Filters
Listeners

Filters

Servlet Filters can be used for the following purposes:

•To intercept the requests coming from a client before they access a resource at back end.

•To manipulate responses from server before they are sent back to the client.

- can **intercept** and **redirect** processing
  - security
  - auditing
- can **modify requests and responses**
  - data conversion (XSLT, gzip, ...)
  - specialized caching

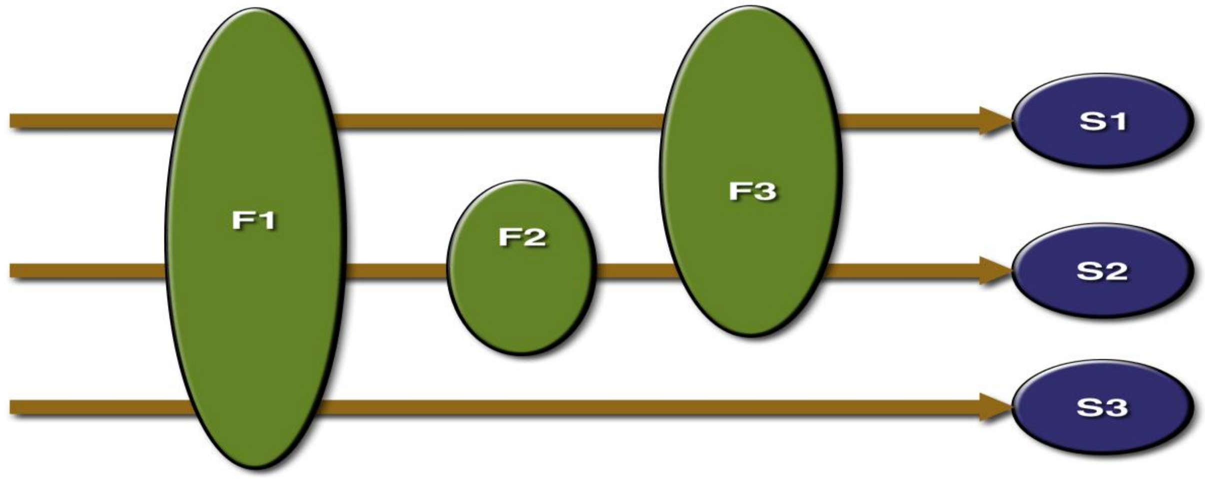
– *all without changing the existing servlet code!*

Filters

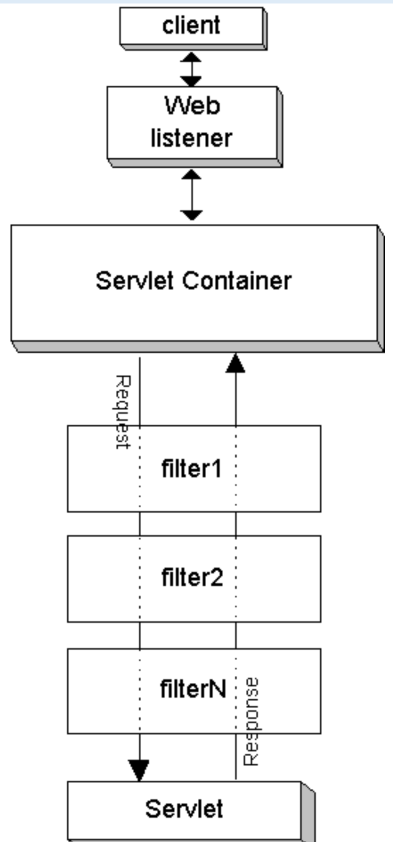
A common scenario for a filter is one in which you want to apply pre-processing or post-processing to requests or responses for a group of servlets, not just for a single servlet.

If you need to modify the request or response for just one servlet, there is no need to create a filter—just do what is required directly in the servlet itself.

We can map a filter to one or more Web resources, and you can map more than one filter to a web resource as shown below:



Filter F1 is mapped to Servlets S1, S2, and S3, filter F2 is mapped to Servlet S2, and filter F3 is mapped to Servlets S1 and S2.



Filter Lifecycle

When the web application starts up, the Servlet Container will create an instance of the filter and keeps it in memory during web application's lifetime.

The same instance will be reused for every incoming request whose URL matches the filter's URL pattern. The **doFilter()** method will then be called on every request.

Filters are not Servlets.

So they do not implement and override HttpServlet methods such as doGet() or doPost().

Rather, a filter implements the methods of the **javax.servlet.Filter** interface.

The methods are:

- init()
- destroy()
- doFilter()

Filter API

The Filter API consists of three interfaces:
1.javax.servlet.Filter
2.javax.servlet.FilterChain
3.javax.servlet.FilterConfig

Methods in **javax.servlet.Filter interface:**

Method	**Description**
public void **init(FilterConfig filterConfig)**	Called by the web container to indicate to a filter that it is being placed into service.
public void **doFilter(ServletRequest request, ServletResponse response, FilterChain chain)**	The doFilter() method is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain.
public void **destroy()**	Called by the web container to indicate to a filter that it is being taken out of service.

Note : To create a filter class you must implement the ***javax.servlet.Filter*** interface and define all the above methods.

javax.servlet.FilterConfig interface

The **javax.servlet.FilterConfig** interface is an argument to the ***init() method.***

This interface contains information about **initial parameters** and provides access to the **ServletContext** as well.

Methods in **javax.servlet.FilterConfig interface**

Method	**Description**
public String **getFilterName()**	Returns the filter-name of this filter as defined in the deployment descriptor.
public ServletContext **getServletContext()**	Returns a reference to the **ServletContext** in which the caller is executing.
public String **getInitParameter(String name)**	Returns a String containing the value of the named initialization parameter, or null if the parameter does not exist.
public Enumeration **getInitParameterNames()**	Returns the names of the filter's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the filter has no initialization parameters.

javax.servlet.FilterChain interface

The **javax.servlet.FilterChain** interface is an argument to the ***doFilter() method.***

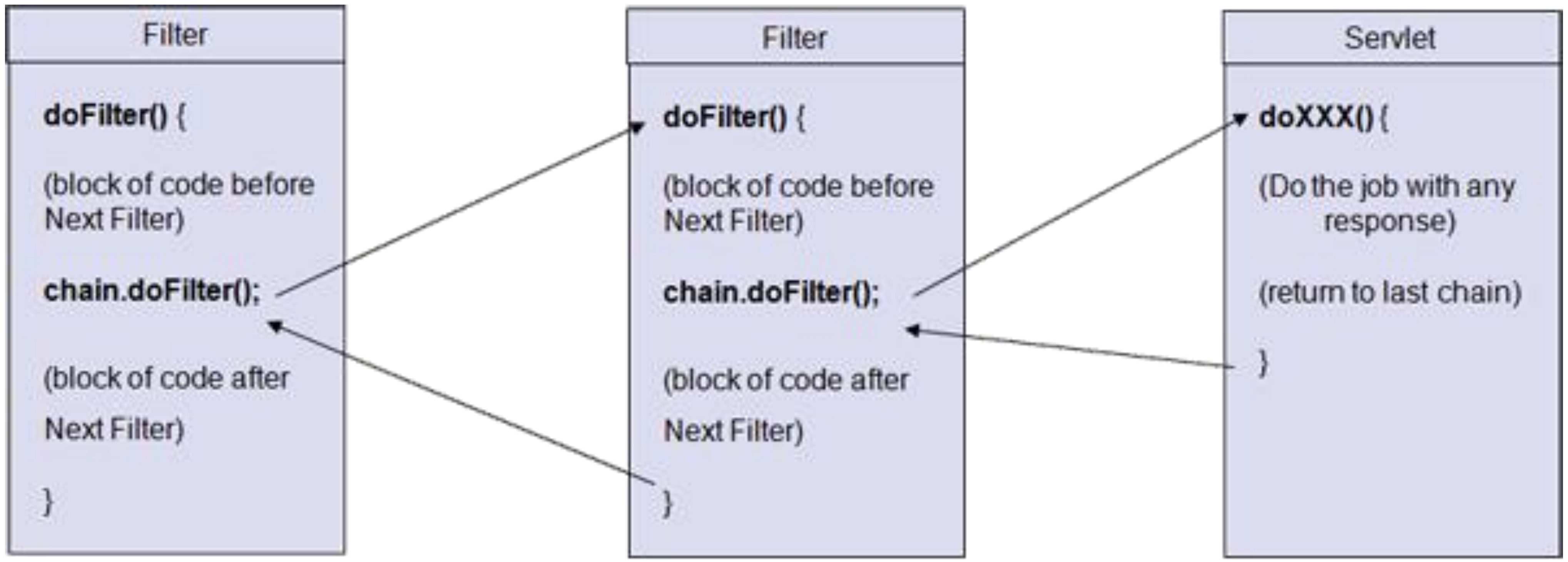
This interface has only one method, **doFilter(),** which causes the next filter in the chain to be invoked.

Methods in **javax.servlet.FilterChain interface**

Method	**Description**
public void **doFilter(ServletRequest request, ServletResponse response)**	Causes the next filter in the chain to be invoked, or if the calling filter, is the last filter in the chain, causes the resource at the end of the chain to be invoked.

Filter Chain

The filter chain


Filter
doFilter() {
(block of code before Next Filter)
chain.doFilter();
(block of code after Next Filter)
}
Filter
doFilter() {
(block of code before Next Filter)
chain.doFilter();
(block of code after Next Filter)
}
Servlet
doXXX() {
(Do the job with any response)
(return to last chain)
}

Using @WebFilter Annotation

@WebFilter has the following attributes :

filterName

description

displayName

initParams

servletNames

value

urlPatterns

dispatcherTypes

asyncSupported

Ex.

```
@WebFilter(filterName="/LogA",urlPatterns="/LoginServlet",
      initParams={@WebInitParam(name="message",value="Hello“) }
)
```

Listeners

- The Servlet specification includes the capability to track key events in your web applications through *event listeners*. This functionality allows more efficient resource management and automated processing based on event status.

- Listeners introduced in Servlet 2.3 API recognize application events which help web developers in controlling the life cycle of **the ServletContext, ServletRequest, and HttpSession objects**.

- Servlet Listeners are used to listen to the **key events** in a web container, such as web application initialization/shutdown , create/invalidate a session, or add/remove an attribute in an session or manage number of concurrent users etc.

- In design pattern terms – observer pattern: An observer (in this case the listener) is notified when an event occurs in the subject (server).

Note:
Use a Listener if you want to intercept key changes in the lifecycle of objects.

Event Types and corresponding Listener Interfaces

Scope Level Events	Event Type	Description	Listener Interface	Methods
Application Level Events	Lifecycle events	Occur when either a servlet context is initialized or destroyed	ServletContext Listener	void contextInitialized(ServletContextEvent) void contextDestoyed(ServletContextEvent)
	Changes to ServletContext attributes	Occur when a new attribute of the ServletContext interface is added or an existing attribute is removed or replaced by	ServletContext AttributeListen er	void attributeAdded(ServletContextAttributeEvent) void attributeReplaced(ServletContextAttributeEvent) void attributeRemoved(ServletContextAttributeEvent)

Event Types and corresponding Listener Interfaces

Scope Level Events	Event Type	Description	Listener Interface	Methods
Session Level Events	Lifecycle events	Occur when an instance of HttpSession is created, invalidated, or destroyed	HttpSessionListener	void sessionCreated(HttpSessionEvent) void sessionDestroyed(HttpSessionEvent)
	Changes to HttpSession attributes	Occur when a new attribute of the HttpSession instance is added or an existing attribute is removed or replaced by another attribute	HttpSessionAttribute Listener	void attributeAdded(HttpSessionBindingEvent void attributeReplaced(HttpSessionBindingEve void attributeRemoved(HttpSessionBindingEve

Event Types and corresponding Listener Interfaces

Scope Level Events	Event Type	Description	Listener Interface	Methods
Request Level Events	Lifecycle events	Occur when a servlet processes a ServletRequest instance	ServletRequestListener	void requestInitialized(ServletRequestEvent) void requestDestroyed(ServletRequestEvent)
	Changes to ServletRequest attributes	Occurs when a new attribute is added or an existing attribute is removed or replaced by another attribute of ServletRequest instance	ServletRequestAttributeListener	void attributeAdded(ServletRequestAttributeEvent) void attributeReplaced(ServletRequestAttributeEvent)

Implementing Listener Interfaces

In a web application, multiple listeners may co-exist.

To apply listeners to our web application, perform the following:

1. Create a class that implements the appropriate Listener interface
2. Register the listener in the deployment descriptor file (web.xml) or use Annotations.

Note: Listeners are implicitly instantiated by the Servlet Container when the web application is loaded.

ServletContextListener Example

Note: Use either annotation or declare in web.xml

```
@WebListener
public class MyContextListener implements ServletContextListener {
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext context = sce.getServletContext();
        System.out.println("ServletContext is initialized");
     }
public void contextDestroyed(ServletContextEvent sce) {
sce.getServletContext().log("Servlet Context object destroyed");    }
}
```

web.xml

```
<listener>
        <listener-class>
                org.asr.listeners. MyContextAttributeListener
        </listener-class>
</listener>
```

Output on the server console when you run the application and after stopping the server.

```
ServletContext is initialized
…….
INFO: Server startup in 455 ms
…………
INFO: Stopping service Catalina
ServletContext is about to be destroyed
```

ServletRequestListener Interface

The **ServletRequestListener** allows the developers to monitor the request coming in and out of scope in a web component. For example we can count the number of requests our Servlet receives by incrementing a static variable.

The **ServletRequestListener** has the following two methods:

1. public void requestInitialized(ServletRequestEvent e)

This method will be executed automatically by the web container at the time of request object creation i.e. just before invoking service method.

2.public void requestDestroyed(ServletRequestEvent e)

This method will be executed by the web container at the time of request object destroy i.e. just after completion of service ().

The **ServletRequestEvent** class which is a sub class *of java.util.Event* defines the following two methods:

1. public ServletRequest getServeltRequest() - returns the ServletRequest that is changing.

2.public ServletContext getServeltContext() - returns the ServletContext of this web application.

ServletRequestListener Example

```
@WebListener
public class ServletRequestListenerDemo implements ServletRequestListener {
    public static int count=0;
    public void requestDestroyed(ServletRequestEvent  sre) {
            System.out.print("The Request object destroyed at  :"+new java.util.Date());
    }
     public void requestInitialized(ServletRequestEvent  sre) {
                count++;
                 System.out.print("Request Object created At:"+ new java.util.Date());
                 System.out.print("The hit count for this web application :"+count);
    }
}
```

```
@WebServlet("/HitCountServlet")
public class HitCountServlet extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
      PrintWriter out = response.getWriter();
      out.print("<h2> This is target Servlet </h2>");
      out.print("<h2>No. of hits for this application :"+ServletRequestListenerDemo.count);
              }
}
```

Thank You!