)request.getAttribute("fecha");
      out.println("<h3 style='font-family:Comic Sans MS text-align:center'>Atributo de la petición" + fecha + "</h3>");%>
    </b>
  </body>
</html>
```

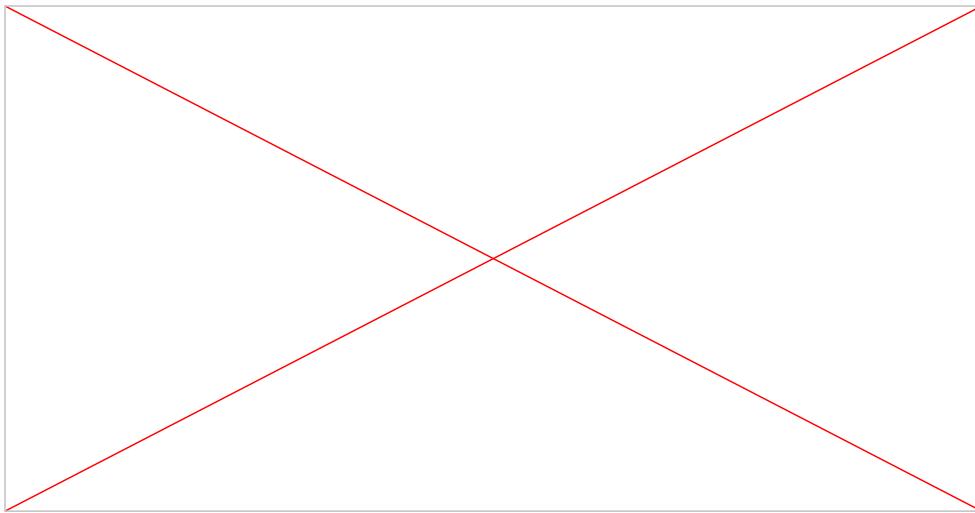
JSP

```
<%@ page isErrorPage="true" import="java.io.*"%>
.....
<body style="font-family:Comic Sans MS" align="justify">
    <h1>Se ha producido una excepción</h1>
    <b>ERROR:</b><%=exception.toString()%><br>
    <b>MENSAJE:</b><%=exception.getMessage()%><br>
    <b>VOLCADO DE PILA:</b>
    <%StringWriter sSalida = new StringWriter();
    PrintWriter salida= new PrintWriter(sSalida);
    exception.printStackTrace(salida);%><%=salida%></p><br>
</body>
</html>
```

```
<%@ page errorPage="paginaError.jsp"%>
.....
<body>
    <%--Creamos un error --%>
    <%int i=1/0;%>
</body>
</html>
```



JSP



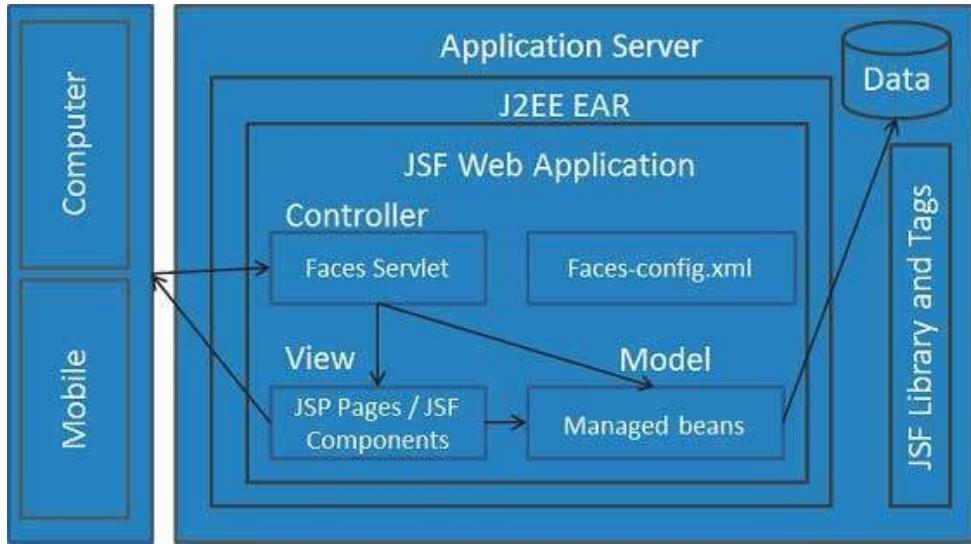


JavaServer Faces es el framework oficial de Java Enterprise para el desarrollo de interfaces de usuario avanzadas en aplicaciones web. La especificación de JSF ha ido evolucionando desde su lanzamiento en 2004 y se ha ido consolidando, introduciendo nuevas características y funcionalidades.

JSF es un framework MVC (Modelo-Vista-Controlador) basado en el API de Servlets que proporciona un conjunto de componentes en forma de etiquetas definidas en páginas XHTML mediante el framework Facelets. Facelets se define en la especificación 2 de JSF como un elemento fundamental de JSF que proporciona características de plantillas y de creación de componentes compuestos.

Es un estándar claro y potente para poder hacer aplicaciones visuales mas potentes.

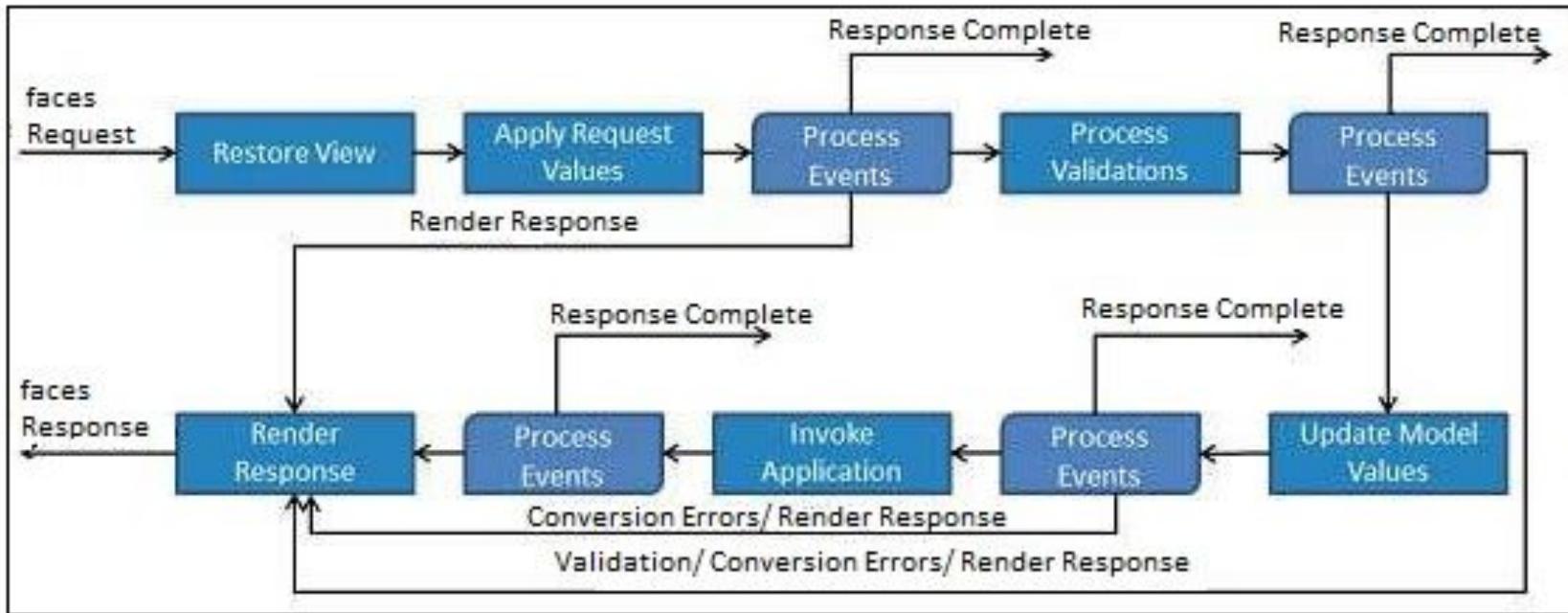
JSF

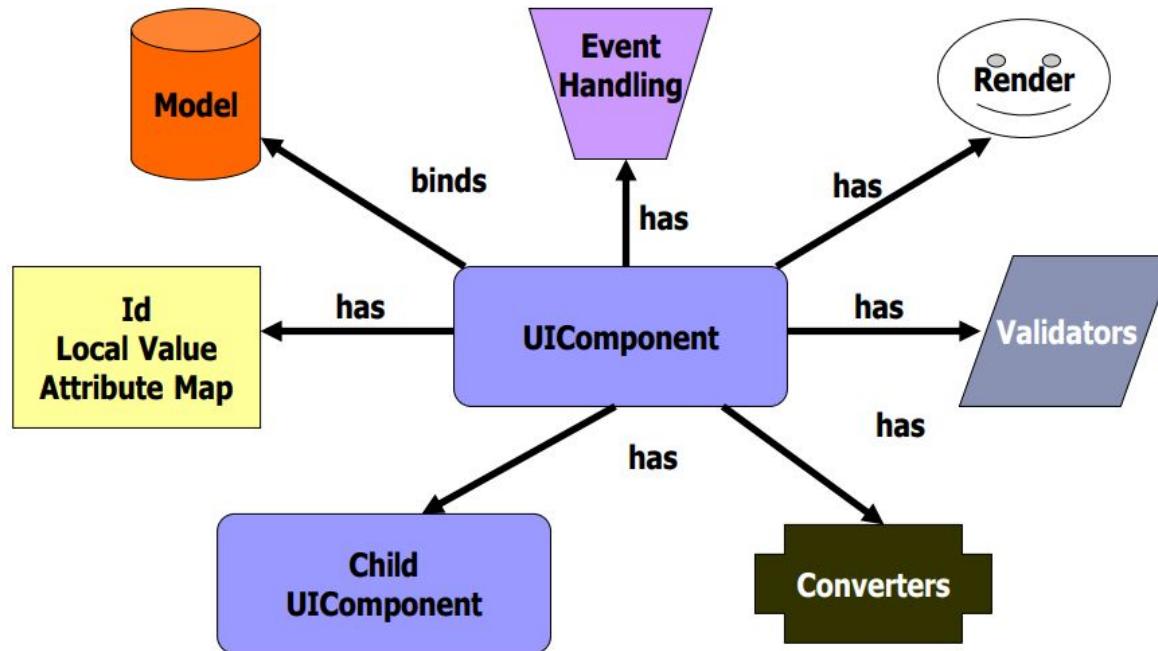


- **Tecnología que simplifica la creación de interfaces de usuario para aplicaciones JavaServer.**
- **Muy baja complejidad.**
- **Permite crear las interfaces a través de componentes reutilizables, los cuales se pueden conectar con cualquier fuente de datos así como también pueden estar asociados a eventos en el servidor**
- **JavaServerFaces (JSF) se rige bajo el ExpertGroup (EG) el cual opera bajo la JCP.**
- **Bajado en MVC.**

- Providing reusable UI components
- Making easy data transfer between UI components
- Managing UI state across multiple server requests
- Enabling implementation of custom components
- Wiring client-side event to server-side application code

JSF





JavaServer Faces Components

Username

Password

#	Item	Quantity	Price
1	Milk	2	€ 1.60
2	Bread	1	€ 2.00
3	Cheese	1	€ 4.75
4	Tomatoes	6	€ 3.80

[Previous](#) - [1](#) [2](#) [3](#) ... [8](#) [9](#) - [Next](#)

JSF

<html xmlns="<http://www.w3.org/1999/xhtml>" xmlns:h="http://xmlns.jcp.org/jsf/html">

Library	Namespace Identifier	Common prefixes
Core	http://xmlns.jcp.org/jsf/core	f
HTML	http://xmlns.jcp.org/jsf/html	h
Facelets	http://xmlns.jcp.org/jsf/facelets	ui
Composite Components	http://xmlns.jcp.org/jsf/composite	composite cc
JSTL Core	http://xmlns.jcp.org/jsp/jstl/core	c
JSTL Functions	http://xmlns.jcp.org/jsp/jstl/functions	fn

[Link](#)

[Link](#)

[Link](#)

[Link](#)

[Link](#)

[Link](#)

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://xmlns.jcp.org/jsf/html">
```

Form	<ul style="list-style-type: none">• form
Navigation Must be in <form> tag	<ul style="list-style-type: none">• h:commandButton• h:commandLink• h:button• h:outputLink• h:link
Input Must be in <form> tag	<ul style="list-style-type: none">• h:inputText• h:inputSecret• h:inputTextarea• h:selectBooleanCheckbox• h:selectOneListbox• h:selectOneRadio• h:selectOneMenu• h:selectManyCheckbox• h:selectManyListbox• h:selectManyMenu
Input Error Messages	<ul style="list-style-type: none">• h:message• h:messages
Display	<ul style="list-style-type: none">• h:outputText• h:outputFormat
Layout and Grouping	<ul style="list-style-type: none">• h:panelGrid• h:panelGroup
Resource Access	<ul style="list-style-type: none">• h:outputStylesheet• h:graphicImage• h:outputScript

JSF Navigation Tags

Table 4–19 h:commandButton and h:button Examples

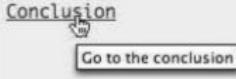
Example	Result
<h:commandButton value="submit" type="submit" action="#{form.submitAction}"/>	<input type="button" value="submit"/>
<h:commandButton value="reset" type="reset"/>	<input type="button" value="reset"/>
<h:commandButton value="click this button..." onclick="alert('button clicked')" type="button"/>	<input type="button" value="click this button to execute JavaScript"/>
<h:commandButton value="disabled" disabled="#{not form.buttonEnabled}"/>	<input type="button" value="disabled"/>
<h:button value="#{form.buttonText}" outcome="#{form.pressMeOutcome}"/>	<input type="button" value="press me"/>

Table 4–20 h:commandLink and h:link Examples

Example	Result
<h:commandLink>register</h:commandLink>	register
<h:commandLink style="font-style: italic">#{msgs.linkText}</h:commandLink>	click here to register
<h:commandLink>#{msgs.linkText} <h:graphicImage value="/registration.jpg"/></h:commandLink>	 click here to register
<h:commandLink value="welcome" actionListener="#{form.useLinkValue}" action="#{form.followLink}"/>	welcome
<h:link value="welcome" outcome="#{form.welcomeOutcome}"><f:param name="id" value="#{form.userId}"/></h:link>	welcome

JSF Navigation Tags

Table 4-22 h:outputLink Examples

Example	Result
<pre><h:outputLink value="http://java.net"> <h:graphicImage value="java-dot-net.jpg"/> <h:outputText value="java.net"/> </h:outputLink></pre>	
<pre><h:outputLink value="#{form.welcomeURL}"> #{form.welcomeLinkText} </h:outputLink></pre>	<u>go to welcome page</u>
<pre><h:outputLink value="#introduction"> <h:outputText value="Introduction" style="font-style: italic"/> </h:outputLink></pre>	<u>Introduction</u>
<pre><h:outputLink value="#conclusion" title="Go to the conclusion"> Conclusion </h:outputLink></pre>	
<pre><h:outputLink value="#toc" title="Go to the table of contents"> <h2>Table of Contents</h2> </h:outputLink></pre>	<u>Table of Contents</u>

JSF Core Tags

- Attributes
- Parameters
- Facets
- Listeners
- Converters
- Validators
- Selection Items

La librería Core contienen los Tags que son independientes del HTML rendering. Muchos de estos tags representan objetos que podemos agregar a nuestros componentes.

JSF Facelets Tags

La librería Facelets contienen los Tags que son usados para el manejo de los layouts y templates.

`ui:include`

`ui:composition`

`ui:insert`

`ui:define`

`ui:params`

`ui:decorate`

page.xhtml

```
<ui:include  
    src="header.xhtml"/>
```

header.xhtml

```
<ui:include  
    src="footer.xhtml"/>
```

footer.xhtml

CABECERA DE NUESTRO PROYECTO

Opcion1
Opcion2
Opcion3

CONTENIDO

PIE DE PAGINA

JSF Facelets Tags

La librería Facelets contienen los Tags que son usados para el manejo de los layouts y templates.

ui:include

- **ui:include**

- Inserts the content of another Facelets XHTML file

- **ui:composition**

- Defines a page that uses a template

- **ui:insert**

- Defines the point inside a template where content is to be inserted

- **ui:define**

- Defines content to be inserted into the template

- **ui:param**

- Passes a parameter to an included file or template

- **ui:decorate**

- Applies a template to part of a page

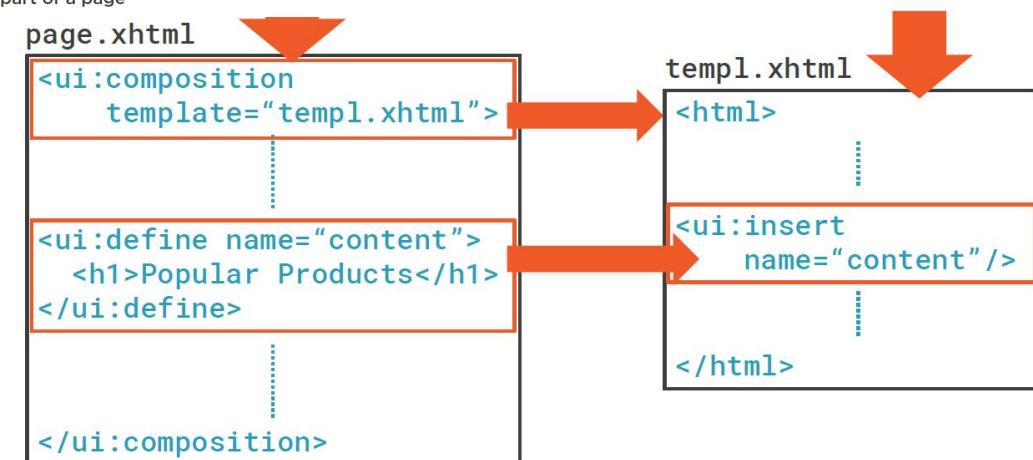
ui:composition

ui:insert

ui:define

ui:params

ui:decorate



- Es la API de persistencia desarrollada para la plataforma Java EE e incluida en el estándar EJB3
- JPA es un Estándar ORM (Object Relational Mapping)
- Disponible para JSE y JEE.
- Definida en **javax.persistence**.
- Definido según el estándar JSR 220 y JSR 317. (Contrato)
- Es la materialización de experiencias de distintas soluciones de ORM para java.
 - Basado en Hibernate y Java Data Object (JDO) entre otros.

Por que usar JPA

- Facilita la persistencia desde el punto de vista del desarrollador.
- Encapsula el uso de SQL, JDBC y otras tecnologías subyacentes.
- Incrementa la respuesta al cambio de las aplicaciones.
- No requiere de un servidor de aplicaciones para su uso.
- Reducción dramática en código de acceso a datos.
- Permite concentrarse en el modelo de objeto y no en el modelo de datos.

- **Entidad:** Objeto con las anotaciones adecuadas para realizar la persistencia a la BBDD.
- **POJO:** Plain Old Java Object
- **Anotación:** Añadir metadatos (datos descriptivos) al código fuente.
- **JPQL:** Java Persistence Query Languaje
- **NamedQuery:** Consultas JPQL
- **NativeQuery:** Consultas en SQL Nativo que se traducen a objetos entidad.



HIBERNATE



eclipse)link



persistence.xml

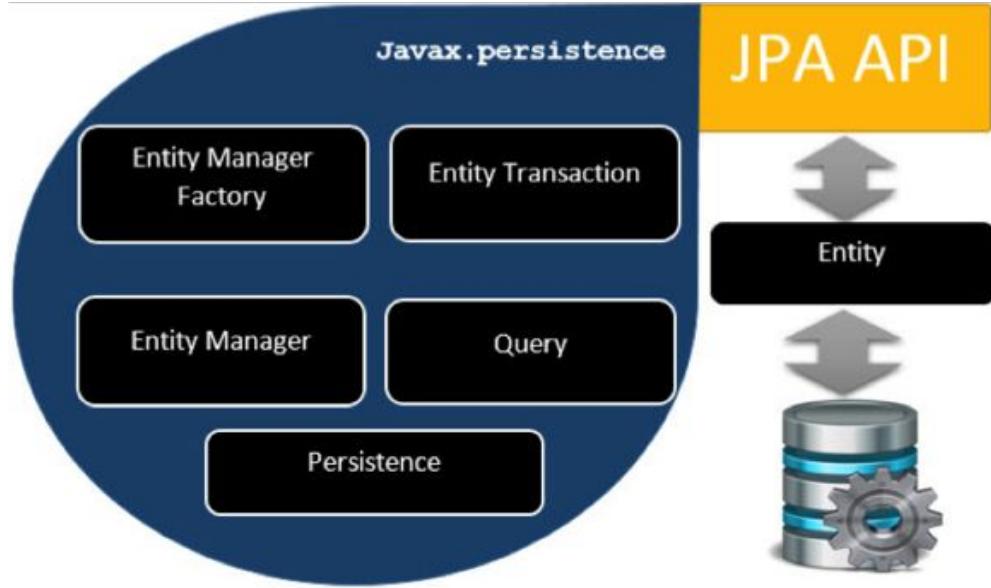
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
    <persistence-unit name="Eclipselink_JPA"
        transaction-type="RESOURCE_LOCAL">
        <class>com.tutorialspoint.eclipselink.entity.Employee</class>
        <properties>
            <property name="javax.persistence.jdbc.url"
                value="jdbc:mysql://localhost:3306/jpadb"/>
            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value="root"/>
            <property name="javax.persistence.jdbc.driver"
                value="com.mysql.jdbc.Driver"/>
            <property name="eclipselink.logging.level" value="FINE"/>
            <property name="eclipselink.ddl-generation"
                value="create-tables"/>
        </properties>
    </persistence-unit>
</persistence>
```

```
public static void main( String[ ] args )
{
    EntityManagerFactory emfactory = Persistence.
        createEntityManagerFactory( "Eclipselink_JPA" );
    EntityManager entitymanager = emfactory.
        createEntityManager( );
    entitymanager.getTransaction( ).begin( );

    Employee employee = new Employee( );
    employee.setEid( 1201 );
    employee.setEname( "Gopal" );
    employee.setSalary( 40000 );
    employee.setDeg( "Technical Manager" );
    entitymanager.persist( employee );
    entitymanager.getTransaction( ).commit( );

    entitymanager.close( );
    emfactory.close( );
}
```

JPA



@OneToMany	Defines a one-to-many relationship between the join Tables.
@OneToOne	Defines a one-to-one relationship between the join Tables.
@NamedQueries	specifies list of named queries.
@NamedQuery	Specifies a Query using static name.

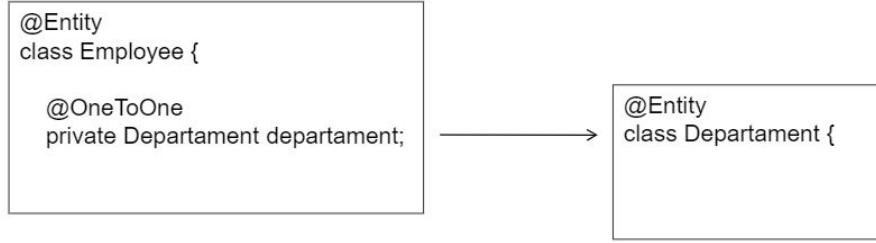
Annotation	Description
@Entity	Declares the class as an entity or a table.
@Table	Declares table name.
@Basic	Specifies non-constraint fields explicitly.
@Embedded	Specifies the properties of class or an entity whose value is an instance of an embeddable class.
@Id	Specifies the property, use for identity (primary key of a table) of the class.
@GeneratedValue	Specifies how the identity attribute can be initialized such as automatic, manual, or value taken from a sequence table.
@Transient	Specifies the property that is not persistent, i.e., the value is never stored in the database.
@Column	Specifies the column attribute for the persistence property.
@SequenceGenerator	Specifies the value for the property that is specified in the @GeneratedValue annotation. It creates a sequence.
@TableGenerator	Specifies the value generator for the property specified in the @GeneratedValue annotation. It creates a table for value generation.
@AccessType	This type of annotation is used to set the access type. If you set @AccessType(FIELD), then access occurs Field wise. If you set @AccessType(PROPERTY), then access occurs Property wise.
@JoinColumn	Specifies an entity association or entity collection. This is used in many-to-one and one-to-many associations.
@UniqueConstraint	Specifies the fields and the unique constraints for the primary or the secondary table.
@ColumnResult	References the name of a column in the SQL query using select clause.
@ManyToMany	Defines a many-to-many relationship between the join Tables.
@ManyToOne	Defines a many-to-one relationship between the join Tables.

- ❑ Se anotan con @Entity
- ❑ Tienen una propiedad anotada con @Id
- ❑ Constructor sin argumentos public/protected
- ❑ No puede ser final
- ❑ Puede extender de otra
- ❑ Puede ser abstracta
- ❑ **Es un POJO (Plain Old Java Objects), objeto liviano que no implementan ninguna interfaz**

```
@Entity  
@Table(name = "student")  
public class Student {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    @Column(name = "id")  
    private int id;  
  
    @Column(name = "first_name")  
    private String firstName;  
  
    @Transient // Field will not be saved in a database  
    private String lastName;  
    // getter and setter methods  
}
```

JPA

- ❑ Relaciones Unidireccionales
 - Sólo se puede navegar desde una entidad a la otra
- ❑ Relaciones Bidireccionales
 - Desde cualquiera de las dos entidades es posible navegar a la otra

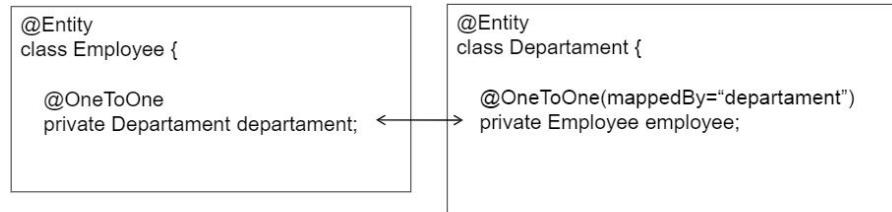


EMPLOYEE		
ID	DEPARTAMENT	...
X	1234	

DEPARTAMENT	
ID	...
	1234

FK to Departament.ID

- ❑ **one-to-one**
- ❑ **one-to-many**
- ❑ **many-to-one**
- ❑ **many-to-many**



EMPLOYEE		
ID	DEPARTAMENT	...
X	1234	

DEPARTAMENT	
ID	...
	1234

FK to Departament.ID

JPA

```
@Entity  
class Project {  
  
    @OneToMany  
    private Collection<Employee> employees  
  
    ...}
```

PROJECT

ID	...	
555		



```
@Entity  
class Employee {  
    ...}
```

EMPLOYEE

ID	...	
11111		
22222		

PROJECT_ID	EMPLOYEE_ID
555	11111
555	22222

```
@Entity  
class Project {  
  
    @OneToMany  
    private Collection<Employee> employees  
  
    ...}
```



```
@Entity  
class Employee {  
  
    @ManyToMany  
    private Collection<Project> projects  
  
    ...}
```



PROJECT

ID	...	
555		

EMPLOYEE

ID	PROJECT	...
11111	555	
22222	555	

```
@Entity  
class Project {  
  
    @ManyToMany(mappedBy="projects")  
    private Collection<Employee> employees  
  
    ...}
```

```
@Entity  
class Employee {  
  
    @ManyToMany  
    private Collection<Project> projects  
  
    ...}
```

PROJECT

ID	...	
555		

EMPLOYEE

ID	...	
11111		
22222		

PROJECT_ID	EMPLOYEE_ID
555	11111
555	22222

JPA

```
public String getPersonName(long personId) throws SQLException {
    PreparedStatement st = null;
    try {
        st = connection.prepareStatement(
            "SELECT name FROM people WHERE id = ?");
        st.setLong(1, personId);
        ResultSet rs = st.executeQuery();
        if (rs.next()) {
            String result = rs.getString("name");
            assert !rs.next();
            return result;
        } else {
            return null;
        }
    } finally {
        if (st != null) { st.close(); }
    }
}
```

JDBC

```
public String getPersonName(long personId) {
    Person p = em.find(Person.class, personId);
    return p.getName();
}
```

JPA

EJB - Managed Bean

Los managed bean son clases java que se registran en el framework de JSF, lo que permite que puedan ser consumidos desde las páginas dinámicas. Desde la versión 2.0 de JSF este registro se puede llevar a cabo mediante la anotación `@javax.faces.bean.ManagedBean`.

A partir de JSF 2.3, `@ManagedBean` es **deprecado**. Por ende ya no existen razones del porque usar `@ManagedBean` por sobre `@Named`.

Un Managed Bean es un **Java Bean** que puede ser accedido desde una página JSF, cada Managed Bean debe tener un *nombre* y un *ámbito*.

```
@Named(value = "helloBean")
@ViewScoped
public class HelloBean implements Serializable {

    /**
     * Creates a new instance of HelloBean
     */
    public HelloBean() {
    }

    public String getMessage() {
        return "Hola Bienvenido al Sistema";
    }
}
```

EJB - Managed Bean

Producto

id

nombre

descripcion

image

EJB - Managed Bean

@RequestScoped ámbito de petición, al terminar la petición se elimina la instancia del bean.

@SessionScoped ámbito de sesión, mientras la sesión exista existe el bean.

@ViewScoped: entre request y session, el bean existirá mientras la petición se envíe a la misma vista.

@ApplicationScoped ámbito de la aplicación web, mientras la aplicación se ejecute el bean existirá.

EJB