# Que es un Servlet?

El servlet es una clase Java, utilizada para ***ampliar las capacidades*** de un servidor. Son utilizadas comúnmente para extender las aplicaciones alojadas por servidores web. Se refieren a ***pequeños programas*** que se ejecutan en el contexto de un navegador web. El uso más común de los servlets es ***generar páginas web de forma dinámica*** a partir de los parámetros de la petición que envíe el navegador web. Toda Servlet por lo general extiende de la clase javax.servlet.http.**HttpServlet**.

# Ciclo de vida

1. **Inicializar el servlet**

Cuando un servidor carga un servlet, ejecuta el método **init** del servlet. El proceso de inicialización debe completarse antes de poder manejar peticiones de los clientes, y antes de que el servlet sea destruido.

Aunque muchos servlets se ejecutan en servidores [multi-thread](https://es.wikipedia.org/wiki/Multihilo), los servlets no tienen problemas de concurrencia durante su inicialización. ***El servidor llama sólo 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**.

1. **Interactuar con los clientes**

Después de la inicialización, el servlet puede dar servicio a las peticiones de los clientes. Estas peticiones serán atendidas por la misma instancia del servlet, por lo que hay que tener cuidado al acceder a variables compartidas, ya que podrían darse problemas de sincronización entre requerimientos simultáneos.

1. **Destruir el servlet**

Los servlets se ejecutan hasta que el servidor los destruye, por ***cierre del servidor*** o bien a petición del administrador del sistema. Cuando un servidor destruye un servlet, ejecuta el método **destroy** del propio servlet. 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.

# Clases y Objetos necesarios

Podemos crear un Servlet haciendo uso del [paquete](https://es.wikipedia.org/wiki/Paquete_Java" \o "Paquete Java) javax.servlet.

Cada nueva Servlet debe extender de la clase javax.servlet.http.**HttpServlet** y sobrescribir el método **doGet(HttpServletRequest req, HttpServletResponse resp)** que es llamado cuando el Servidor Http recibe una petición de tipo GET desde el Browser .

El método **doGet** recibe dos parámetros:

* **HttpServletRequest** Interfaz que contiene toda la información acerca de la petición.
* **HttpServletResponse**  Interfaz que se utiliza para rellenar la respuesta que será enviada al Browser.

|  |  |  |
| --- | --- | --- |
| **MÉTODO** | **CONCEPTO** | **OBSERVACIONES** |
| **GET** | GET lleva los datos de forma "visible" al cliente (navegador web). El medio de envío es la URL. Los datos los puede ver cualquiera. | Los datos son visibles por la URL, por ejemplo:  www.aprenderaprogramar.com/  action.php?nombre=pedro&apellidos1= gomez |
| **POST** | POST consiste en datos "ocultos" (porque el cliente no los ve) enviados por un formulario cuyo método de envío es post. Es adecuado para formularios. Los datos no son visibles. | La ventaja de usar POST es que estos datos no son visibles al usuario de la web. En el caso de usar get, el propio usuario podría modificar la URL escribiendo diferentes parámetros a los reales en su navegador, dando lugar a que la información tratada no sea la prevista. |

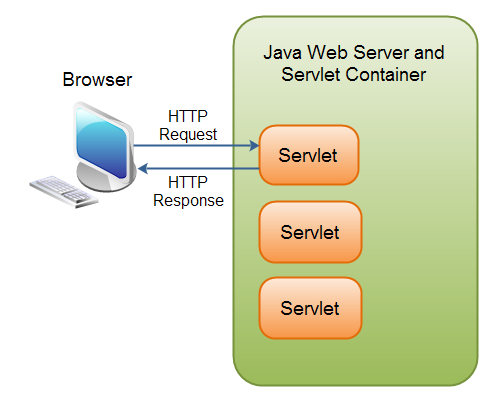
Para utilizar Servlets y JSPs es necesaria la utilización de un **"Contenedor de Servlets"**. Existen muchos contenedores de Servlets como Apache Tomcat, **Jboss**, etc.

## Que es un contenedor de Servlets?

Un contenedor de Servlet es un programa capaz de recibir peticiones de páginas web y redireccionar estas peticiones a un objeto Servlet.

## Como funcionan un contenedor de Servlets?

1. El Browser pide una página al servidor HTTP que es un contenedor de Servlets
2. El contenedor de Servlets delega la petición a un Servlet en particular elegido de entre los Servlets que contiene.
3. El Servlet, que es una objeto java, se encarga de generar el texto de la página web que se entrega al contenedor.
4. El contenedor devuelve la página web al Browser que la solicitó.



# Ejemplo

Código de ejemplo de un Servlet que procesa una petición [GET](https://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol" \o "Hypertext Transfer Protocol) y devuelve una página web [HTML](https://es.wikipedia.org/wiki/HTML" \o "HTML) sencilla:

**package** **org.pruebas**;

**import** **java.io.IOException**;

**import** **java.io.PrintWriter**;

**import** **javax.servlet.ServletException**;

**import** **javax.servlet.http.HttpServlet**;

**import** **javax.servlet.http.HttpServletRequest**;

**import** **javax.servlet.http.HttpServletResponse**;

**public** **class** **HolaMundoServlet** **extends** HttpServlet {

*/\*\**

*\* Servlet de ejemplo que procesa una petición GET*

*\* @param request*

*\* @param response*

*\* @throws ServletException*

*\* @throws IOException*

*\*/*

@Override

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

response.setContentType("text/html");

PrintWriter out = response.getWriter();

out.println("<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 Transitional//EN\">");

out.println("<html>");

out.println("<head><title>Ejemplo Hola Mundo</title></head>");

out.println("<body>");

out.println("<h1>¡Hola Mundo!</h1>");

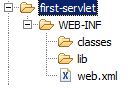
out.println("</body></html>");

}

}

# web.xml

Una aplicación web según las especificaciones de J2EE debe tener una estructura de directorios y un archivo llamado **web.xml** tal como se ve en el siguiente esquema:



Las páginas web estáticas se colocan dentro del directorio de aplicación pero nunca dentro del directorio **WEB-INF\**ya que este no puede ser accedido desde el Browser.

Dentro del directorio **WEB-INF\classes\** se colocaran las clases java compiladas que componen la aplicación (incluidos los servlets). En el directorio **WEB-INF\lib\** se pueden incluir librerías (archivos .jar) que pueda necesitar la aplicación.

**web.xml** es un *descriptor de despliegue* que no es mas que una componente de aplicaciones [J2EE](https://es.wikipedia.org/wiki/J2EE" \o "J2EE) que describe cómo se debe desplegar (o implantar) una aplicación web. Los elementos del web.xml son:  
  
**<web-app>:** Es el elemento raíz del fichero xml.  
**<icon>:** Define la ruta para las imágenes asociadas a los iconos pequeño y grande que representan a la aplicación.

**<display-name>:** Es el nombre que representará a la aplicación dentro de las herramientas del servidor de aplicaciones, no es un nombre funcional.

**<description>:** Texto descriptivo de la aplicación, que al igual que las dos anteriores propiedades, sólo es representativo.

**<context-param>:** Permite configurar parámetros de inicialización del contexto de nuestra aplicación web.

<context-param>

<param-name>nombrehost</param-name>

<param-value>localhost</param-value>

<description> Nombre del host en cual se esta ejecutando la aplicacion</description>

</context-param>

# <filter>: Permite definir un filtro en la aplicación.

# <filter>

# <filter-name>LogSolicitudes</filter-name>

# <filter-class>cl.bee.dukechile.LogSolicitudes</filter-class>

# <init-param>

# <param-name>nombrearchivo</param-name>

# <param-value>registrolog</param-value>

# </init-param>

# </filter>

# <filter-mapping>: Define un mapeo para aplicar las reglas de un determinado filtro a una URL.

# <filter-mapping>

# <filter-name>LogSolicitude</filter-name>

# <url-pattern>/cl/jug/dukechile/\*</url-pattern>

# </filter-mapping>

# <listener>: Permite definir una clase oyente, la que puede escuchar eventos relacionados al clico de vida de la aplicación o modificaciones de un objeto.

# <listener>

# <listener-class>cl.jug.dukechile.ListenerDeContexto</listener-class>

# </listener>

# <servlet>: Permite definir la configuración de nuestros servlets.

# <servlet>

# <servlet-name>DukeChile</servlet-name>

# <servlet-class>cl.jug.dukechile.MiServlet</servlet-class>

# <init-param>

# <param-name>configuracion</param-name>

# <param-value>/WEB-INF/dukechile-config.xml</param-value>

# </init-param>

# <load-on-startup>1</load-on-startup>

# </servlet>

# <servlet-mapping>: Permite relacionar los servlets que hemos declarado con las URL que van a escuchar.

# <servlet-mapping>

# <servlet-name>DukeChile</servlet-name>

# <url-pattern>\*.jsp</url-pattern>

# </servlet-mapping>

# <session-config>: Permite definir el timeout de la sesión en minutos.

# <session-config>

# <session-timeout>10</session-timeout>

# </session-config>

# <mime-mapping>: Permite definir una relación entre las extensiones y los tipos mime.

# <mime-mapping>

# <extension>doc</extension>

# <mime-type>application/vnd.ms-word</mime-type>

# </mime-mapping>

# <mime-mapping>

# <extension>dsp</extension>

# <mime-type>text/html</mime-type>

# </mime-mapping>

# <welcome-file-list>: Permite definir los ficheros de bienvenida de la aplicacion.

# <welcome-file-list>

# <welcome-file>index.jsp</welcome-file>

# <welcome-file>index.html</welcome-file>

# </welcome-file-list>

# <error-page>: Permite asociar errores HTML y excepciones lanzadas por nuestra aplicación a páginas de error personalizadas para la aplicación.

# <error-page>

# <error-code>404</error-code>

# <location>/noEncontrado.jsp</location>

# </error-page>

# <error-page>

# <exception-type>java.lang.Throwable</exception-type>

# <location>/error.jsp</location>

# </error-page>

# <resource-ref>: Permite declarar recursos externos que nuestra aplicación va a utilizar. Por ejemplo, una conexión a una base de datos.

# <resource-ref>

# <description>Conexion BD</description>

# <res-ref-name>jdbc/ConexionMySQL</res-ref-name>

# <res-type>javax.sql.DataSource</res-type>

# <res-auth>Container</res-auth>

# </resource-ref>

# <login-config>: Permite configurar el tipo de login que se requerirá al entrar en un área protegida previamente declarada.

# <login-config>

# <auth-method>FORM</auth-method>

# <realm-name>Login por formulario</realm-name>

# <form-login-config>

# <form-login-page>/login.jsp</form-login-page>

# <form-error-page>/error.jsp</form-error-page>

# </form-login-config>

# </login-config>

# <security-role>: Permite definir los roles de usuarios que se van a utilizar en la aplicación.

# <security-role>

# <role-name>duke</role-name>

# </security-role>

# Despliegue en JBoss

Para realizar el **despliegue de una aplicación web** en JBoss basta con:

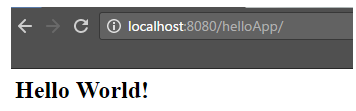
1. Crear el empaquetado de la aplicación (.war).
2. Copiarlo dentro de la carpeta **jboss\standalone\deployments.**
3. Levantar el servidor desde **jboss\bin\standalone.bat**.
4. Abrir la url llamando la servlet que definamos.

Por último nos resta reiniciar el servidor JBoss y llamar en el browser la URL **http://localhost:8080/helloApp/**

# Crear Proyecto Web con Maven

1. Generar archetipo para proyectos web **maven-archetype-webapp**.

mvn archetype:generate -DgroupId=com.everis.web -DartifactId=helloApp -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false



1. Para poder crear servlets debemos agregar la siguiente dependencia al pom.xml:

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>**javax.servlet-api**</artifactId>

<version>3.1.0</version>

</dependency>

1. Creamos una clase SimpleServlet que imprima el mensaje "SimpleServlet Executed".

public class SimpleServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

PrintWriter out = response.getWriter();

out.println( "SimpleServlet Executed" );

out.flush();

out.close();

}

}

1. Definir la servlet en el web.xml.

<servlet>

<servlet-name>simple</servlet-name>

<servlet-class>com.everis.servlets.SimpleServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>simple</servlet-name>

<url-pattern>/simple</url-pattern>

</servlet-mapping>

</servlet>

## Páginas web dinámicas: JSP

Las páginas JSP (Java Server Pages) surgen con la idea de facilitar la creación de contenido dinámico a desarrolladores sin necesidad de conocer a fondo el lenguaje Java.

Una página JSP combina **código HTML** con fragmentos de **código Java** con el objetivo de producir un contenido Web en el que se mezclan tanto componentes estáticos como dinámicos.

Para pasar datos de un JSP a una Servlet utilizaremos las Request mediante el uso de formularios. Utilizaremos los formularios para la creación de cuestionarios, páginas de comentarios, logins o cualquier parte del documento que requiera interacción con el usuario.

**Formulario**:

* **Id**: Nombre del formulario, deberá ser único.
* **Action**: Servlet que tratará la petición.
* **Method**: Protocolo que se usará para el envío de los datos. Los valores pueden ser GET y POST, con el primero los par se envían los parámetros de la request para el servlet dentro de la URL mientras que con el segundo los parámetros de la request se encapsulan en la trama que se envía por el protocolo HTTP.

A continuación mostramos un ejemplo de login mediante el uso de Servlets, éste código sóolo realiza el envío de los datos, en el servlet tendremos que recibirlos y tratarlos. Lo primero sera crear una pagina login.jsp dentro de la carpeta webapp:

*<form id="myForm" action="./login" method="get">*

*<ul>*

*<li>*

*<input id="username" name="username" placeholder="Username"*

*title="Email" type="text" value="" size="40"/>*

*</li>*

*</ul>*

*<ul>*

*<li>*

*<input id="password" name="password" placeholder="Password"*

*title="Password" type="password" value="" size="25"/>*

*<input id="submit" name="submit" type="submit" value="Login">*

*</li>*

*</ul>*

*<ul>*

*<input type="hidden" value="login" name="accion"/>*

*</ul>*

*</form>*

*<br><b><a style="color:red"><%=request.getParameter("msg")!=null?request.getParameter("msg"):""%></a></b>*

Luego vamos a crear una Servlet nueva LoginServlet

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

String userName = request.getParameter("username");

String password = request.getParameter("password");

if(userName != null && !userName.isEmpty() && password != null && !password.isEmpty()){

PrintWriter out = response.getWriter();

out.println("User "+userName + " logged!!!");

out.flush();

out.close();

}

RequestDispatcher a = request.getRequestDispatcher("login.jsp?msg=Usuario y/o contraseña incorrectos");

a.forward(request, response);

}

Definimos en web.xml la nueva servlet:

<servlet>

<servlet-name>login</servlet-name>

<servlet-class>com.everis.servlets.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>login</servlet-name>

<url-pattern>**/login**</url-pattern>

</servlet-mapping>

Entrar url <http://localhost:8080/helloApp/login.jsp> para ver pantalla de login.

Entrar usuario y contraseña y dar click en el boton Login.