# Autor: Héctor García González

# Servlets,

# JSP (Java Server Pages)

# y EJB (Enterprise JavaBeans)

|  |
| --- |
| **SERVLETS, JSP (Java Server Pages) y EJB (Enterprise JavaBeans)**  **¿QUE SON?, ¿PARA QUE SE UTILIZAN?, CARACTERÍSTICAS DE CADA UNO** |

|  |
| --- |
| **SERVLETS** |

**1-¿Qué es SERVLETS?**

Servlet es una tecnología consistente en una clases en lenguaje Java usada para ampliar la funcionalidad o capacidades de los servidores web a los que se accede vía modelo de programación request-response (petición-respuesta).  Estas clases Java forman un componente Web que se carga dinámicamente dentro de un contenedor o servidor web y genera contenido dinámico. Con esto se crean programas para aplicaciones web.

Su **funcionamiento o ciclo de vida** sería el siguiente

**1ºPaso**: Lo datos son enviados por el cliente al servidor y el servidor inicializa el Servlet que lee los datos enviados por el cliente extrayendo cualquier información útil incluida en la cabecera HTTP o en el cuerpo del mensaje de petición enviado por el cliente.

**2ºPaso**: El Servlets manejará la petición del cliente más las posibles siguientes peticiones del cliente para luego responderle generando dinámicamente resultados y formateándolos en un documento HTML y a su vez estableciendo los parámetros HTTP adecuados incluidos en la cabecera de la respuesta (por ejemplo: el tipo de documento, cookies, etc.). Más tarde envía la respuesta al cliente.

**3ºPaso**: El servidor elimina el Servlet.

**Servlet tiene una serie de características o ventajas tales como:**

- Están escritos en lenguaje Java y siguen un API bien estandarizado.

- Son independientes de la plataforma,

- Consumen menos recursos al usar hilos de ejecución y sólo cargan una vez. Cada petición se ejecuta en un hilo, no en un proceso.

- Son portables,

- Se ejecutan bajo la misma máquina virtual de java y no requiere soporte para java en el explorador del cliente, ya que operan en el ámbito del servidor y envía los resultados en HTML.

- Con los Servlet, hay un número indeterminado de threads pero sólo una copia de la clase Servlet. Los Servlet también tienen más alternativas que los programas CGI para optimizaciones como mantener abiertas las conexiones de las bases de datos.

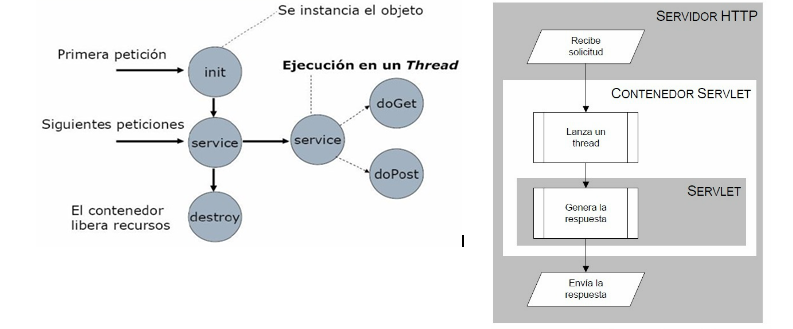
- Puede ser utilizados sobre cualquier SO y en la mayoría de servidores Web.

- 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.

Un **ejemplo práctico de uso con tres capas** sería un Servlet que toma los datos de un formulario HTML del cliente **(capa de usuario)** y aplica la lógica de negocio **(capa de negocio)** para actualizar la base de datos **(capa de persistencia)** distribuida de una empresa  y manteniendo su información de forma persistente en un EJB. A la vez también envía una respuesta al cliente.

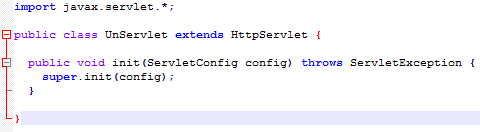
**2-Ciclo de vida del Servlet**

****

**1ºFASE**: La primera vez que realicemos una petición (un formulario HTML del cliente) sobre el Servlet se ejecutará un método de inicio, denominado [init](http://www.w3api.com/wiki/Java:Servlet.init()" \o "init()), en el cual inicializaremos las variables generales. La inicialización se completa antes de manejar peticiones de clientes y antes de que el Servlet sea destruido.

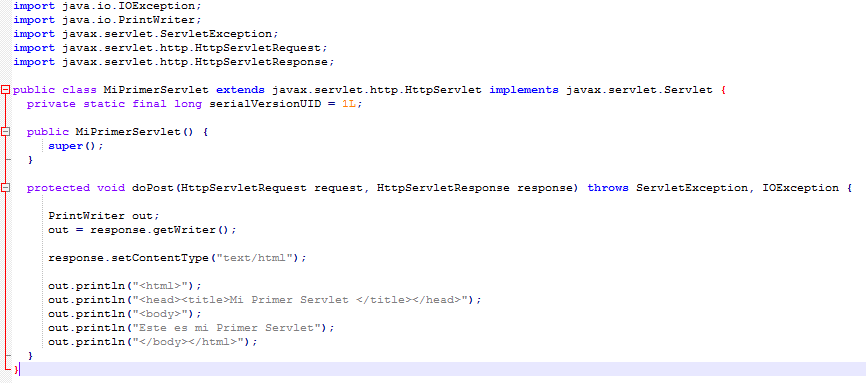
Una vez que nos hemos inicializado el Servlet a la escucha en espera de peticiones.  El servidor sólo llamará una vez al método init al crear la instancia del Servlet, y no lo llamará de nuevo a menos que vuelva a recargar el Servlet. El servidor no puede recargar un Servlet sin primero haber destruido el Servlet llamando al método destroy.

Ejemplo del **método init**:

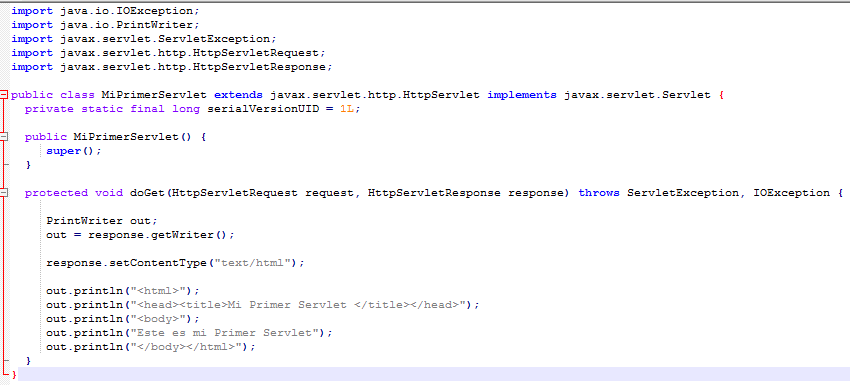


**2ºFASE**: Después de la inicialización, el Servlet puede manejar las siguientes peticiones del cliente. Cada una de las siguientes peticiones que recibamos serán atendidas en hilos de ejecución diferentes, a no ser que indiquemos lo contrario por posibles problemas de sincronización. Dependiendo de cómo lleguen los datos (mediante post o get) al Servlet se ejecutará un método para tratar los datos: [doPost](http://w3api.com/wiki/Java:HttpServlet.doPost()" \o "doPost()) o [doGet](http://www.w3api.com/wiki/Java:HttpServlet.doGet()" \o "doGet()).

Ejemplo del **método** [**doPost**](http://w3api.com/wiki/Java:HttpServlet.doPost())**:**



Ejemplo del **método** [**doGet**](http://www.w3api.com/wiki/Java:HttpServlet.doGet())**:**



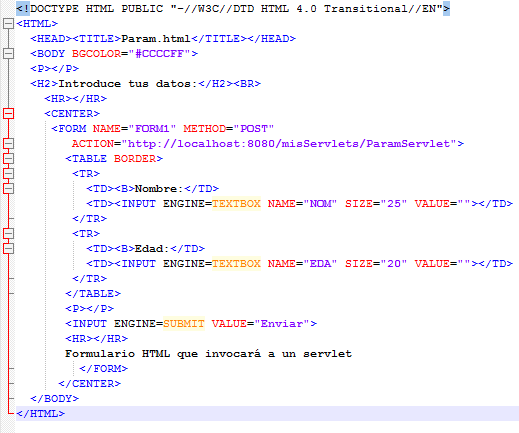
**3ºFASE**: Los Servlet se ejecutan hasta que el servidor los destruye, por cierre el servidor o bien a petición del administrador del sistema. Cuando un servidor destruye un Servlet, ejecuta el método destroy del propio Servlet eliminando las variables creadas en su inicialización, conexiones a bases de datos, etc. Este método sólo se ejecuta una vez y puede ser llamado cuando aún queden respuestas en proceso por lo que hay que tener la atención de esperarlas. El servidor no ejecutará de nuevo el Servlet, hasta haberlo cargado e inicializado de nuevo.

Ejemplo del **método destroy:**

****

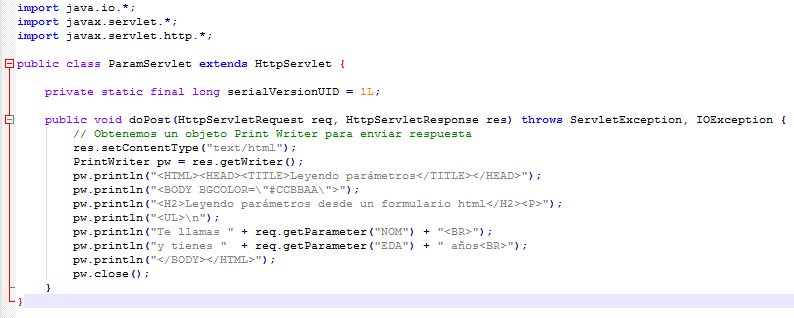
**3-Funcionamiento**

1-Los datos son enviados por el cliente y son recibidos por el Servlet como también los datos embebidos en la propia petición HTTP (cookies, host, etc ).



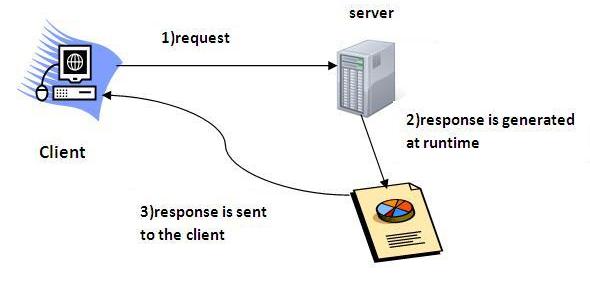
2-Cuando el Servlet acepta la petición del cliente recibe dos objetos o interfaces definidos en el paquete javax.servelt :

\*ServletRequest (encapsula la comunicación del cliente al servidor) \*ServletResponse(encapsula la comunicación del Servlet al cliente)



-El Servlet genera los resultados accediendo a una base de datos o no. La respuesta al cliente puede ser una página HTML, un comprimido ZIP, etc. Luego establece los parámetros adecuados de la respuesta.

-Envía la respuesta al cliente.

****

**Fuentes:**

<http://www.manualweb.net/java-ee/introduccion-a-los-servlets/>

<http://beginnersbook.com/2013/05/servlet-life-cycle/>

<https://users.dcc.uchile.cl/~lmateu/CC60H/Trabajos/jbarrios/vida.html>

<http://www.javatpoint.com/servlet-tutorial>

<https://www.ntu.edu.sg/home/ehchua/programming/java/JavaServlets.html>

<http://chuwiki.chuidiang.org/index.php?title=Ejemplo_sencillo_de_Servlet>

<http://slideplayer.es/slide/1613234/>

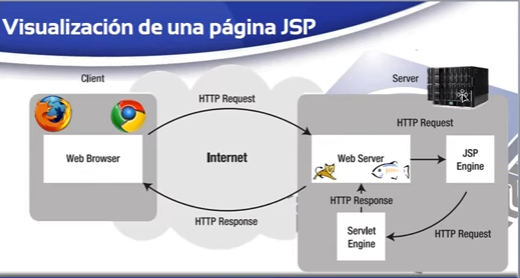
<https://jtagua.wordpress.com/2010/10/31/tutorial-de-servlet-1-introduccion-ciclo-de-vida-y-ejemplo-basico/>

<http://www.lab.inf.uc3m.es/~a0080802/RAI/servlet.html>

|  |
| --- |
| **JSP** |

**1-¿Qué es JSP?**

JSP (Java Server Pages o páginas servidor Java), es una tecnología orientada a crear páginas web con programación en Java. Es una tecnología que permite mezclar HTML estático con HTML generado dinámicamente a través de etiquetas especiales (se trata de un tipo especial de páginas HTML en las cuales se inserta programas o scripts que están en internet). Además de código HTML y fragmentos de código Java, una página JSP puede instanciar clases, hacer llamadas a otras páginas JSP, Servlet e incluir JavaBeans y applets.



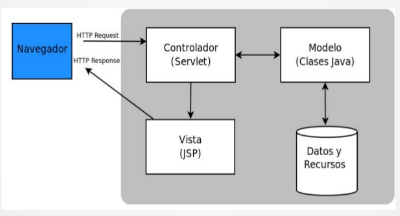
Con JSP generamos archivos con extensión JSP que se incluyen dentro de la estructura de las etiquetas HTML. Todas las páginas JSP deben tener la extensión JSP para que el servidor pueda interpretarlo.



JSP tiene una serie de elementos: **Directivas** (indican información general de la página como puede ser la importación de clases, páginas que forman parte de una sesión, etc), **declaraciones** (sirven para declarar métodos o variables), **scriplets** (código Java embebido), **expresiones** (expresiones Java que se evalúan y se envían a la salida), **tags JSP** (etiquetas especiales que interpreta el servidor).

Para poder utilizarlo es necesario un servidor que de soporte a páginas HTML y código que de soporte a páginas HTML y código que implemente un contenedor JSP donde se ejecuten etiquetas JSP. Sin embargo para la mayoría de los servidores es necesario añadir código suplementario al contenedor JSP (para ello es necesario APIS). Es importante que ese servidor sea compatible con JSP.

La idea es que un JSP sea la capa de presentación, separándose así del negocio y el control de la aplicación.



**JSP tiene una serie de características o ventajas tales como:**

-Mejora en el rendimiento.

-Independencia del servidor.

-Mejor para dividir el trabajo en grupos, unos HTML y otros Java.

- Separación del contenido estático del dinámico.

-Utilización de hilos java para el manejo de las peticiones.

-Manejo de múltiples sobre una página .jsp en un instante

-El contenedor Servlet puede ser ejecutado como parte del servidor web.

-Facilidad para compartir recursos entre peticiones (hilos con el mismo Servlet container).

- Es más sencillo de aprender que los SERVLETS de Java.

- El motor de las páginas JSP está basado en los Servlet de Java.

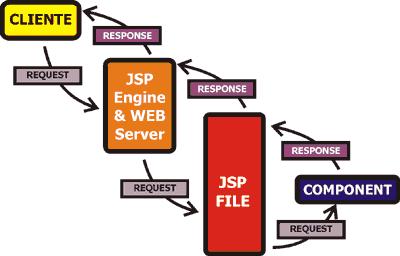
- JSP permite apreciar mejor la distinción entre el contenido y su presentación.

-A diferencia de Java, JSP es un lenguaje de scripting.

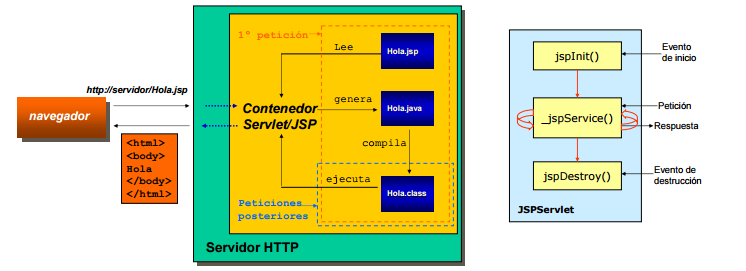
Su **funcionamiento o ciclo de vida** sería el siguiente:



El cliente, sin mantener comunicación previa, solicitara una página con extensión JSP, en lugar de solicitarlo con una página estática HTML. El servidor recibe la petición con información adicional del cliente y entonces ejecuta la página con extensión JSP con un programa JSP y obtiene un resultado en formato HTML estático. Cuando procesa la información también puede hacer consultas a bases de datos. EL resultado HTML es enviado al cliente (rompiendo el enlace de comunicación) y el navegador del cliente visualiza los resultados como si realmente hubiera estado almacenado en el servidor.



**2-Ciclo de vida de un JSP**



**1ºFASE:** Fase de traducción: Tras la primera petición de la página JSP, se traduce en un Servlet (código java) implementando en un archivo class (byte codes de Java). Se realiza únicamente cuando la página JSP se crea o modifica. Existe la opción de pre compilar en código para evitar el tiempo de espera la primera vez que un cliente solicite la página.

**2ºFASE:** Fase de compilación: Dicho Servlet es compilado para poder servir la petición del cliente y las clases de los Servlet generados se cargan en el contenedor de Servlet. Normalmente las fases de traducción y compilación ocurren juntas, y son realizadas por el contenedor automáticamente en la primera petición. Para atender concurrentemente al procesamiento de peticiones, el servidor lanza un proceso ligero (thread) por cada uno de los clientes el cual corre el método \_jspService(). Este método \_jspService() no puede reemplazarse por otros métodos definidos por el programador, pero en cambio puede adecuarse su inicialización y terminación con los métodos jspInit() y jspDestroy().

**3ºFASE:** Fase de ejecución: Se crea una instancia de dicho Servlet, que residirá en memoria de forma permanente mientras el servidor siga en funcionamiento. Para las peticiones posteriores se emplea la misma instancia del Servlet (no se vuelve a compilar la página).

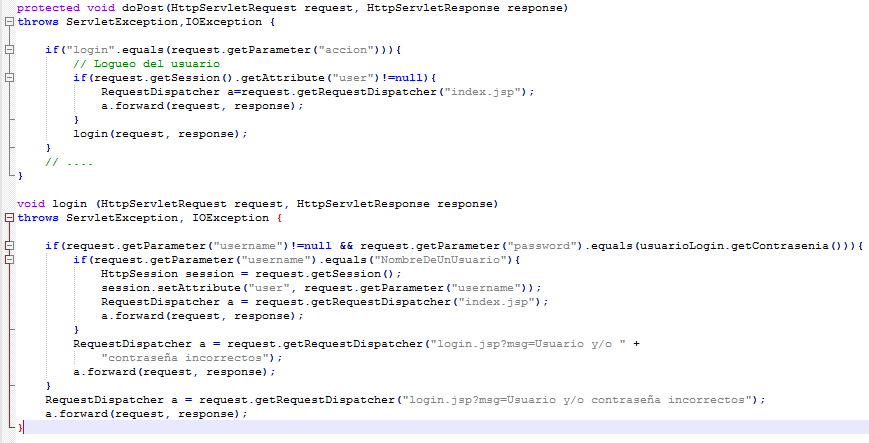
**3- Funcionamiento**

-Durante la fase de traducción se recibe la petición de la página JSP, la cual se traduce en un Servlet.

Ejemplo de **página JSP:**



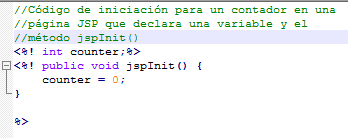
Ejemplo de **Servlet:**



-Durante la fase de compilación  se valida el código Java que escribimos en declaraciones, scriptlets y expresiones.

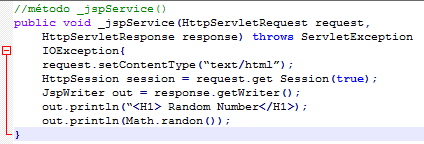
-Durante la fase de ejecución, el contenedor invoca del servlet generado los métodos:  **\*jspInit():** Permite ejecutar cierto código cuando se produce la primera petición.

Ejemplo del **método jspInit:**



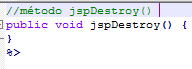
**\*\_jspService():** Se ejecuta en TODAS las peticiones. El programador JSP no debe administrarlo directamente. La mayoría del código java y HTML se incluye dentro. 

Ejemplo del **método jspService:**



**\*jspDestroy():** Permite ejecutar código antes de que finalice el servlet.

Ejemplo del **método jspDestroy:**



**Fuentes:**

<http://www.academia.edu/5376632/Pr%C3%A1ctica_4_PROGRAMACI%C3%93N_WEB_CON_SERVLETS_Y_JSP>

<http://www.vc.ehu.es/jiwotvim/ISOFT2009-2010/Teoria/BloqueIV/Integracion-Jsps-Servlets.pdf>

<https://prezi.com/ojbrawecgrs0/descripcion-de-las-caracteristicas-basicas-del-uso-de-servlets-y-jsp-para-html-del-lado-del-cliente/>

<http://www.inf-cr.uclm.es/www/mpolo/asig/0708/tutorJavaWeb.pdf>

<http://dis.um.es/~lopezquesada/documentos/IES_1213/IAW/curso/UT5/ActividadesAlumnos/19/1.html>

<http://informatica.uv.es/it3guia/ARS/transparencias_2c/Tema13_Servlet-JSP.pdf>

<http://www.tamps.cinvestav.mx/~vjsosa/clases/sd/JSP-extended.pdf>

<http://es.slideshare.net/elimaqp/qu-es-jsp?next_slideshow=1>

<http://es.slideshare.net/fanduberangelwha/prograweb?next_slideshow=2>

<http://es.slideshare.net/mateoramirez23/jsp-35306214?next_slideshow=3>

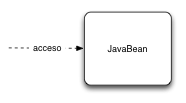
<https://preparandoocejwcd6.wordpress.com/2015/11/15/el-modelo-de-la-tecnologia-jsp-conceptos-basicos/>

<https://www.youtube.com/watch?v=g2NOIlQwl0g>

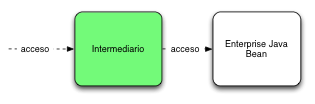
|  |
| --- |
| **EJB (ENTERPRISE JAVABEANS)** |

**1-¿Qué es EJB (ENTERPRISE JAVABEANS)?**

Para empezar hay que entender lo que es un JavaBean. Un JavaBean es una especie de clase de Java que contiene varios objetos dentro de ella con sus métodos y constructores. Su finalidad es ser reutilizable. Un ejemplo sería la clase jugadorCartas que contiene objetos como la mano (cartas que tiene que sería un array u otro objeto) y atributos (número de jugador) , y luego tendrías sus métodos como apostar y por último debe ser serializable. Al JavaBean se puede acceder desde un programa o aplicación directamente.

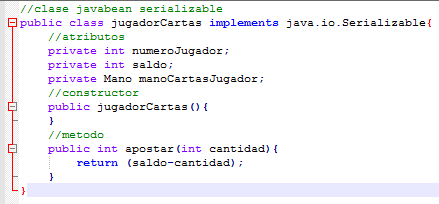


Un EJB (Enterprise Java Bean) en cambio no es una clase sino una serie de clases desarrolladas en lenguaje Java, pero más complejas que conforman componentes que permiten ser utilizados en una estructura de componentes distribuidos del lado del servidor al cual no podemos acceder de una forma tan directa cómo el JavaBean porque necesitaremos de algún tipo de intermediario. Ese programa java realiza alguna gestión compleja y que aguarda a que otro programa cliente lo invoque.

****

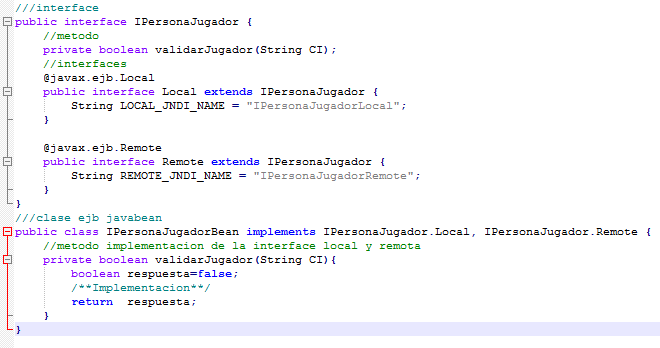
Ejemplo **JavaBean**:

Clase jugadorCartas serializable con atributos y objetos, constructor y un método.



Ejemplo **EJB (Enterprise Java Bean):**

En el ejemplo vemos que EJB (Enterprise Java Bean) debe contener al menos una clase de implementación y dos interfaces; una local o "Home" y la remota. En la implementación describen todos los métodos de las interfaces remota y algunos de la interfaz local. En este caso el método implementado sería validarJugador:



El intermediario que necesita EJB para funcionar es un servidor donde debe instalarse el EJB. Este servidor es otro software encargado de controlar el acceso al EJB, la seguridad, las transacciones y otras tareas o servicios definidos por los estándares en los cuales el EJB se puede apoyar. Un ejemplo sería el servidor de aplicaciones JBoss, BEA WebLogic, IBM Websphere o Oracle10g.

**EJB tiene una serie de características o ventajas tales como:**

-Simplifican el desarrollo de aplicaciones grandes y distribuidas.

- Gran parte de la funcionalidad la gestiona el contenedor. El contenedor proporciona servicios a nivel de sistema a los EJB´S (Seguridad: comprobación de permisos de acceso a los métodos, transacciones: apertura y cierre de transacciones asociadas a las llamadas a los métodos).

-Concurrencia: llamada simultánea a un mismo bean desde múltiples clientes.

-Servicios de red: comunicación entre el cliente y el bean en máquinas distintas.

-Permite dedicarnos sólo a problemas de negocios.

-Abstrae al programador de la capa cliente y este entonces puede dedicarse exclusivamente al diseño.

-Son portables y al ser portables pueden ser incluidos en otra aplicación con beans ya existentes.

-Gestión de recursos: gestión automática de múltiples recursos, como colas de mensajes, bases de datos o fuentes 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 base de datos.

-Escalabilidad: posibilidad de constituir clusters de servidores de aplicaciones con múltiples hosts para poder dar respuesta a aumentos repentinos de carga de la aplicación con sólo añadir hosts adicionales.

-Adaptación en tiempo de despliegue: posibilidad de modificación de todas estas características en el momento del despliegue del bean.

-Son reusables.

-Se define mediante declaraciones.

-El servidor gestionará todas las transacciones.

-Multithreading: Las políticas de multithreading a aplicar se declaran y el servidor se encarga de ejecutarlas. El programador no tiene que ocuparse de este aspecto.

-Con su utilización es posible escribir aplicaciones escalables, fiables y seguras sin escribir código de infraestructura. La existencia de infraestructura permite un desarrollo más rápido de la parte servidora.

**2-Ciclo de vida de un EJB**

Hay dos tipos de beans: **de sesión** - beans de sesión sin estado y con estado – y **con entidad**, cada uno con sus propios ciclos de vida.

**\*BEANS DE SESIÓN**

Los beans de sesión representan sesiones interactivas con uno o más clientes. Los bean de sesión pueden mantener un estado, pero sólo durante el tiempo que el cliente interactúa con el bean. Esto significa que los beans de sesión no almacenan sus datos en una base de datos después de que el cliente termine el proceso así que los beans no son persistentes.

A diferencia de los bean de entidad, los beans de sesión no se comparten entre más de un cliente, sino que existe una correspondencia uno-uno entre beans de sesión y clientes. Por esto, el contenedor EJB no necesita implementar mecanismos de manejo de concurrencia en el acceso a estos beans.

**Existen dos tipos de beans de sesión: con estado y sin él.**

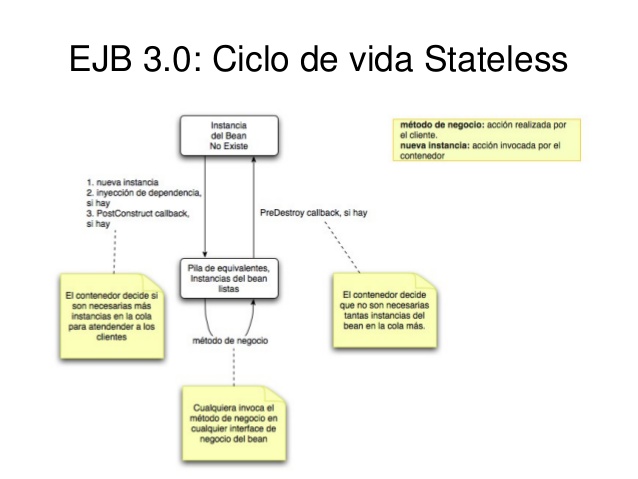
#### Beans de sesión sin estado

-Un cliente invoca un método de un bean de sesión sin estado

-El contenedor EJB obtiene una instancia de la reserva. Cualquier instancia servirá, ya que el bean no puede guardar ninguna información referida al cliente.

-Tan pronto como el método termina su ejecución, la instancia del bean está disponible para otros clientes. Esta propiedad hace que los beans de sesión sin estado sean muy escalables para un gran número de clientes. El contenedor EJB no tiene que mover sesiones de la memoria a un almacenamiento secundario para liberar recursos, simplemente puede obtener recursos y memoria destruyendo las instancias.

Un ejemplo sería cuando una tarea no está ligada a un cliente específico. Un ejemplo es una página que calcula el coste del seguro según los datos pasados pero que no guarda datos personales, sólo lo necesario para el cálculo.



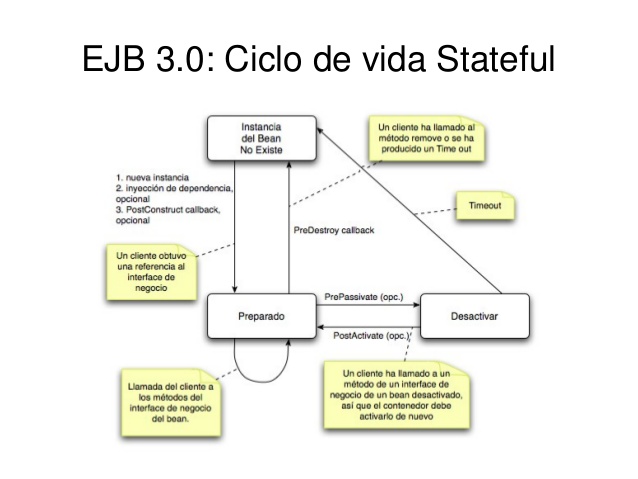
#### Beans de sesión con estado

-El cliente va realizando llamadas a los métodos de negocio del bean.

-Las variables de instancia del bean almacenan datos específicos obtenidos durante la conexión con el cliente. Almacena el estado de un cliente que interactúa con el bean. Este estado conversacional se modifica conforme el cliente va realizando llamadas a los métodos de negocio del vean.

-El estado del bean persiste mientras que existe el bean. A diferencia de los beans de entidad, no existe ningún recurso exterior al contenedor EJB en el que se almacene este estado.

Un ejemplo típico es un carrito de la compra, en donde el cliente va guardando uno a uno los ítem que va comprando.

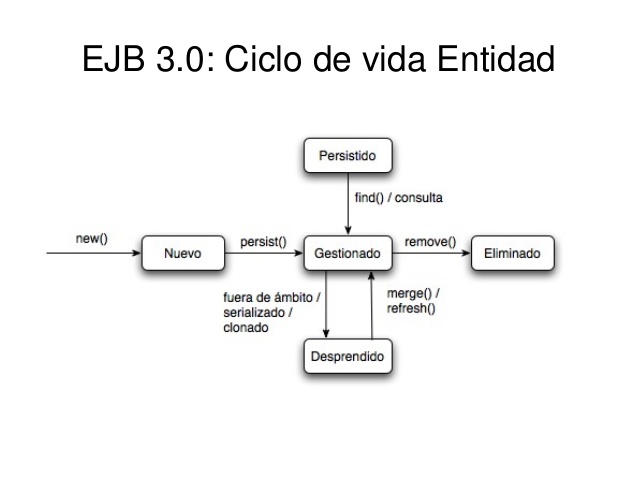


**\*BEANS DE ENTIDAD**

-Primero el cliente debe obtener una referencia a la instancia concreta del bean de entidad que se está buscando (el estudiante "Francisco López") mediante un método finder. Estos métodos finder se encuentran definidos en la interfaz home e implementados en la clase bean. Los métodos finder pueden devolver uno o varios beans de entidad.

-El cliente interactúa con la instancia del bean usando sus métodos get y set. El estado del bean se carga de la base de datos antes de procesar las llamadas a los métodos. Esto se encarga de hacerlo el contenedor de forma automática o el propio bean en la función ejbLoad().

-Por último, cuando el cliente termina la interacción con la instancia del bean sus contenidos se vuelcan en el almacén persistente. O bien lo hace de forma automática el contenedor o bien éste llama al método ejbStore().



**3-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.

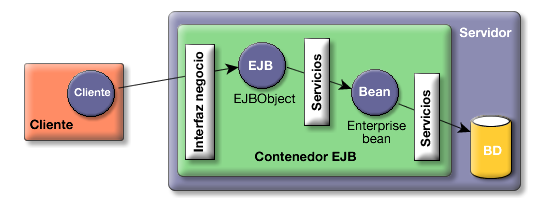
1-El cliente realiza una petición al servidor que contiene el bean. El cliente nunca se comunica directamente con el enterprise bean. La petición del cliente se hace a través de un objeto EJB que realiza al enterprise bean.

2-El servidor que contiene el bean se ejecutará en máquinas virtuales Java distintas.

3-El contenedor EJB proporciona un EJBObject que hace de interfaz.El contenedor EJB comprueba los permisos del cliente para acceder, ofrece una serie de servicios y se comunica con el enterprise bean.

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

5- El contenedor EJB que abre la transacción al comienzo de la llamada, la cierra al terminar.



**Fuentes:**

<http://chuwiki.chuidiang.org/index.php?title=Ejemplo_Sencillo_con_EJB>

<https://es.wikipedia.org/wiki/JavaBean>

<http://di002.edv.uniovi.es/~dflanvin/docencia/dasdi/teoria/Transparencias/13.%20Introducci%C3%B3n%20EJBs.pdf>

<http://javabeansvsejb.blogspot.com.es/2013/02/java-beans-representa-una-clase-como-el.html>

<http://www.arquitecturajava.com/introduccion-a-ejb-3-1-i/>

<http://wiki.netbeans.org/PorqueUsarEJB>

<http://www.inf-cr.uclm.es/www/mpolo/asig/0708/tutorJavaWeb.pdf>

<https://www.fdi.ucm.es/profesor/jpavon/web/45-ejb.pdf>

<http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/ejb/sesion01-apuntes.htm>

<https://www.fdi.ucm.es/profesor/jpavon/web/45-ejb.pdf>

<http://letras-diferentes.info/computadoras/redes-de-computadoras/ejb-del-ciclo-de-vida.php>

<http://es.slideshare.net/aitoracedolegarre/jboss-e>

|  |
| --- |
| **¿EN QUE SE DIFERENCIAN?. CARACTERÍSTICAS DIFERENCIADORAS DE CADA UNO DE ELLOS** |

-La **primera diferencia** entre ellos es que cada uno de ellos se puede aplicar a una capa distinta.

* Los **Servlets** controlan la lógica o **capa de negocio** ya que son extensiones del servidor y proveen una funcionalidad extra al servidor web, son utilizados como controladores de componentes.
* Los **JSP** se utilizan para la presentación visual o **capa de presentación** ya que son páginas web, con una estructura similar a las páginas HTML.
* Un **EJB (Enterprise Java Bean**) es una serie de clases desarrolladas en lenguaje Java, que permiten ser utilizados en una estructura de componentes distribuidos del lado del servidor o **capa de negocios.**

-La **segunda diferencia** es que las aplicaciones de servidores admiten Servlets, EJB y JSP mientras que **los servidores web sólo admite Servlets y JSP.**

-La **tercera diferencia** es la complejidad, el EJB y su desarrollo es sumamente complejo a diferencia de Servlets y JSP que son más fáciles.

-La **cuarta** **diferencia** entre JSP y Servlet es que JSP interactúa directamente con HTML (es la presentación gráfica de la aplicación Java) mientras que el Servlet imprime el HTML con las operaciones resueltas y reenviadas en JSP.

-La **quinta diferencia** es que JSP es un lenguaje de scripting web que pueden generar contenido dinámico mientras Servlet y EJB son programas Java que ya están compilados que también crea contenido web dinámico.

-La **sexta** **diferencia** es que EJB y Servlet se ejecutan más rápido en comparación con JSP.

-La **séptima diferencia** es que los Servlet son clases que deben implementar la clase abstracta HttpServlet, en especial el método doGet() o doPost() y deben ser previamente compilados, mientras que los archivos JSP deben ser interpretados por el servidor al momento de la petición por parte del usuario.

-La **octava** y última diferencia entre Servlets y JSP es que los Servlets son clases que deben implementar la clase abstracta **HttpServlet**, en especial el método **doGet()** o **doPost()** y deben ser previamente compilados, mientras que los archivos JSP contienen código Java entre código HTML utilizando los símbolos **<%** y **%>**.

**Fuentes:**

<http://onj2ee.blogspot.com.es/2009/12/conceptos-basicos-servlets-y-jsp.html>

<http://javarevisited.blogspot.com.es/2012/05/5-difference-between-application-server.html>

<http://www.forosdelweb.com/f45/cual-diferencia-entre-servlet-pagina-jsp-298649/>

<http://www.diferenciaentre.net/jsp-vs-servlet/>

<http://users.dcc.uchile.cl/~lmateu/CC60H/Trabajos/jbarrios/jsp.html>

|  |
| --- |
| **¿PARA QUE TIPO DE APLICACIONES WEB ES MEJOR CADA UNO DE ELLOS?, ¿ES INDIFERENTE?** |

Todo depende del tipo de aplicación web. Si nuestra aplicación necesita alta disponibilidad, escalabilidad, seguridad, portabilidad y delegar la concurrencia es mejor usar EJB pero a la vez es más difícil de desarrollar. La aplicación estará totalmente conectada y distribuida.

También el EJB es mejor usarlo cuando la aplicación tendrá un número alto de usuarios y cada uno de esos usuarios sea muy distinto entre ellos, por ejemplo la página web de un banco con sus clientes, y necesitemos distribuirla en varias máquinas que deban ser localizables. A la vez esa aplicación debe tener integridad de datos, soportar múltiples transacciones, ser reutilizable y deba soportar múltiples accesos concurrentes a sus objetos. Otro ejemplo sería un carrito de compras que donde persistan múltiples sesiones con un mismo cliente.

También usaremos EJB´S si queremos concentrarnos más en la parte lógica de negocio sin tener que preocuparse el desarrollador de las transacciones, detalles de manejo de estado, multi-threading, connection pooling, seguridad y otros detalles de bajo nivel, etc. Debe haber una clara separación entre desarrollo, explotación y administración.

Servlet  y JSP es mejor para aplicaciones más sencillas. Servlet  y JSP siempre van juntos porque cuando JSP se compila siempre se trasforma en Servlet . Y a diferencia de EJB , es conveniente usarlo cuando queremos centrar más nuestra aplicación en la parte gráfica de presentación que en la de negocio.

Por lo contrario hay que usar JSP cuando el dinamismo que se pretende no supone introducir mucho código Java en las páginas, puesto que esto oscurecería el código. Con JSP sólo tienes que usar casi todo el tiempo el HTML y Java se tocará poco excepto cuando haya que modificar alguna funcionalidad. Sin embargo cuando hay mucha funcionalidad y necesitamos mucho código Java, lo ideal sería utilizar una página JSP que llamase a un Servlet que contenga la funcionalidad. En este caso quizás sería más útil usar un EJB si hay que estar realizando múltiples modificaciones en Java.

Es mucho mejor para separar contenidos en grupos de trabajo que EJB ya que una parte del equipo desarrollador puede centrarse en construir el HTML y el equipo programador puede concentrarse en los Servlet . Y también es mucho más fácil de aprender.

También con JSP te ahorras muchos print en la programación.

Por último es más conveniente JSP si los cambios a realizar en el código HTML relativos a cómo son mostrados los datos, no interfieren en la lógica de programación y viceversa.

**Fuentes:**

<https://github.com/sidlors/CursoEJB/wiki/lession3>

<https://blogdeaitor.wordpress.com/2008/10/20/servlet-frente-a-jsp/>

<http://www.inf-cr.uclm.es/www/mpolo/asig/0708/tutorJavaWeb.pdf>

<http://wiki.netbeans.org/PorqueUsarEJB>

<http://www.forosdelweb.com/f45/cuando-utilizar-servlet-jsp-579964/>

<http://www.ciberaula.com/articulo/jsp>

<http://blog.continuum.cl/wp-content/uploads/2008/08/to_ejb_or_not_to_ejb.pdf>

|  |
| --- |
| **VISIÓN PERSONAL** |

Por lo que he investigado creó que Servlet es sólo la parte lógica de la aplicación que sabe que hacer pero no él cómo. Eso se lo indican tanto JSP como el EJB que se pueden usar con esa tecnología.

Todo está basado en código de programación Java para la parte lógica o de negocio pero creó que JSP se centra mucho más en la parte de presentación al usuario (la parte en HTML) tanto para recibir como para enviar los datos que queremos procesar. Y también creó que cómo es más sencillo de usar que EJB está más dirigido a realizar pequeñas aplicaciones web donde la carga de usuarios es mucho más pequeña. Un ejemplo para mí sería una aplicación para una tienda de ropa que maneja relativamente pocos clientes y no tiene demasiadas funcionalidades.

Su desarrollo puede ser realizado por empresas pequeñas ya que su curva de aprendizaje no es alta y las tareas pueden ser divididas con más facilidad.

Por otra parte EJB la veo más para aplicaciones web mucho más grandes y complicadas donde se realizan múltiples operaciones distintas, tienen una base de datos grande que debe ser mantenida constantemente y su utilización es continua por parte de los usuarios (clientes, trabajadores, etc). Su complejidad requerirá que sea muy robusta, escalable (con lo que requerirá un mantenimiento constante de la base de datos), distribuida. El ejemplo que se me ocurre sería por ejemplo la seguridad social donde se realizan múltiples operaciones todos los días, es accedida constantemente por millones de usuarios y sufre múltiples modificaciones.

Otro ejemplo para EJB sería la universidad de La Rioja donde veó distintos componentes como la matriculación de los alumnos (que sería una aplicación web), el campus virtual (que sería otra aplicación web), la gestión de los trabajadores (esto sería una aplicación interna) y la gestión de los profesores (otra aplicación interna distinta de la de los trabajadores).

Su curva de aprendizaje será mucho más grande y debido a su tamaño creó que solo pueden desarrollarla empresas grandes con los recursos suficientes para su mantenimiento y con equipos suficientes preparados no solo para crearlas y gestionarlas, sino también para su constante actualización y modificación.

También creó que una arquitectura bien diseñada dejaría las JSP como vistas, los Servlets como controladores y JavaBeans o EJB´S para la capa de negocio pero debe ser complicado.