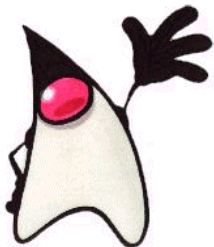




# Java™ Filters en la Práctica

Igvir Ramírez  
igvir@yahoo.com



# Agenda



- ✍ ¿Qué es un filtro?
- ✍ Usos comunes
- ✍ ¿Cómo funcionan los filtros?
- ✍ Definir un filtro en el web.xml
- ✍ La cadena de filtros
- ✍ Reutilizar un filtro
- ✍ Ejemplo: Control de Credenciales

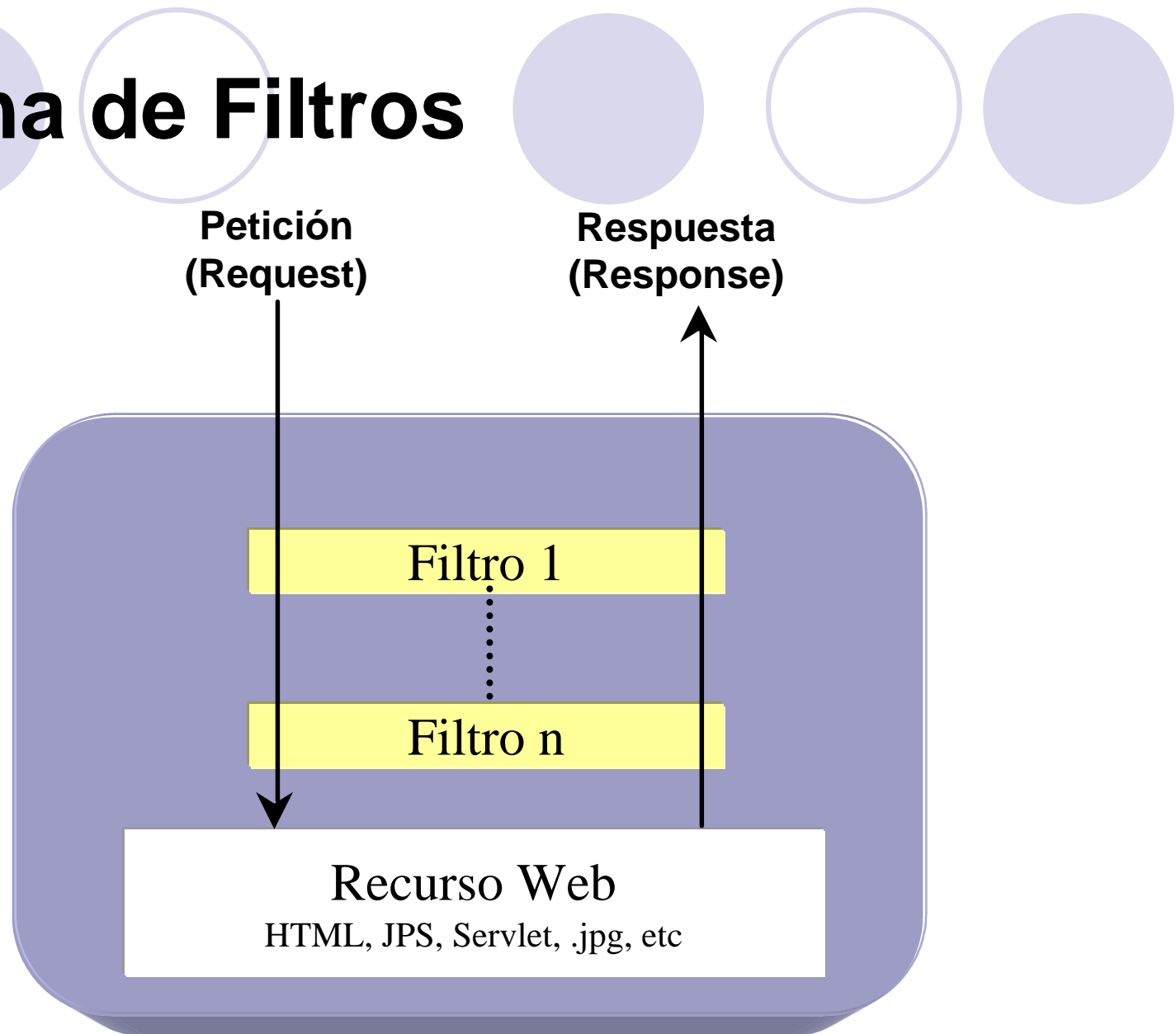
# ¿Qué es un filtro? (1 de 2)

- ✍ Los filtros son componentes que pueden utilizarse para analizar y/o transformar tanto los datos solicitados como los enviados en una petición web.
- ✍ Pueden trabajar en conjunto con páginas jsp o servlets.

# ¿Qué es un filtro? (2 de 2)

- ✍ Son parte de la especificación de Servlets 2.3 lo que los hace portables entre los diferentes contenedores disponibles en el mercado.
- ✍ Pueden trabajar “encadenados”. Un filtro realiza su trabajo y pasa el control al filtro siguiente.

# Cadena de Filtros



# Ventajas



- ✍ Son componentes reutilizables
- ✍ Son parte del estándar
- ✍ Son fáciles de implementar
- ✍ Se pueden incorporar y retirar de forma sencilla
- ✍ Pueden ofrecer gran variedad de servicios

# Usos Comunes



- ✍ Control de acceso a la aplicación
- ✍ Compresión de datos y Cache
- ✍ Transformaciones XML / HTML
- ✍ Procesamiento de imágenes
- ✍ Auditoria/registro de actividades
- ✍ Virtualización de recursos
- ✍ Cifrado de datos

# ¿Cómo funcionan? (1 de 3)

✍ Un filtro debe implementar la interfaz **javax.servlet.Filter** que incluye los métodos:

✍ `init()`

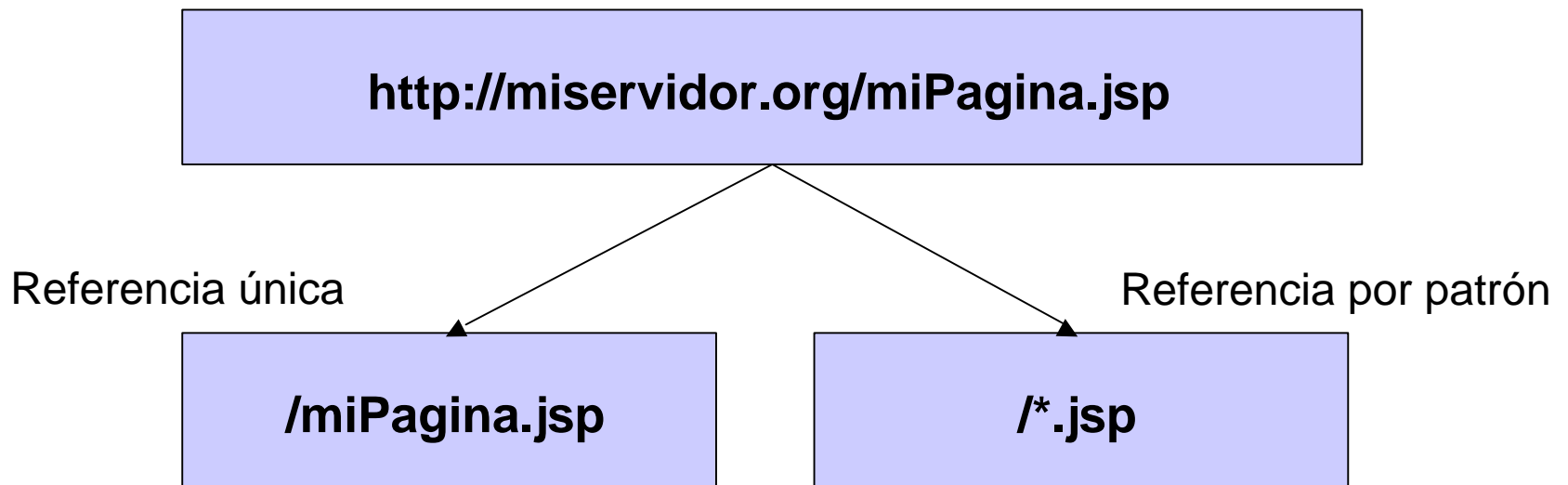
✍ `doFilter()`

✍ `destroy()`

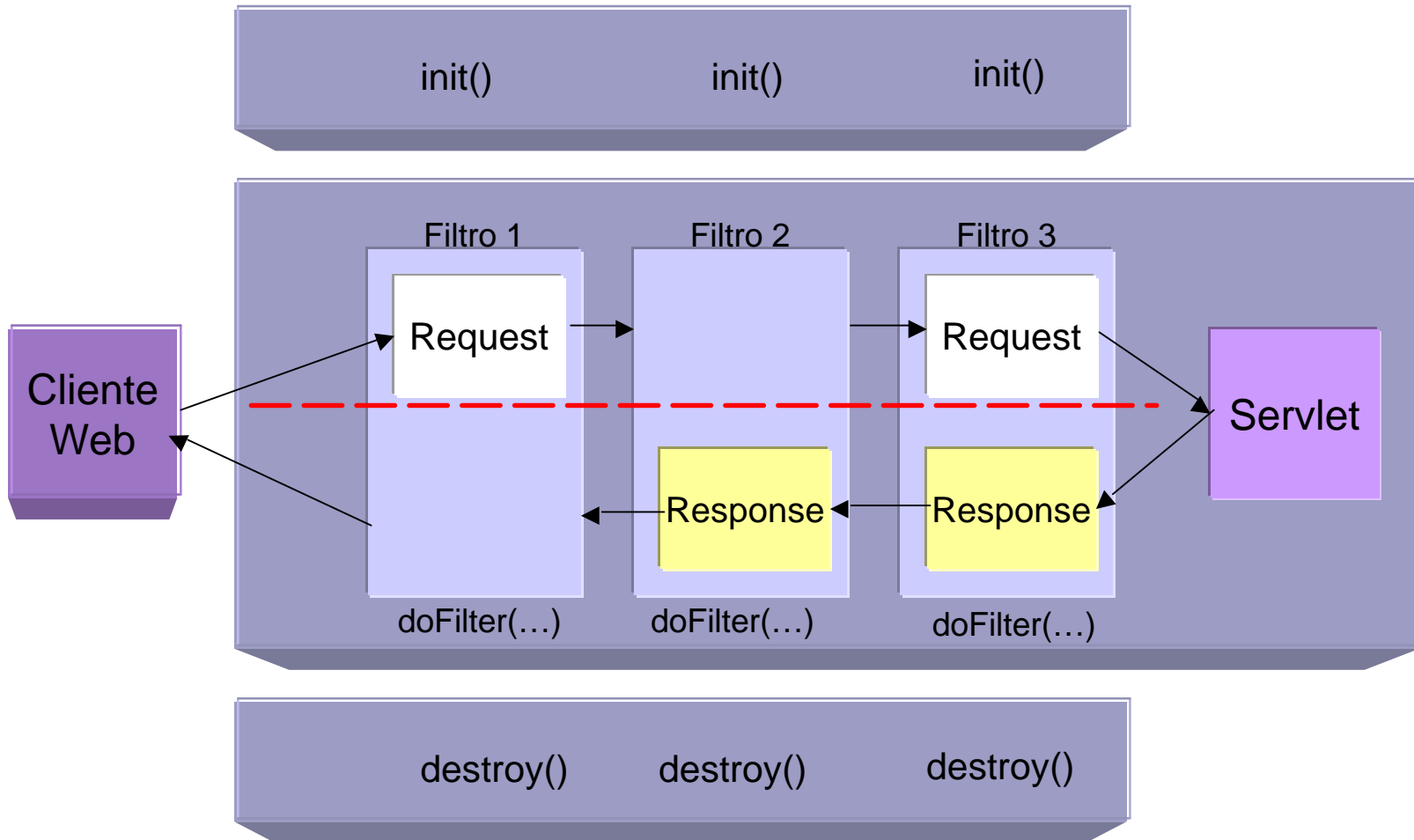
✍ La descripción del filtro se ubica en el archivo `web.xml`

# ¿Cómo funcionan? (2 de 3)

- ✍ Cada filtro responde a un Servlet o un URL que represente uno o más recursos



# ¿Cómo funcionan? (3 de 3)



# La interfaz `javax.servlet.FilterConfig`

Define cuatro métodos:

- ✍ **`getFilterName()`** : Retorna el nombre asociado al filtro en la configuración en un **String**
- ✍ **`getInitParameter(String)`** : Retorna el valor de un parámetro de configuración
- ✍ **`getInitParameterNames()`** : Retorna los nombres de los parámetros de configuración.
- ✍ **`getServletContext()`** : Retorna la referencia al **ServletContext** en el que trabajará el filtro

# La interfaz `javax.servlet.FilterChain`

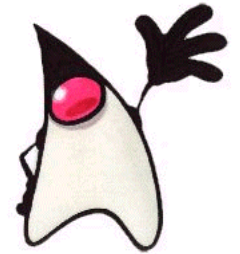
✍ Define un único método:

✍ `doFilter()` **throws** `ServletException`

✍ El método `doFilter` recibe como argumentos `ServletRequest` y `ServletResponse`.

✍ Es invocado por el contenedor cuando el filtro forma parte de la cadena creada ante la petición/respuesta de un recurso web.

# Ejemplo de Filtro



## ✍ Crear la clase MiFiltro

```
public class MiFiltro implements Filter{
    private FilterConfig filterConfig = null;
    public void init(FilterConfig filterConfig)
        throws ServletException {
        this.filterConfig = filterConfig;
    }
    public void doFilter(ServletRequest request,
        ServletResponse response,
        FilterChain chain)
        throws ServletException {
        chain.doFilter(request,response);
    }
    public void destroy(){
        this.filterConfig =null;
    }
}
```

**FilterChain** es el  
vinculo al siguiente  
filtro

# Cambiar el web.xml (1 de 3)

## ✍ Definir el filtro y sus parámetros

```
<filter>
  <filter-name>Filtro Aniversario Java</filter-name>
  <filter-class>MiFiltro</filter-class>
  <init-param>
    <param-name>aniversarioNumero</param-name>
    <param-value>10</param-value>
  </init-param>
</filter>
```

Nombre clase:  
Paquete.clase

# Cambiar el web.xml (2 de 3)

✍ Definir el patrón de recursos para el filtro

```
<filter-mapping>
  <filter-name>Filtro Aniversario Java</filter-name>
  <url-pattern>/miPagina.jsp</url-pattern>
</filter-mapping>
```

Solo miPagina.jsp

```
<filter-mapping>
  <filter-name>Filtro Aniversario Java</filter-name>
  <url-pattern>/* .jsp</url-pattern>
</filter-mapping>
```

Todos los recursos jsp

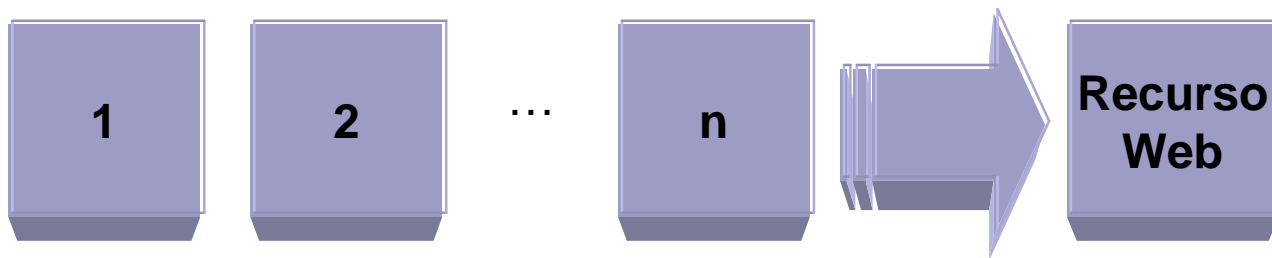
# Cambiar el web.xml (3 de 3)

## Mapping a un Servlet

```
<filter-mapping>  
  <filter-name>Filtro Aniversario Java</filter-name>  
  <servlet-name>MiServlet</ servlet-name >  
</filter-mapping>
```

# La Cadena de Filtros (1 de 4)

- ✍ Los filtros se incorporan a la cadena en el mismo orden en que aparecen en web.xml
- ✍ El último filtro de la cadena hará entrega de la solicitud al recurso web



# La Cadena de Filtros (2 de 4)

- ✍ La regla para seleccionar los miembros de la cadena es:
  - ✍ Primero obtener los filtros cuyo patrón URL coincida con el recurso solicitado,
  - ✍ Luego los que el nombre Servlet coincida con el Servlet solicitado.

# La Cadena de Filtros (3 de 4)

- Una misma *implementación*, puede ser utilizada para distintos recursos empleado nombres diferentes e incluso parámetros diferentes

```
<filter>
  <filter-name>FiltroJSP</filter-name>
  <filter-class>MiFiltro</filter-class>
  <init-param>
    <param-name>param</param-name>
    <param-value>10</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>FiltroJSP</filter-name>
  <url-pattern>/* .jsp</url-pattern>
</filter-mapping>
```

```
<filter>
  <filter-name>FiltroHTM</filter-name>
  <filter-class>MiFiltro</filter-class>
  <init-param>
    <param-name>param</param-name>
    <param-value>20</param-value>
  </init-param>
</filter>
<filter-mapping>
  <filter-name>FiltroHTM</filter-name>
  <url-pattern>/* .htm</url-pattern>
</filter-mapping>
```

# La Cadena de Filtros (4 de 4)

- ✍ Un elemento a considerar es la longitud de la cadena para un recurso o grupo de recursos específico.
- ✍ Debe evitarse el uso excesivo de filtros ya que impactarían en el tiempo de respuesta
- ✍ Uno o dos filtros suelen ser suficientes en la mayoría de los casos

# Reutilizar un Filtro

✍ Al utilizar la misma clase en diferentes filtros:

- ✍ Cada definición debe tener un nombre diferente en `<filter-name>`
- ✍ El contenedor creará una instancia por cada definición `<filter>` en el web.xml
- ✍ Las llamadas concurrentes activaran múltiples Threads para la instancia del filtro asociado

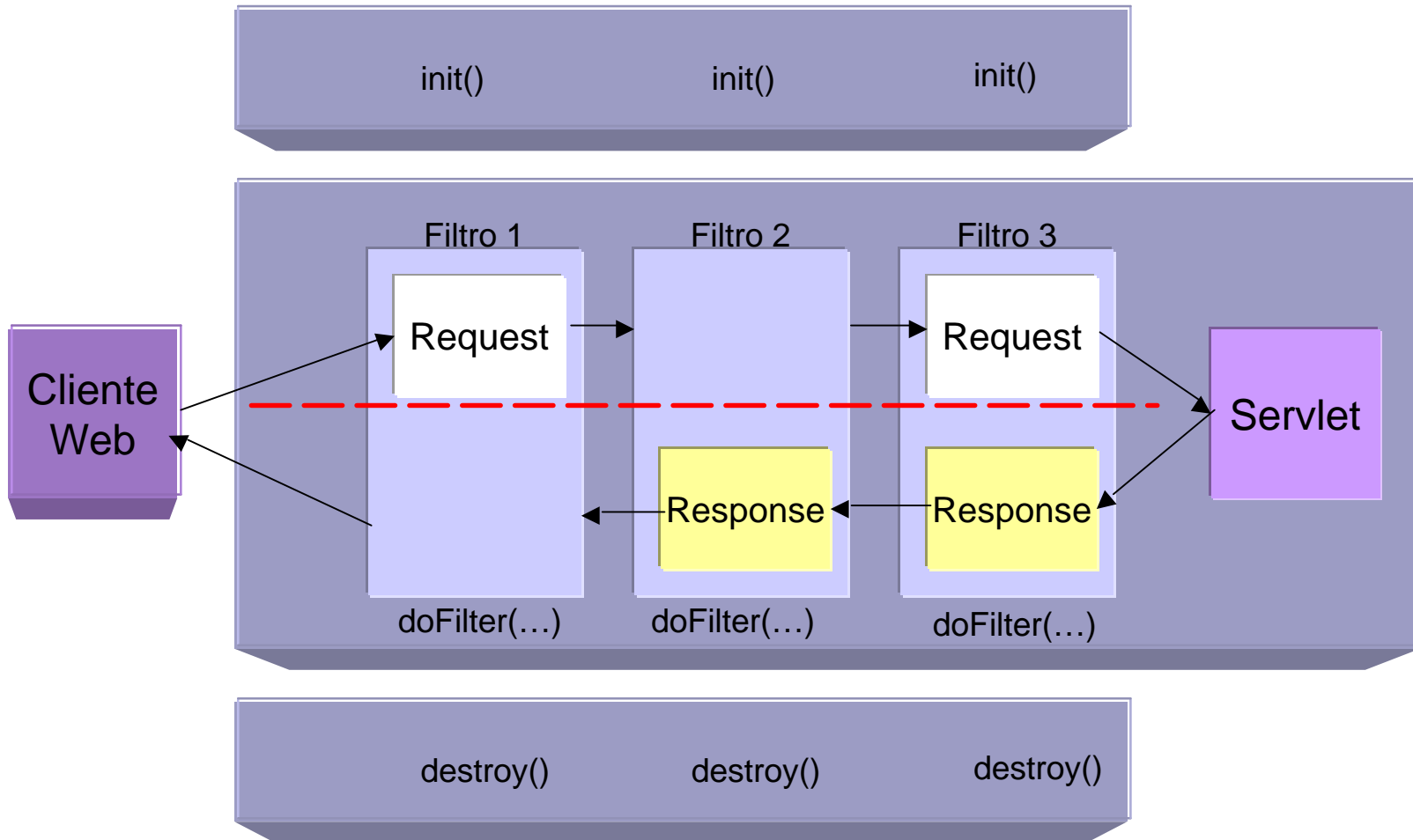
# Secuencia de ejecución (1 de 2)

✍ La ejecución de un filtro se divide en dos momentos: antes y después de invocar la cadena

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


    //Código previo a la cadena
    //Es el mejor lugar para modificar el request
    chain.doFilter(request,response); //Entrega el control al próximo de la cadena
    //Código posterior a la cadena
    //Este es el mejor lugar para modificar la respuesta.
}
```


# Secuencia de ejecución (2 de 2)



# Ejemplo: Control de Credenciales





## Filtro de Control de Credenciales (FCC)

 **Problema:** Un grupo específico de usuarios de una aplicación web requiere obtener sus credenciales desde dos repositorios diferentes, el resto de los usuarios solo requiere uno.

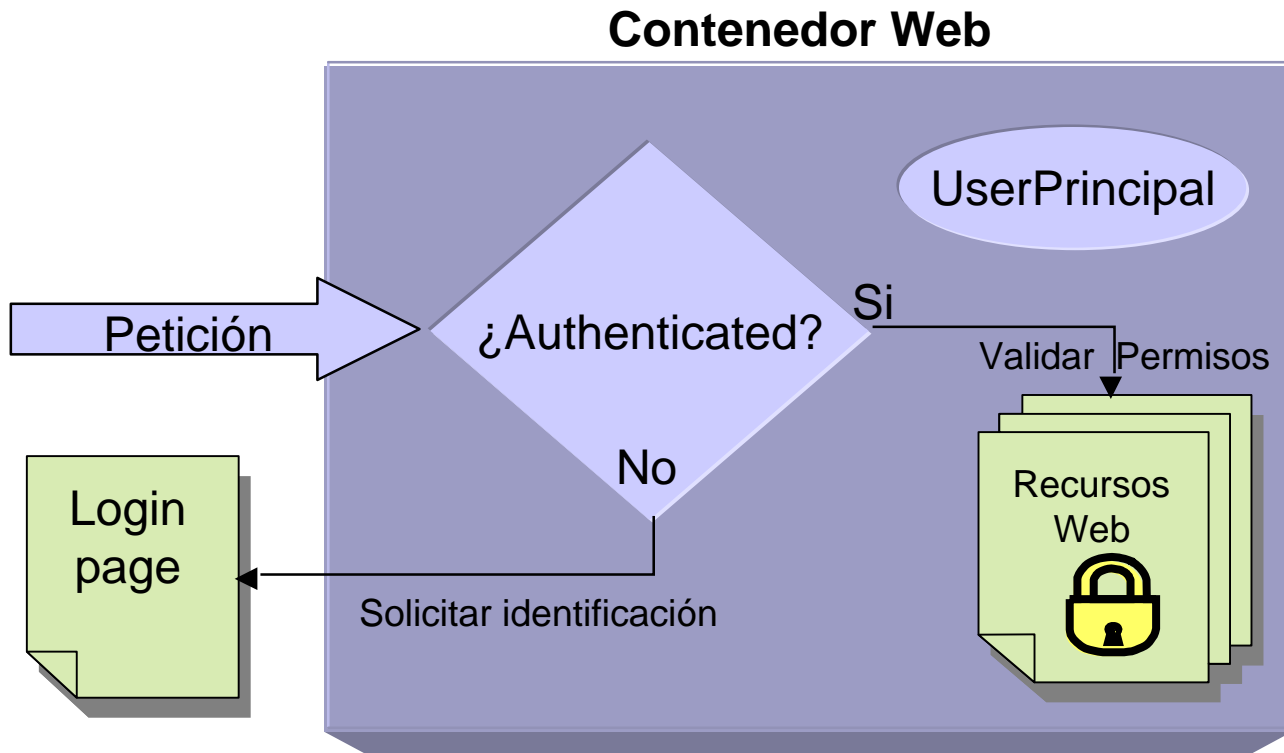
 **Propósito:** Asegurar la obtención de ambas credenciales sólo en caso de que el usuario pertenezca al grupo que lo requiere.

# ¿Por qué un filtro?

## Premisas de trabajo:

-  Debe ser desarrollado en Java
-  La seguridad de ser controlada por el contendor.
-  La solución debe ser fácil de implantar y retirar sin afectar el funcionamiento de la aplicación
-  La solución debe ser reutilizable

# Seguridad J2EE

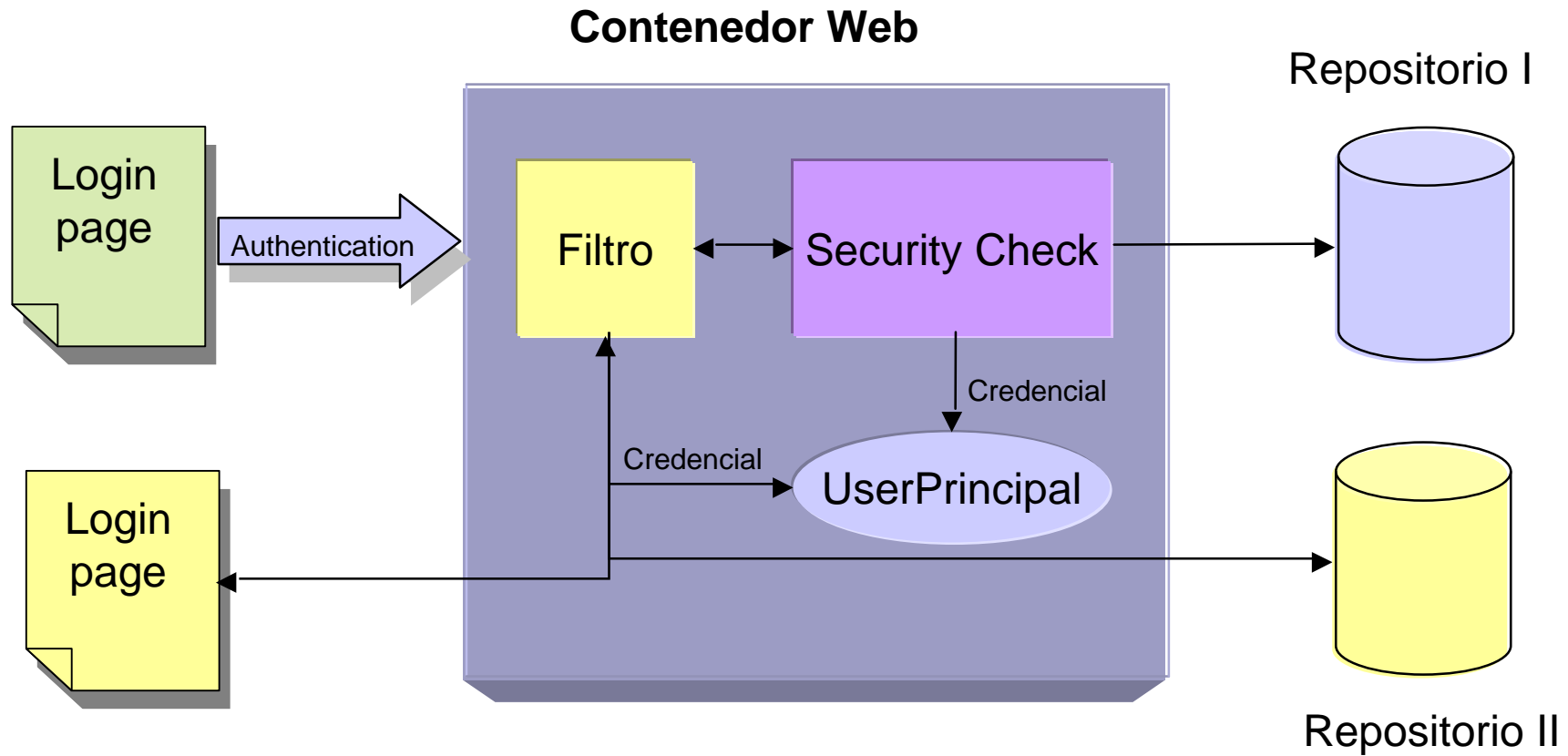


El cliente web realiza una petición, el contenedor verifica la identidad y si no existe ninguna solicita la identificación

# Acciones realizadas por el FCC

- ✍ Verifica si el usuario que intenta iniciar sesión pertenece al grupo de control,
- ✍ Si pertenece al grupo, desvía la petición a una segunda página de login. Si no, continua la cadena,
- ✍ Intenta obtener las credenciales contra el segundo repositorio
- ✍ Si tiene éxito, la credencial se almacena para uso futuro en la aplicación. Si falla, lo envía al inicio.
- ✍ Pasa el control a la cadena de filtros.

# Filtro Control de Credenciales



# Mapping Control de Credenciales

`<filter>`

`<filter-name>CtrlCredenciales</filter-name>`

`<filter-class>FilterClassName</filter-class>`

`</filter>`

`<filter-mapping>`

`<filter-name>CtrlCredenciales</filter-name>`

`<url-pattern>/j_security_check</url-pattern>`

`</filter-mapping>`

# Filtros Similares

✍ Un filtro muy útil y que se ha vuelto muy popular en aplicaciones para intranets basadas en Windows es el

**jcifs.http.NtlmHttpFilter**

✍ El filtro obtiene y valida las credenciales suministradas por el usuario al inicio de su sesión Windows simulando SSO

✍ Info: <http://jcifs.samba.org/>

# Resumen



- ✍ Los filtros son componentes reutilizables que forman parte de la especificación de Servlets desde la versión 2.3
- ✍ Trabajan en una cadena de filtros y pueden manipular la solicitud (request) y la respuesta (response)
- ✍ Implementan la interfaz `javax.servlet.Filter`
- ✍ Cada instancia del filtro puede ser manipulada por múltiples Threads
- ✍ Pueden ofrecer gran variedad de servicios

# Información en la red

✍ Java J2EE Specifications

<http://java.sun.com/j2ee>

✍ JCIFS y NtlmHttpFilter

<http://jcifs.samba.org/>

✍ Jacson, configurable text filtering

<http://jacson.sourceforge.net/filters.html>

✍ Caucho filter gallery

<http://209.47.15.67/resin-doc/servlet/filter-library.xtp>

# El grupo SoloJava



- ✍ Es el grupo de los programadores Java de Venezuela
- ✍ Fué creado en Agosto del año 2000 y en la actualidad cuenta con más de 300 usuarios activos y seguimos creciendo
- ✍ Sus direcciones en internet son:  
<http://espanol.groups.yahoo.com/group/solojava/> y  
<http://solojava.blogspot.com>