Servlets

Aplicaciones Distribuidas

Curso 2025/2026

Introducción a Servlets

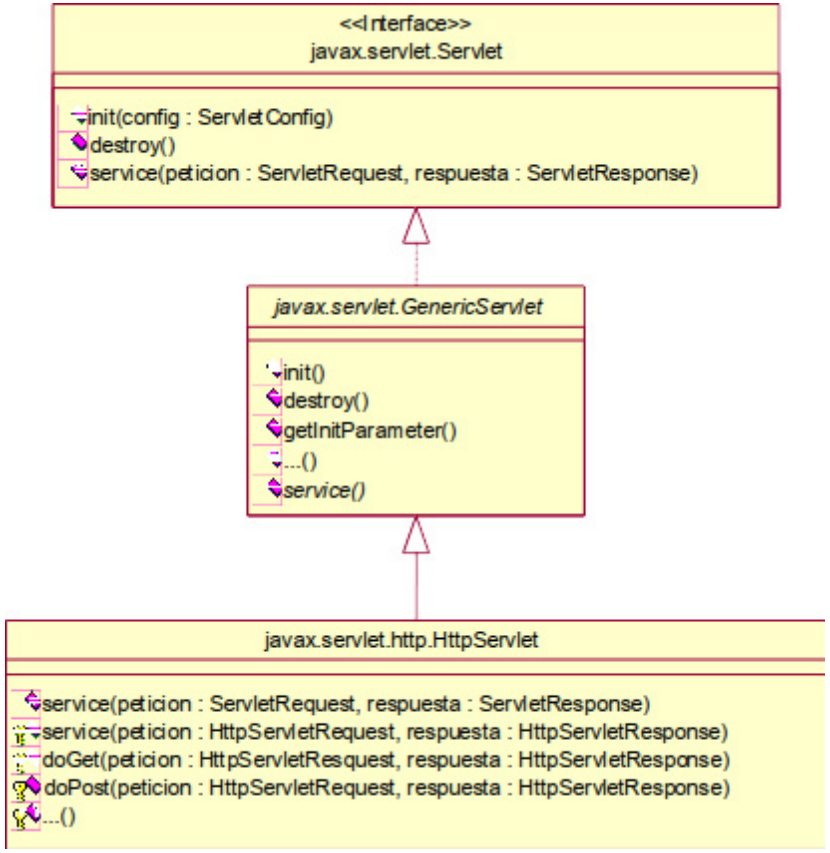
- Un Servlet es un componente Java que puede ser usado en un servidor para generar contenedio web dinámico:
  - El contenido generado depende de la petición realizada o del momento en el que se hace la petición.
- Se ejecutan dentro de un contenedor de servlets en el lado del servidor y gestionan las peticiones HTTP que recibe del cliente.
- Tecnología Java que surge como alternativa a la programación CGI (Common Gateway Interface).
  - Los servlets son portables y más rápidos, permite compartir datos, se comunican directamente con el servidor web, etc.

Contenedor de Servlets

- Conjunto integrado de objetos que proporcionan un entorno de ejecución para los componentes Java Servlet. Se ocupan de:
  - Proporciona los servicios de red a través de los cuales se envían la petición y la respuesta.
  - Proporciona el servicio de decodificación y codificación de mensajes basados en MIME.
  - Gestiona todo el ciclo de vida de un Servlet.
  - Gestiona los recursos estáticos y dinámicos (archivos HTML, servlets, páginas JSP).
  - Gestiona la autorización y autenticación de acceso a los recursos.
  - Gestiona la sesión de usuario.

Framework de Servlets

- La especificación Servlet define un framework de programación Petición/Respuesta.
- Especialización para el protocolo HTTP.
- Paquetes:
  - `javax.servlet.*`
  - `javax.servlet.http.*`



Configuración

- Un servlet se puede definir mediante configuración XML o mediante anotaciones.
  - Mediante anotaciones:

```
@WebServlet("/contador")
public class ContadorServlet extends javax.servlet.http.HttpServlet{
    //...
}
```

- Por defecto el nombre del servlet es el nombre de la clase. Se puede modificar con el atributo `name` de la anotación `WebServlet`. También se pueden desplegar en varias URLs y definir parámetros de inicialización.

```
@WebServlet(urlPatterns={"/contador", "/contadorServlet"})
public class ContadorServlet extends javax.servlet.http.HttpServlet{
    //...
}
```

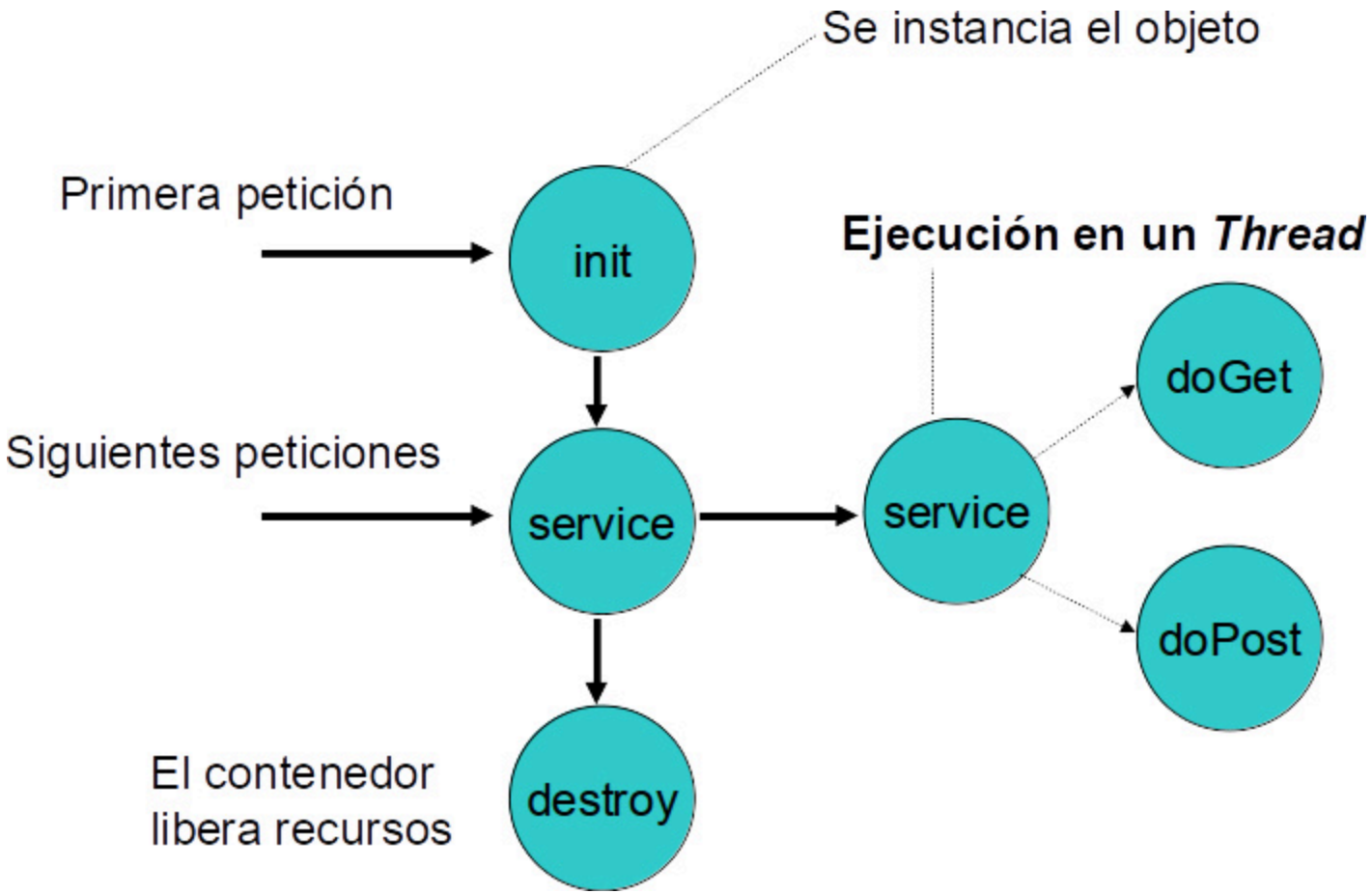
Configuración

- La configuración por xml se realiza en el archivo `web.xml`.

```
<servlet>
    <servlet-name>ContadorServlet</servlet-name>
    <servlet-class>aadd.ejemplo.ContadorServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>ContadorServlet</servlet-name>
    <url-pattern>/contador</url-pattern>
</servlet-mapping>
</web-app>
```

- La configuración por xml sobreescribe a la configuración por anotaciones.

Ciclo de vida



Modelo de ejecución

- Un solo objeto instanciado por Servlet.
- Cada petición se ejecuta en un hilo diferente.
- Para hacer programación `thread-safe`:
  - Uso de variables locales y parámetros.
  - Bloques de sincronización para las actualizaciones.

```
private int codigo ;
public void doPost (...) {
    synchronized ( this ) {
        codigo ++;
        // ...
    }
}
```

Procesamiento de peticiones

- La interfaz `HttpServlet` tiene un método `doXX` para gestionar las peticiones HTTP GET, POST, PUT, DELETE, HEAD, OPTIONS, y TRACE.
- Normalmente los desarrolladores trabajan con los métodos `doGet` y `doPost`.

```
@Override
protected void doGet(HttpServletRequest request,HttpServletResponse response) {
//. . .
}
```

- Los objetos `HttpServletRequest` y `HttpServletResponse` capturan la petición del cliente y la respuesta enviada.

Procesamiento de una petición

- Información de petición: `HttpServletRequest`
- Recoger parámetro de la petición:
  - Organizados en un diccionario multivaluado de cadenas.

```
String nombre = request. getParameter("nombre");
String[] preferencias = request.getParameterValues("preferencias");
```

  - Parámetros petición:
    - `getParametersNames():Enumeration`
- Recoger cabeceras HTTP:

```
String referer = request.getHeader("referer");
```

- Los parámetros siempre son `String`.
- Cuando un parámetro no ha sido definido, devuelve `null`.

Generación de la respuesta

- Información Respuesta: `HttpServletResponse`.
- La salida siempre se genera siguiendo la estructura de una respuesta HTTP.
- Código de estado: `setStatus(valor)`.
  - Por defecto, `OK (SC_OK)`.
- Cabeceras:
  - Tipo MIME: `setContentType(), text/html, image/gif`.
  - Otras: refresco caché `response.setHeader("refresh", "10; index.html");`.
- Respuesta: se genera a un `stream` de salida.

```
PrintWriter out = response.getWriter();
out.println("Hola Mundo");
```

- También: `sendError` y `sendRedirect`.

Códigos de estado

Mnemonic Constant	Code	Default Message	Meaning
SC_OK	200	OK	The client's request was successful, and the server's response contains the requested data. This is the default status code.
SC_NO_CONTENT	204	No Content	The request succeeded but there was no new response body to return. Browsers receiving this code should retain their current document view. This is a useful code for a servlet when it accepts data from a form but wants the browser view to stay at the form, as it avoids the "Document contains no data" error message.
SC_MOVED_PERMANENTLY	301	Moved Permanently	The requested resource has permanently moved to a new location. Future references should use the new URL in requests. The new location is given by the `Location` header. Most browsers automatically access the new location.
SC_MOVED_TEMPORARILY	302	Moved Temporarily	The requested resource has temporarily moved to another location, but future references should still use the original URL to access the resource. The new location is given by the `Location` header. Most browsers automatically access the new location.
SC_UNAUTHORIZED	401	Unauthorized	The request lacked proper authorization. Used in conjunction with the `WWW-Authenticate` and `Authorization` headers.
SC_NOT_FOUND	404	Not Found	The requested resource was not found or is not available.
SC_INTERNAL_SERVER_ERROR	500	Internal Server Error	An unexpected error occurred inside the server that prevented it from fulfilling the request.
SC_NOT_IMPLEMENTED	501	Not Implemented	The server does not support the functionality needed to fulfill the request.
SC_SERVICE_UNAVAILABLE	503	Service Unavailable	The service (server) is temporarily unavailable but should be restored in the future. If the server knows when it will be available again, a `Retry-After` header may also be supplied.

Contexto de la aplicación

- Los servlets se empaquetan en una aplicación en un fichero `.war`.
- Se pueden empaquetar multitud de servlet juntos y todos comparten el contexto de la aplicación.
- El objeto `ServletContext` proporciona detalles del entorno de ejecución y se usa para comunicarse con el contenedor.
- Se obtiene a partir del objeto `HttpServletRequest`.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) {
    ServletContext context = request.getServletContext();
    //. . .
}
```

Seguimiento de sesiones

- Protocolo HTTP sin estado.
- Es necesario mecanismos adicionales para el seguimietno de la sesión del usuario.
- El servlet envía la cookie `JSESSIONID` con un ID de sesión único para cada cliente.
- Esta cookie le permite al servidor mantener el estado de la sesión a través de múltiples solicitudes.
- Es una cookie no persistente.
- Si las cookies no estás disponibles: reescritura de URL.
  - El servidor agrega el ID de sesión a todas las URLs que se envían al cliente.
  - Inclusión de un parámetro en la URL `http://ejemplo.com/aadd?jsessionid=12345ABCDEF`.

Sesiones

- La información de la sesión está en el objeto `HttpSession`.
- El contenedor ofrece al servlet la sesión asociada a la petición:

```
HttpSession sesion = request.getSession();
```

- Vincular objetos a la sesión:
  - Tabla (String - Object).

```
Cliente c = (Cliente)sesion.getAttribute("cliente");
sesion.setAttribute("fallos", new Integer(0));
```

- Tiempo de vida limitado
  - Configurable en `web.xml` de la aplicación web:

```
<session-config>
    <session-timeout>20</session-timeout>
</session-config>
```

Cookies

- Cookies adicionales pueden ser enviadas y recuperadas desde y hacia el cliente.
- Las cookies se añaden a la respuesta:

```
Cookie c = new Cookie("id", identificacion);
response.addCookie(c);
```

- Las cookies se recuperan de la petición:

```
Cookie [] cookies = request . getCookies ();
if ( cookies [0]. getName (). equals ("id"))
    out . println ( cookies [0]. getValue ());
```

Colaboración entre servlets

- Un servlet puede pasarle una petición a otro servlet si se necesita realizar procesamiento adicional.
- A través del objeto `RequestDispatcher`:
- Un servlet puede invocar a otro servlet y guardar información para otros servlets
- Invocación de un servlet `RequestDispatcher`:

```
protected void doGet(HttpServletRequest request,
 HttpServletResponse response) {
    RequestDispatcher rd =
        request.getRequestDispatcher("servlet/Hola");
    rd.forward(request,response);
    //..
```

Colaboración entre servlets

- También se puede redirigir una respuesta a otro recurso invocando el método `HttpServletResponse.sendRedirect`.
- Se envía una respuesta de redirección al cliente, y el cliente envía una nueva petición a la URL de redirección.

```
protected void doGet(HttpServletRequest request,
HttpServletResponse response) {
    response.sendRedirect("http://ejemplo.com/aadd/OtroServlet");
//...
```

- La URL de redirección puede referirse a otro servidor y puede ser relativa o absoluta respecto al contenedor.

Colaboración entre servlets

- Comunicación a través del Contexto:
  - Los servlets comparten un objeto `ServletContext`.

```
contexto.setAttribute("factoriaDAO", factoriaDAO);
FactoriaDAO f = (FactoriaDAO) contexto.getAttribute("factoriaDAO");
```

- El contexto es accesible a travlés del objeto `ServletConfig`:

```
ServletContext contexto = getServletConfig().getServletContext();
```

Servlet Filter

- Permite interceptar una petición a un servlet.
  - Preprocesamiento de la petición antes de que se envíe al servlet y postprocesamiento de la respuesta.
- Tareas comunes:
  - Escribir ficheros de `Log`.
  - Autenticación y autorización de peticiones a recursos.
  - Formatear datos de la petición (compresión, encriptado...).
  - Modificar la respuesta añadiendo cookies, cabeceras, etc.
- Podemos tener múltiples filtros en un mismo recurso (cadena de filtros): patrón **Cadena de Responsabilidad**.

Servlet Filter

- Intefaz `javax.servlet.Filter`.
- Anotación `@WebFilter("/*)` o configuración en `web.xml`.
- Método `doFilter`:
  - `doFilter(ServletRequest request, ServletResponse response, FilterChain chain)`.
- Petición y Respueta compartida por filtros y servlet.
- `FilterChain`: propaga la petición por la cadena de filtros `chain.doFilter(request, response)`.

Event Listeners

- Los *Event Listeners* son clases que soportan notificaciones de eventos para cambios de estado en los objetos `ServletContext`, `HttpSession` y `ServletRequest`.
- Las clases se anotan con `@WebListener` o se configurarn en el `web.xml`.
- `ServletContextListener`: escucha a eventos relacionados con el ciclo de vida del contexto de la aplicación.
- `ServletContextAttributeListener`: escucha por cambios en atributos del contexto.
- `HttpSessionListener`: escucha eventos sobre el ciclo de vida del objeto de sesión.
- `HttpSessionAttributeListener`: escuchar eventos relacionados con la gestión de atributos en la sesión.
- `HttpSessionBindingListener`: escuchar eventos relacionados con la vinculación de recursos a la sesión.
- `ServletRequestListener`: eventos relacionados con el ciclo de vidad e una solicitud.

Soporte asíncrono

- Soporte de ejecución asíncrona en servlets y filtros
- Evita bloquear el `thread` en procesos largos.
- El servidor ejecuta el proceso asíncronamente.
- El control es devuelto al contenedor para realizar otras tareas.
- `AsyncContext` representa el contexto de ejecución de la petición.
- Para completar la petición asíncrona: `asyncContext.complete()`.

```
@WebServlet(urlPatterns="/ejemplo", asyncSupported=true)
public class MyAsyncServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
    HttpServletResponse response) {
    AsyncContext ac = request.startAsync();
    //...
```

Gestión de solicitudes multipartes

- Con la anotación `@MultipartConfig` se indica que el servlet espera una petición de tipo `multipart/form-data`.

```
@WebServlet(urlPatterns = {"/FileUploadServlet"})
@MultipartConfig(location="/tmp")
public class FileUploadServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request,HttpServletResponse response){
            for (Part part : request.getParts()) {
            part.write("myFile");
        //...
```

En este código:

- El atributo `location` se usa para indicar el directorio en el que se guardarán los archivos.
- Se usa `part.write` para escribir la *parte* subida en disco.

Uso de Servlets

- Uso tradicional:
  - Aplicaciónes web dinámicas.
  - Controladores en el patrón MVC (Modelo-Vista-Controlador).
  - Manejo de sesiones y autenticación.
  - Creación de servicios Restful sencillos.
  - Generación de archivos y descargas.
- Hoy en día existen otras tecnologías y frameworks que nos evitan la necesidad de usar servlets de forma directa, aunque todavía se utilizan para tareas específicas en aplicaciónes JEE.

Servlets en Frameworks de Desarrollo Web

- **Spring MVC**: parte del ecosistema Spring.
  - Framework que sigue el patrón Modelo-Vista-Controlador (MVC).
  - Utiliza servlets de manera transparente. `DispatcherServlet` actúa como el controlador central que enruta las solicitudes a los controladores (*controllers*) definidos por el desarrollador.
- **JavaServer Faces (JSF)**: framework basado en componentes para desarrollar interfaces de usuario en aplicaciones Java EE.
  - Utiliza servlets de forma transparente para gestionar las solicitudes y el ciclo de vida de los componentes.
  - `FacesServlet` gestiona el ciclo de vida de las solicitudes HTTP.
  - El desarrollador no interactura directamente con los servlets.

Servlets en Frameworks de Desarrollo Web

- **Grails**: framework de desarrollo rápido que utiliza **Groovy** y se basa en el ecosistema Spring.
  - Utiliza Spring MVC y, por lo tanto, el `DispatcherServlet` para gestionar las solicitudes.
  - El desarrollador no interactura con los servlets.
- **JHipster**: plataforma para generar aplicaciones basadas en Spring Boot y Angular/React, que abstrae la mayoría del ciclo de vida de los servlets utilizando el stack de Spring.
  - Sus aplicaciónes pueden usar internamente `DispatcherServlet` en el backend, pero de forma totalmente transparente al desarrollador.