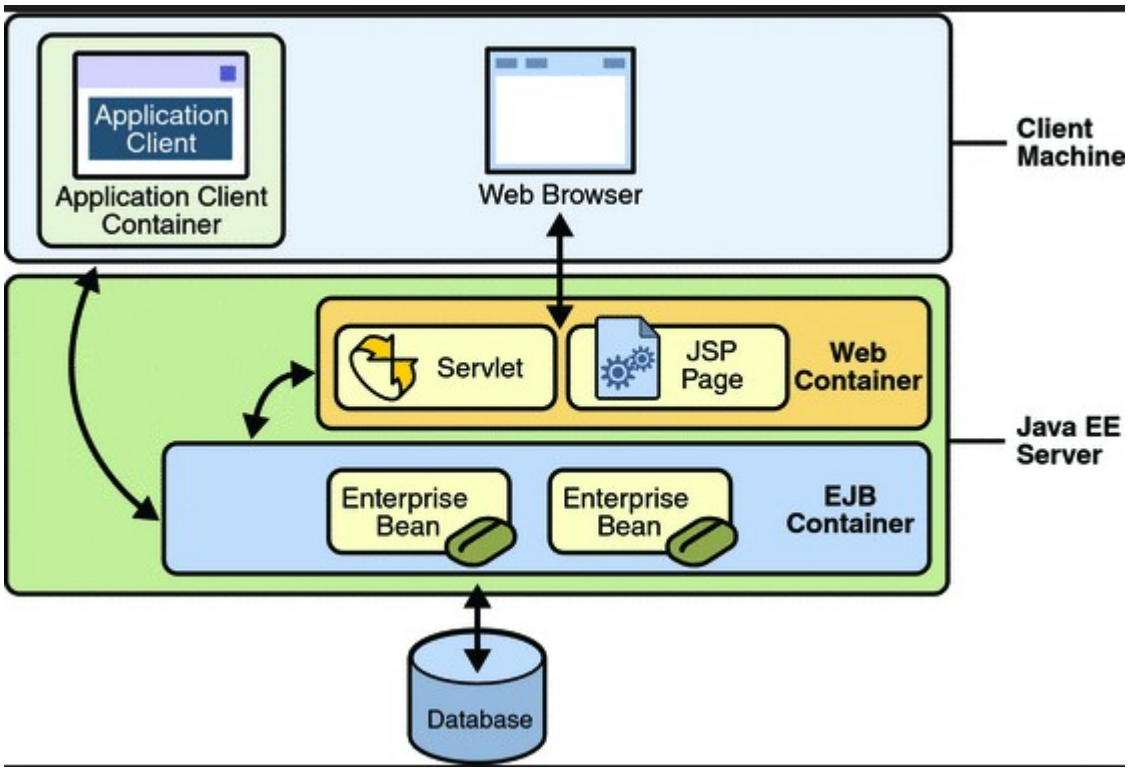
# SERVLETS, JAVA SERVER PAGES Y ENTERPRISE JAVABEANS

**Autora:**

**Alexandra Mora White**



# ***SERVLETS, JAVA SERVER PAGES Y ENTERPRISE JAVABEANS:***

## ➔ Servlets:

Los Sevlets son módulos Java que nos sirven para extender las capacidades de respuesta a los clientes de los servidores, que pueden o no ser servidores web. Aunque pueden responder a cualquier tipo de solicitudes, son utilizados comúnmente para extender las aplicaciones alojadas por servidores web. Este tipo de servlets son la contra parte Java de otras tecnologías de contenido dinámico Web, como PHP y ASP.NET.

-El ciclo de vida de un servlet es:

- **Inicialización del servlet:** Cuando un servidor carga un servlet, ejecutará un método de inicio, llamado, ***init.***

- **Interacción con los clientes:** Una vez inicializado queda a la espera de peticiones. En los Servlets sólo existe una copia cargada en la máquina virtual y por cada petición se inicia un hilo, lo cual reduce el uso de memoria del servidor y el tiempo de respuesta. Dependiendo de como lleguen los datos (mediante *post* o *get*) al servlet se ejecutará un método *doPost* o *doGet*.

- **Destruir el servlet:** Los servlets se ejecutan hasta que el servidor los destruye, por cierre o a petición del administrador. Ejecutando el método ***destroy***. Este método sólo se ejecuta una vez y puede ser llamado cuando aún queden respuestas en proceso, por lo que hay que tener cuidado y asegurarse de que todos los procesos han finalizado.

Se consideran un reemplazo efectivo para los CGI ya que proporcionan una forma de generar documentos dinámicos utilizando las ventajas de la programación en Java como conexión a bases de datos, manejo de peticiones concurrentes, programación distribuida, etc.

### ➢ Para que se utilizan?

Podremos desarrollar desde un simple servlet que nos muestre una página web saludándonos hasta uno que se conecte a una base de datos utilizando un pool de conexiones, encriptando la información en su envío, accediendo a bases de datos distribuidas y manteniendo su información de forma persistente en un EJB. Todo ello para conseguir una información dinámica.

- **Funcionalidad:**

1. Leer los datos enviados por el cliente.
2. Extraer cualquier información útil incluida en la cabecera HTTP o en el cuerpo del mensaje de petición enviado por el cliente.
3. Generar dinámicamente resultados.
4. Formatear los resultados en un documento HTML.
5. Establecer los parámetros HTTP adecuados incluidos en la cabecera de la respuesta (por ejemplo: el tipo de documento, cookies, etc.)
6. Enviar el documento final al cliente.

- **Ventajas:**

- Es integrable con páginas JSP y se pueden usar filtros.
- Instancia permanentemente cargada en memoria por cada Servlet.
- Cada petición se ejecuta en un hilo, no en un proceso.
- Con los Servlets, hay *n* threads pero sólo una copia de la clase Servlet.
- Tienen más alternativas que los programas CGI para optimizaciones como mantener abiertas las conexiones de las bases de datos.
- Están escritos en Java y siguen un API bien estandarizado.
- Pueden ser utilizados sobre cualquier SO y la mayoría de servidores Web.
- Por ser Java ofrece: máquina virtual, chequeo de tipos, gestión de memoria, excepciones, seguridad, etc.
- Actúan como capa intermedia entre la petición que proviene del navegador Web u otro cliente HTTP y Bases de Datos o Aplicaciones en el servidor HTTP.

## ➔ JSP- Java Server Pages:

Las páginas JSP surgen con la idea de facilitar la creación de contenido dinámico a desarrolladores sin necesidad de conocer a fondo el lenguaje Java. Es una tecnología que ayuda a los desarrolladores de software a crear páginas web dinámicas basadas en HTML, XML, entre otros tipos de documentos. Es similar a PHP, pero usa el lenguaje de programación Java.

Puede instanciar clases, hacer llamadas a otras páginas JSP, Servlets e incluir JavaBeans y applets.

Es la capa de presentación, separándose así del negocio y el control de la aplicación. Cuando una página es solicitada por un usuario y procesada por un servidor HTTP, el código HTML pasará directamente al usuario, mientras que las porciones de código Java serán ejecutadas en el servidor para generar el contenido dinámico de la página. Cuando el usuario acceda al código de la página que le llega sólo verá HTML, sin poder acceder al código JSP subyacente.

Para pasar datos de un JSP a otro o bien a un Servlet utilizaremos las Request mediante el uso de formularios.

Para desplegar y correr JavaServer Pages, se requiere un servidor web compatible con contenedores servlet como *Apache Tomcat* o *Jetty.*

Con JSP podemos crear aplicaciones web que se ejecuten en servidores web, de múltiples plataformas.

### ➢ Beneficios:

- Las peticiones de páginas JSP son implementadas mediante servlets, de forma que el contenedor JSP, maneja múltiples solicitudes a la vez, requiriendo menor overhead, y menos recursos.
- Componentes reutilizables: característica derivada de la orientación a objetos de Java. JSP permite implementar contenido dinámico incluyendo código Java directamente en la página. También ofrece una serie de etiquetas que le permiten actuar sobre objetos Java residentes en el servidor.
- Separación de presentación e implementación: La implementación del programa puede ser llevada a cabo por los objetos Java, así podemos separar lo que es la presentación y el código encargado de generar la información necesaria que aparecerá en la página.
- División de labor: Separación de presentación e implementación . Por lo que alguien que no sepa Java le bastará conocer las propiedades que le ofrece un conjunto de objetos y podrá encargarse de la parte de la página relacionada con la presentación y un programador Java se encargaría de crear el código que generará la información dinámica,

despreocupándose de los problemas de presentación de la página.

- **Como funciona JSP:**

Las solicitudes del navegador llegan al servidor HTTP y las páginas JSP son enviadas al servlet compilador de páginas que corre en el contenedor JSP. Si el servlet para la página actual está actualizado lo genera y compila, cargándolo en el contenedor servlet. En caso contrario el control es transferido al servlet de la página JSP que se encarga de manejar la solicitud generando la respuesta y enviándola al servidor HTTP el cual la remitirá al navegador.

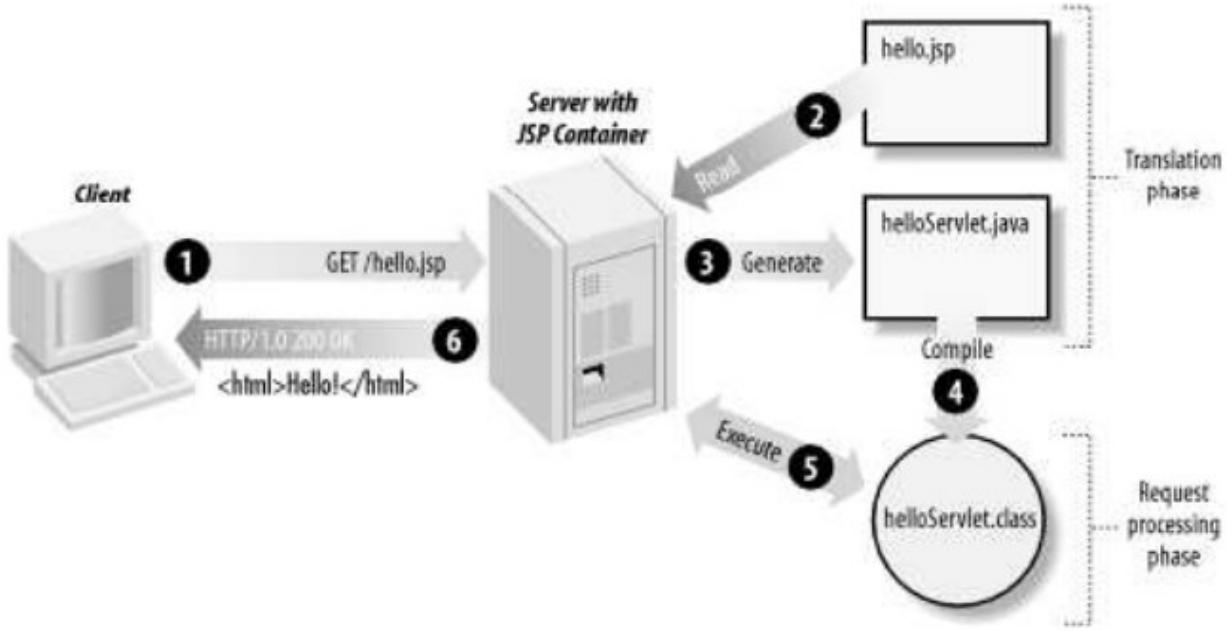



**Fig. Ejecución en el modelo web cliente-servidor**

- **Ejemplo sencillo de código JSP:**

```
<%@ page language="java" contentType="text/html" %>

<html>
   <head>
         <title>Hola, mundo!!</title>
   </head>
   <body>
         <h1>Hola, mundo!</h1>
         Hoy es <%= new java.util.Date() %>.
   </body>
</html>
```

## ➔ EJB- Enterprise JavaBeans:

Un EJB es un componente del lado del servidor que encapsula la lógica del negocio de una aplicación. En cualquier aplicación, los beans enterprise implementan los métodos de la lógica del negocio, que pueden ser invocados por clientes remotos para acceder a los servicios importantes proporcionados por la aplicación.
- Los servicios más importantes que ofrece EJB son:

- **Manejo de transacciones:** apertura/cierre de transacciones asociadas a las llamadas a los métodos del bean.
- **Seguridad:** comprobación de permisos de acceso a los métodos del bean.
- **Concurrencia:** llamada simultánea a un mismo bean desde múltiples clientes.
- **Servicios de red:** comunicación entre el cliente y el bean en máquinas distintas.
- **Gestión de recursos:** gestión automática de múltiples recursos, como colas de mensajes, bases de datos en aplicaciones heredadas que no han sido traducidas a nuevos lenguajes/entornos y siguen usándose en la empresa.
- **Persistencia:** sincronización entre los datos del bean y tablas de una BBDD.
- **Gestión de mensajes:** manejo de *Java Message Service* (*JMS*).
- **Escalabilidad:** si tenemos un número creciente de usuarios, los EJBs nos permitirán distribuir los componentes de nuestra aplicación entre varias máquinas con su localización transparente para los usuarios.
- **Adaptación en tiempo de despliegue:** posibilidad de modificación de todas estas características en el momento del despliegue del bean.

### ➢ Funcionamiento:

El funcionamiento de los componentes EJB se basa fundamentalmente en el trabajo del contenedor EJB. El contenedor EJB es un programa Java que corre en el servidor y que contiene todas las clases y objetos necesarios para el correcto funcionamiento de los enterprise beans.
La figura siguiente muestra una representación de muy alto nivel del funcionamiento básico de los enterprise beans.

1. El cliente realiza peticiones al bean y el servidor que contiene el bean están ejecutándose en máquinas virtuales Java distintas (incluso pueden estar en distintos hosts). El cliente nunca se comunica directamente con el enterprise bean.

2. Sino que el contenedor EJB proporciona un EJBObject que hace de interfaz. Cualquier petición del cliente se debe hacer a través del objeto EJB, el cual solicita al contenedor EJB una serie de servicios y se comunica con el enterprise bean.

3. El bean realiza las peticiones correspondientes a la base de datos.

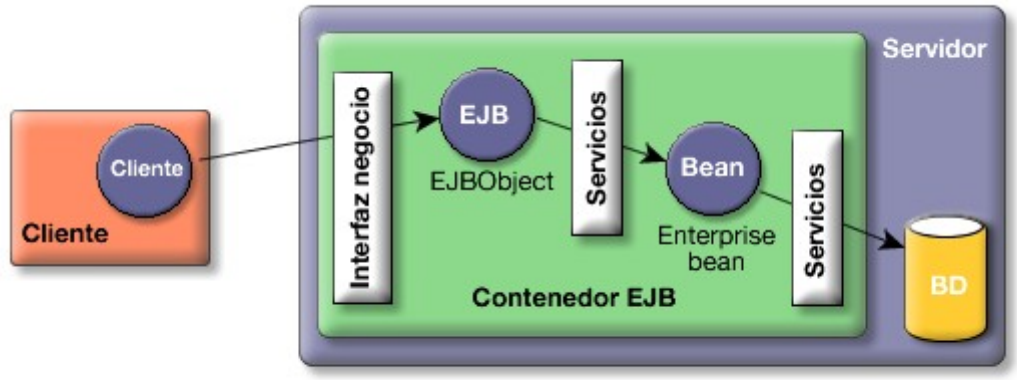



*Figura 1.1: representación de alto nivel del funcionamiento de los enterprise beans.*

## ➢ Tipos de beans

- **Beans de entidad:** representan un objeto concreto que tiene existencia en alguna base de datos de la empresa. Una instancia de un bean de entidad representa una fila en una tabla de la base de datos.
  - Se distinguen dos tipos de beans de entidad:

- Bean Managed Persistence: requiere que la lógica necesaria para acceder a la base de datos se defina manualmente, por lo general esta lógica se encuentra en JDBC y define: como y cuando debe ser accedida/actualizada/guardada la información entre el EJB y la BBDD.
- Container Managed Persistence: es manejado por el "EJB Container", en un "Container Managed EJB" el "EJB Container" genera toda lógica de acceso para el sistema de información ( base de datos ).
  Aunque en una primera estancia, parezca que un "Bean Managed EJB" no tiene mucha razón de ser, hay casos donde se utiliza lógica de acceso sumamente compleja la cual no es posible generar a través del "EJB Container", por eso los "Bean Managed EJB's" son necesarios a pesar de las facilidades ofrecidas por "Container Managed EJB's"

- **Beans dirigidos por mensajes :** únicos beans con funcionamiento asíncrono. Usando el *Java Messaging System (JMS)*, se suscriben a un tema (topic) o cola (queue) y se activan al recibir un mensaje dirigido a dicho tema o cola. No requieren de su instanciación por parte del cliente.

- **Bean de sesión:** representa un proceso o una acción de negocio. Generalmente, cualquier llamada a un servicio del servidor debería comenzar con una llamada a un bean de sesión. Mientras que un bean de entidad representa una cosa que se puede representar con un nombre, un bean de sesión se podría representar como un verbo.
  - Existen dos tipos de Beans de sesión:

- Stateless (Session) EJB's (Sin estado): son utilizados para realizar tareas rutinarias que no requieren identificar o rastrear al cliente, algunos EJB's de este tipo son: operaciones matemáticas complejas, búsquedas generales, etc.

- Statefull (Session) EJB's (Con estado): permiten mantener la sesión del cliente en el "EJB Container", de esta manera el cliente puede trabajar con cierto juego de datos especifico administrado por el "EJB Container", la aplicación ideal para este tipo de EJB es un componente de compra ("Shopping Cart") el cual puede identificar artículos e información personal del cliente a través de un lapso de tiempo extenso ("Session").

➢ **Ventajas:**

- **Servicios ("Middleware"):** Cuando se diseña un componente de Software se deben definir varios servicios para su funcionamiento, como que procedimiento ejecutar si existe un error, si la base de datos se encuentra desactivada, cuál es la alternativa, etc. Mediante un "EJB Container" se ofrecen estos servicios y es a través de un "Enterprise Java Bean" que es posible desarrollar los componentes principales.

- **División de Trabajo:** La posibilidad de dividir "Servicios"(EJB Container) de "Componentes Principales"(EJB'S) permite una clara división de trabajo. Un diseñador de "componentes"(EJB's) puede concentrar sus esfuerzos en la "lógica de proceso" sin preocuparse del diseño de servicios. De la misma manera un "diseñador" de servicios puede concentrarse en su área.
- **Diversos Vendedores:** El uso de especificaciones para EJB's permite que existan diversos vendedores tanto de "EJB Containers" los cuales son incluidos en un java application server , así como "Enterprise Java Bean's" los cuales resuelven algún tipo de lógica.

- **Procedimientos Remotos ( RMI )**

- **Diversos Clientes:** Un EJB puede interactuar con una gran gamma de clientes desde: JSP o Servlets , bases de datos , Applets , sistemas ERP (SAP,JDEdward's).

- **CONCLUSIONES:**

**¿Para que tipos de aplicaciones web es mejor cada uno de ellas? ¿es indiferente? ¿porque? ¿que características diferenciadoras tiene cada uno?**

Estas son mis conclusiones tras investigar y leer diferentes definiciones y explicaciones en las urls informadas, al no conocer previamente que eran los servlets, JSP y EJB es posible que no lo haya entendido bien ,aunque espero que sí.

Respecto al uso dependiendo del tipo de aplicaciones web, entiendo que tanto en el caso de Servlets como JSP es indiferente, ya que ambas ejecutan peticiones de clientes, independientemente del tipo de solicitud que sean.
Sin embargo, debido a los servicios que ofrece EJB, creo que debería usarse en aplicaciones de empresas que requieran una seguridad, persistencia de datos y concurrencia de los mismos.

A primera vista, los EJBs y los Servlets son tecnologías similares, ya que ambos son componentes distribuidos del lado del servidor.
Sin embargo, según he leido, los EJBs no pueden aceptar peticiones HTTP.
Por lo que los EJBs no pueden servir peticiones que vienen directamente desde un navegador Web, y los servlets sí.
Servlets y JSPs se pueden usar para implementar presentación y control web, pero , a diferencia de los EJBs, no pueden manejar transacciones distribuidas.

Servlets y JSP interactúan en cierto modo con el cliente, recibiendo las peticiones , y comunicándose con la Base de datos, en ocasiones a través de EJB (aunque no necesariamente). JSP al igual que PHP, se implementa dentro del código HTML, para crear páginas web dinámicas, mostrando únicamente al cliente la información HTML pero enviando al servidor el código JSP. Entiendo que JSP tiene servlets propios, aunque también utiliza Servlets que ya han sido inicializados, cómo apoyo o porque ya disponen de la respuesta, al haber realizado el hilo anteriormente, por lo que no necesitan volver a solicitar la información al Servidor, al disponer de ella.

Por lo que he entendido EJB , estaría más en la capa de persistencia o de datos, ya que es el que se comunica con la BBDD y controla quien accede a la información. Aunque según el tipo de petición, no considero necesario su uso, sólo si se accede a una base de datos de empresa o corporación que requiere un manejo y seguridad de sus datos.

Esto es lo que he entendido tras realizar el trabajo, si hay algo que no sea correcto me dices.

- **BIBLIOGRAFÍA:**
- **Fuentes de información y paginas visitadas:**

www.inf-cr.uclm.es/www/mpolo/asig/0708/tutorJavaWeb.pdf
https://users.dcc.uchile.cl/~jbarrios/servlets/general.html
https://es.wikipedia.org/wiki/Java_Servlet
http://www.manualweb.net/java-ee/introduccion-a-los-servlets/
http://desarrolloweb.com/articulos/831.php
https://es.wikipedia.org/wiki/Enterprise_JavaBeans
http://www.arquitecturajava.com/introduccion-a-ejb-3-1-i/
http://www.oracle.com/technetwork/java/javaee/ejb/index.html
http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/ejb/sesion01-apuntes.htm
https://www.osmosislatina.com/java/ejb.htm
http://www.lab.inf.uc3m.es/~a0080802/RAI/servlet.html
https://es.wikipedia.org/wiki/JavaServer_Pages