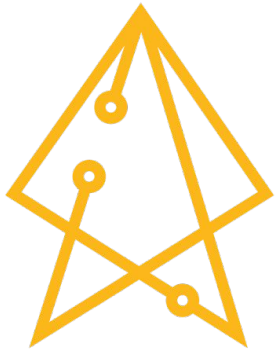
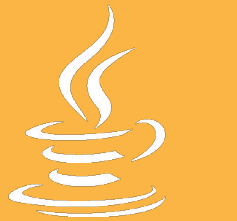
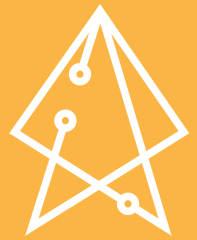


ACADEMIK



Servlets (Filtering, Context & Sessions)



ACADEMIK

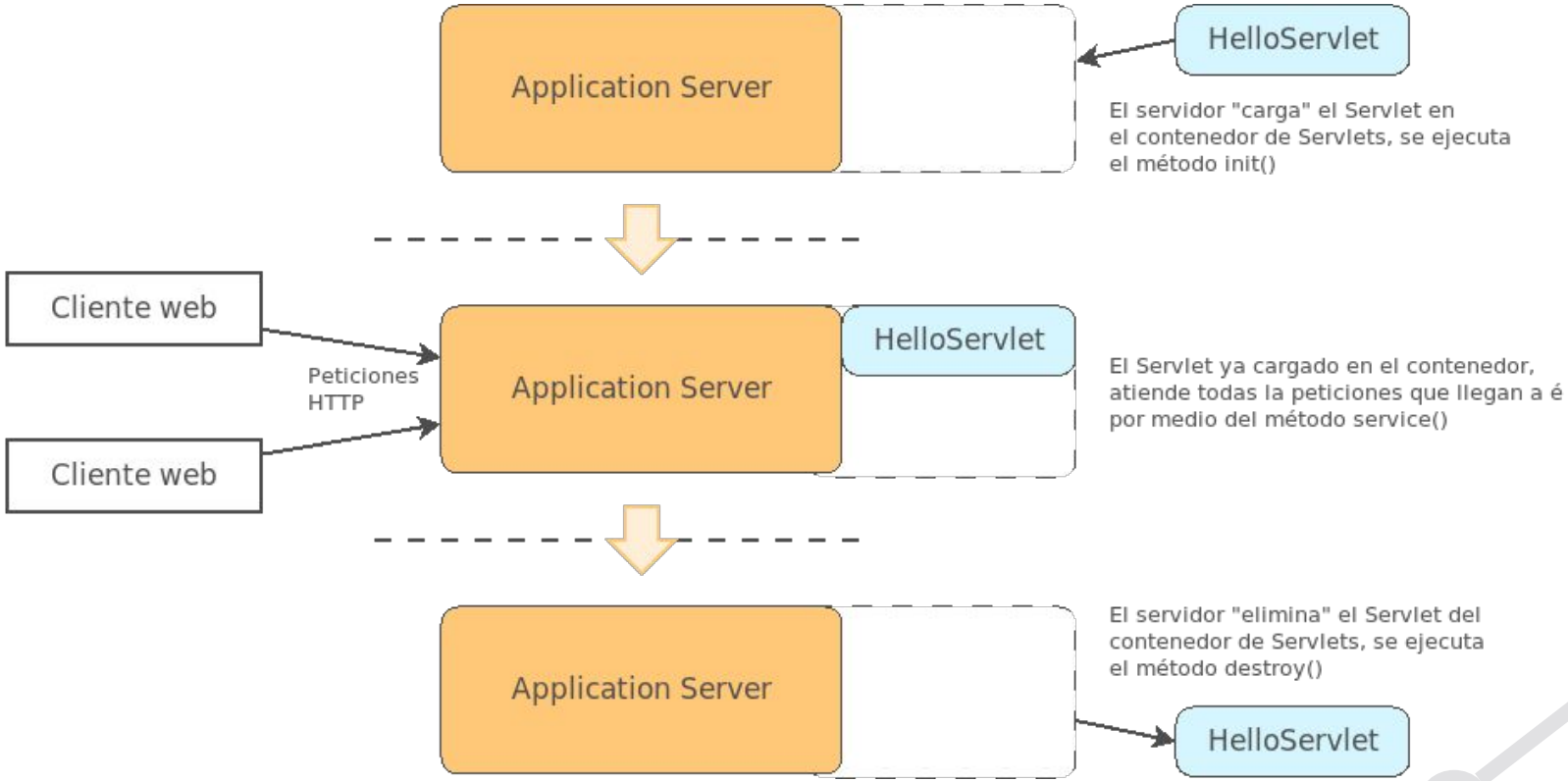
¿Qué más se puede hacer con los Servlets?

Los servlets son la base de la mayoría de componentes web, y mucho más

Ciclo de vida de un Servlet

Los Servlets tienen un ciclo de vida característico:

- El servidor de aplicaciones carga e inicializa el Servlet, ejecución del método **init()**
- El Servlet gestiona las peticiones del cliente por medio del método **service()**, este método es el que se encarga de llamar a los distintos métodos de servicio como **doGet**, **doPost**, **doPut**, etc.
- El servidor destruye los Servlets, ejecutando el método **destroy()**, en alguna de las siguientes situaciones:
    - El contenedor o la aplicación son “apagados”
    - Cuando el contenedor se “queda corto” de memoria
    - Cuando el servlet no ha atendido peticiones en mucho tiempo



Ciclo de vida de un Servlet

Inicialización de un Servlet

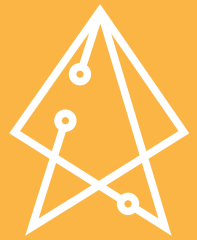
```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {

    @Override
    public void init() throws ServletException {
        // initialize here
    }

}
```

Un Servlet trabajando

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {

    @Override
    protected void service(
        HttpServletRequest req,
        HttpServletResponse resp
    ) throws ServletException, IOException {
        // all request are dispatched here
    }

}
```

Destrucción de un Servlet

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {

    @Override
    public void destroy() {
        // all finish here
    }

}
```



ACADEMIK

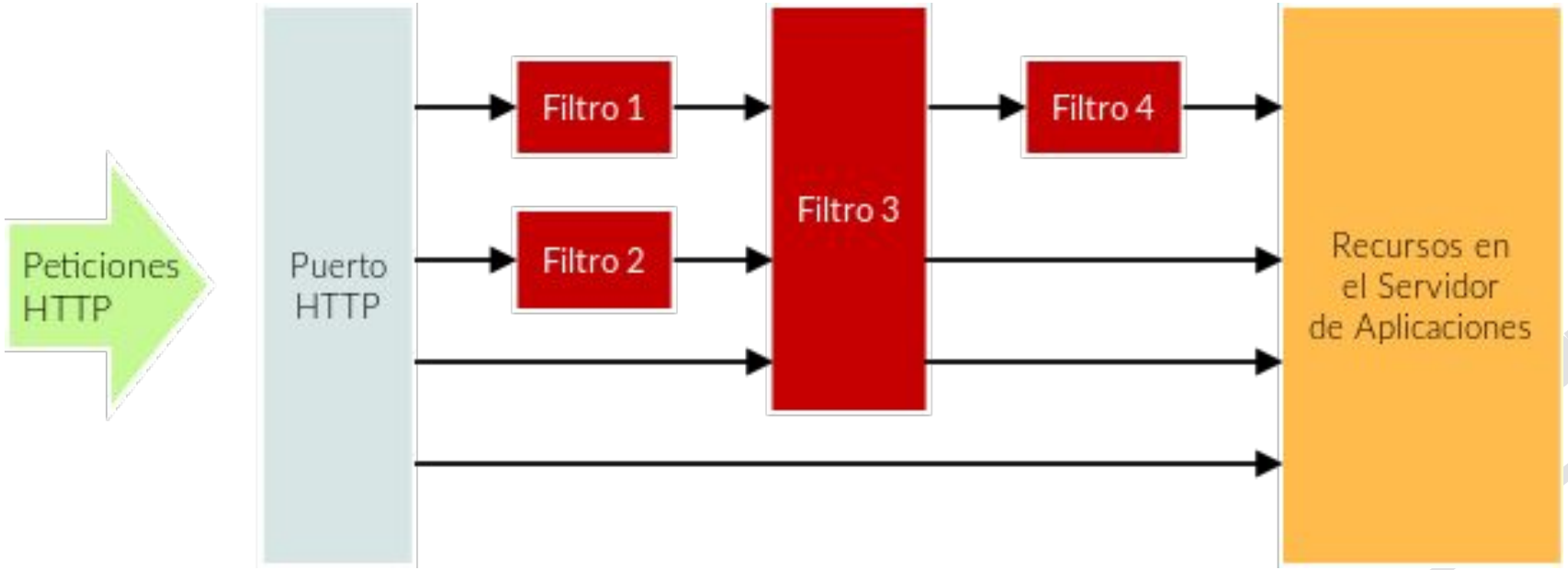
¿Qué más se puede hacer con Servlets?

Almacenar sesiones, redirigir y filtrar peticiones

Redireccionando

Un Servlet puede realizar una redirección utilizando el objeto HttpServletResponse que recibe como parámetro en cualquiera de los métodos de servicio.

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {

    @Override
    protected void doGet(
        HttpServletRequest req, HttpServletResponse resp
    ) throws ServletException, IOException {
        resp.sendRedirect("https://www.google.com");
    }

}
```



Funcionamiento de **Filters** en Java EE

Filtrando peticiones

Las peticiones que llegan al servidor pueden ser filtradas, esto con el objetivo de realizar validaciones de forma general, **por ejemplo**, proteger el acceso a algunas rutas por medio de headers. Al implementar la interface javax.servlet.Filter es necesario sobrescribir los métodos:

```
public void init(FilterConfig conf) throws ServletException;

public void destroy();

public void doFilter(
    ServletRequest request, ServletResponse response, FilterChain chain
) throws ServletException, IOException;
```

Filtrando peticiones (II)

El filtrado se realiza tanto para elementos dinámicos (código Java) como para los recursos estáticos, archivos HTML u otros recursos del proyecto.

```
@WebFilter("/private/*")
public class PrivacityFilter implements Filter {

    @Override
    public void doFilter(
        ServletRequest req, ServletResponse resp, FilterChain chain
    ) throws ServletException, IOException {
        // filter request, You shall not pass!!!
    }

}
```

Filtrando peticiones (III)

Los filtros en Java EE funcionan como paredes, las cuales son “pasadas” por las peticiones cuando se llama al método **doFilter** del objeto de tipo FilterChain recibido como tercer parámetro en el método **doFilter**.

```
@WebFilter("/private/*")
public class PrivacityFilter implements Filter {

    @Override
    public void doFilter(
        ServletRequest req, ServletResponse resp, FilterChain chain
    ) throws ServletException, IOException {
        chain.doFilter(req, resp);
    }

}
```

Primera parte

- Crear una nueva aplicación web Maven
- En el paquete principal crear un paquete llamado **servlets**
- En el paquete principal crear un paquete llamado **filters**
- Dentro del paquete **servlets** crear los siguientes servlets:
  - **HelloServlet**, con el método doGet habilitado, path **“hello”**
  - **GoogleServlet**, con el método doGet habilidato, path **“go-google”**
- En HelloServlet sobrescribir los métodos **init()** y **destroy()** colocando algún mensaje en consola cuando el servlet sea inicializado o destruido, probar y discutir cuando son llamados estos métodos
- En HelloServlet sobrescribir el método service() y como cuerpo colocar:

```
super.service(req, resp);
System.out.println(“Ahorita no joven estoy trabajando… ”);
```

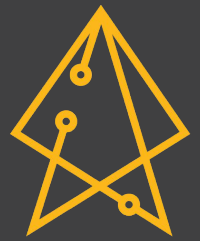
- Consultar la URL “/hello”, luego discutir qué sucede cuando se comenta la línea `super.service(req, resp);` y se vuelve a consultar “/hello”

Segunda parte

- Dentro del paquete **filters** crear una clase llamada **PrivacityFilter**
  - Hacer que esa clase herede de **javax.servlet.Filter**
  - Agregar a nivel de clase la anotación **@WebFilter(“/private/*”)**
  - Implementar los métodos **init()**, **destroy()** y **doFilter()** colocando algún mensaje en consola en los métodos de inicialización y destrucción
  - En el método doFilter, colocar como instrucciones:

```
chain.doFilter(req, resp);
System.out.println(“Estoy filtrando en PrivacyFilter”);
```

- Dentro del paquete **servlets** crear un Servlet llamado **SecretServlet**
  - Colocar a este servlet el path **“/private/hello”**
  - En este servlet habilitar el método **doGet** mostrando un HTML que diga Bienvenido elegido!!!
  - Probar la URL **/private/hello**
- Poner atención a la explicación que sigue



ACADEMIK

Gracias por su atención

Escríbenos a: cursos@nabenik.com

www.academik.io