Servlet Filters

A **filter** is an object that is invoked at the preprocessing and postprocessing of a request.

It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

Filters differ from web components in that filters usually do not themselves create a response. Instead, a filter provides functionality that can be “attached” to any kind of web resource. Consequently, a filter should not have any dependencies on a web resource for which it is acting as a filter; this way, it can be composed with more than one type of web resource.

The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.

So maintenance cost will be less.

Servlet Filters

The main tasks that a filter can perform are as follows:

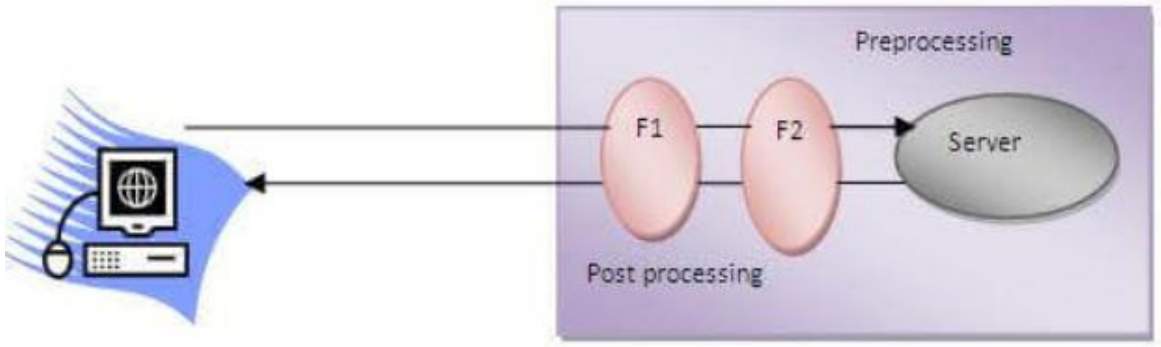
Query the request and act accordingly.

Block the request-and-response pair from passing any further.

Modify the request headers and data. You do this by providing a customized version of the request.

Modify the response headers and data. You do this by providing a customized version of the response.

Interact with external resources.



Working of filters

When a request reaches the **Web Container**, it checks if any filter has URL patterns that matches the requested URL.

The **Web Container** locates the first filter with a matching URL pattern and filter's code is executed.

If another filter has a matching URL pattern, its code is then executed. This continues until there are no filters with matching URL patterns left.

If no error occurs, the request passes to the target servlet. Hence we know, that the request will be passed to the target servlet only when all the related Filters are successfully executed.

The servlet returns the response back to its caller. The last filter that was applied to the request is the first filter applied to the response.

At last the response will be passed to the **Web Container** which passes it to the client.

Declaring a Servlet Filter inside Deployment Descriptor

```
<web-app ...>
<filter>
<filter-name>MyFilter</filter-name>
<filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>..</url-pattern> or <servlet-name>.</servlet-name>
</filter-mapping>
</web-app>
```

<filter-name> tag is used to give a internal name to your filter

<filter-class> declares the filter that you have created

Either the <url-pattern> or the <servlet-name> element is mandatory which web app resource will use this filter

Filter API

Like servlet filter have its own API. The javax.servlet package contains the interfaces of Filter API.

The filter API brings in all the required interfaces for creating a filter class into the javax.servlet package. It contains three interfaces which create and handle the functionalities of a filter. These are

Filter interface.

FilterConfig interface

FilterChain interface.

Filter Interface

The filter interface is mandatory for creating the filter component in the Web container. Create a class that implements this interface so that it can gain access over the methods in the interface.. Some of the methods in the *Filter* interface are,

init()

This is called by the servlet container to initialize the filter. It is called once.

public void init(FilterConfig filterConfig)throws ServletException

doFilter()

The doFilter() method of the Filter interface is called by the coordinator each time a request or response is processed.

public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain)throws IOException , ServletException

destroy()

This method is called by the servlet container to inform the filter that its service is no longer required.

public void destroy()

Filter Config Interface

The FilterConfig interface is vital to initialize the filter. The initialization of filters is done when init() method is called. The init() method retrieves the initial parameters from the FilterConfig object, which is passed to it as argument. It has the following methods.

getFilterName()

This method returns the name of filter defined in the deployment descriptor file web.xml in web application. Syntax : public String getFilterName().

getInitParameter()

This method returns the value of the named initialization parameter as a string.. It returns null if the servlet has no attributes.

Syntax - public String getInitParameter(String name)

Filter Config Interface

getInitParameterNames()

This method returns the names of the initialization parameter as an enumeration of String objects.

Syntax - public Enumeration getInitParameterNames()

getServletContext()

This method returns the ServletContext object used by the caller too interact with its servlet container and filter.

Syntax - public ServletContext getServletContext()

Filter Chain Interface

This interface supports the concepts of filter chain in an application. An object of FilterChain has all the information about the sequence of control flow among a set of filters in a filter chain from the web.xml file.

It contains the doFilter (request, response) method, which invokes the next filter in the line of action in a Web application.

The doFilter(request, response) method is invoked by an object of the FilterChain interface. This is not a standalone program. It needs servlet to run.

A filter is a servlet component that provides some useful service to the request and response. A filter works as an interface between the client and the web application and performs intercept requests, processes the wrapped request, sends requests to servlet, and sends responses. The filter API brings in all the required interfaces for creating a filter class into the javax.servlet package. The filter interface is mandatory for creating the filter component in the Web container. The FilterConfig interface is vital to initialize the filter.

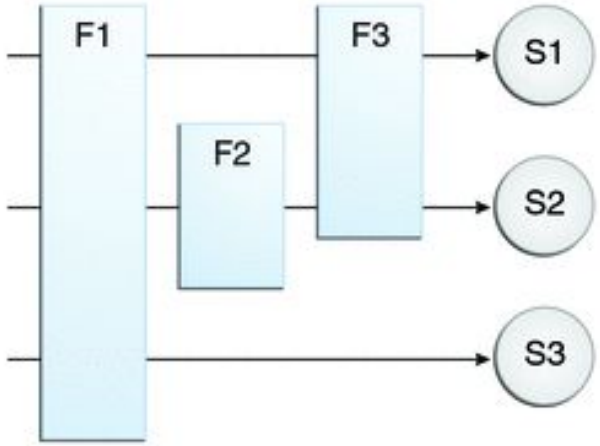
Filter Mappings

A web container uses filter mappings to decide how to apply filters to web resources. A filter mapping matches a filter to a web component by name or to web resources by URL pattern. The filters are invoked in the order in which filter mappings appear in the filter mapping list of a WAR. You specify a filter mapping list for a WAR in its deployment descriptor by either using NetBeans IDE or coding the list by hand with XML.

If you want to log every request to a web application, you map the hit counter filter to the URL pattern /*.

You can map a filter to one or more web resources, and you can map more than one filter to a web resource. This is illustrated in Figure below, where 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 Mappings



Recall that a filter chain is one of the objects passed to the doFilter method of a filter. This chain is formed indirectly by means of filter mappings. The order of the filters in the chain is the same as the order in which filter mappings appear in the web application deployment descriptor.

When a filter is mapped to servlet S1, the web container invokes the doFilter method of F1. The doFilter method of each filter in S1's filter chain is invoked by the preceding filter in the chain by means of the chain.doFilter method. Because S1's filter chain contains filters F1 and F3, F1's call to chain.doFilter invokes the doFilter method of filter F3. When F3's doFilter method completes, control returns to F1's doFilter method.

Accessing the Web Context

The context in which web components execute is an object that implements the ServletContext interface. You retrieve the web context by using the getServletContext method. The web context provides methods for accessing

Initialization parameters

Resources associated with the web context

Object-valued attributes

Logging capabilities

The counter's access methods are synchronized to prevent incompatible operations by servlets that are running concurrently. A filter retrieves the counter object by using the context's getAttribute method. The incremented value of the counter is recorded in the log.

ServletContext Interface

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

Easy to maintain if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem

Usage of ServletContext Interface

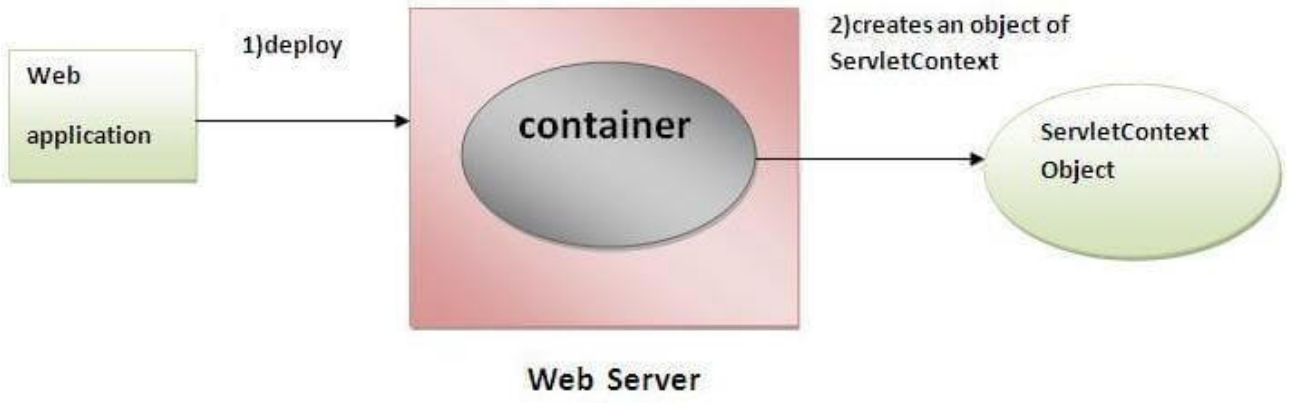
There can be a lot of usage of ServletContext object. Some of them are as follows:

The object of ServletContext provides an interface between the container and servlet.

The ServletContext object can be used to get configuration information from the web.xml file.

The ServletContext object can be used to set, get or remove attribute from the web.xml file.

The ServletContext object can be used to provide inter-application communication.



Commonly used methods of ServletContext interface

There is given some commonly used methods of ServletContext interface.**public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.

public Enumeration getInitParameterNames(): Returns the names of the context's initialization parameters.

public void setAttribute(String name,Object object):sets the given object in the application scope.

public Object getAttribute(String name): Returns the attribute for the specified name.

public Enumeration getInitParameterNames(): Returns the names of the context's initialization parameters as an Enumeration of String objects.

public void removeAttribute(String name): Removes the attribute with the given name from the servlet context.

Maintaining Client State

Many applications require that a series of requests from a client be associated with one another. For example, a web application can save the state of a user's shopping cart across requests.

Web-based applications are responsible for maintaining such state, called a session, because HTTP is stateless. To support applications that need to maintain state, Java Servlet technology provides an API for managing sessions and allows several mechanisms for implementing sessions.

Accessing a Session

Sessions are represented by an HttpSession object. You access a session by calling the getSession method of a request object. This method returns the current session associated with this request; or, if the request does not have a session, this method creates one.

Accessing object with a session

You can associate object-valued attributes with a session by name. Such attributes are accessible by any web component that belongs to the same web context and is handling a request that is part of the same session.

Recall that your application can notify web context and session listener objects of servlet lifecycle events (Handling Servlet Lifecycle Events). You can also notify objects of certain events related to their association with a session such as the following:

- When the object is added to or removed from a session. To receive this notification, your object must implement the javax.servlet.http.HttpSessionBindingListener interface.
- When the session to which the object is attached will be passivated or activated. A session will be passivated or activated when it is moved between virtual machines or saved to and restored from persistent storage. To receive this notification, your object must implement the javax.servlet.http.HttpSessionActivationListener interface.

Session Management

Because an HTTP client has no way to signal that it no longer needs a session, each session has an associated timeout so that its resources can be reclaimed. The timeout period can be accessed by using a session's getMaxInactiveInterval and setMaxInactiveInterval methods.

To ensure that an active session is not timed out, you should periodically access the session by using service methods because this resets the session's time-to-live counter. When a particular client interaction is finished, you use the session's invalidate method to invalidate a session on the server side and remove any session data.

To Set the Timeout Period Using NetBeans IDE, To set the timeout period in the deployment descriptor using NetBeans IDE, follow these steps.

- Open the project if you haven't already.
- Expand the project's node in the Projects pane.
- Expand the Web Pages node and then the WEB-INF node.
- Double-click web.xml.
- Click General at the top of the editor.
- In the Session Timeout field, type an integer value.

Session Tracking

To associate a session with a user, a web container can use several methods, all of which involve passing an identifier between the client and the server. The identifier can be maintained on the client as a cookie, or the web component can include the identifier in every URL that is returned to the client.

If your application uses session objects, you must ensure that session tracking is enabled by having the application rewrite URLs whenever the client turns off cookies. You do this by calling the response's encodeURL(URL) method on all URLs returned by a servlet. This method includes the session ID in the URL only if cookies are disabled; otherwise, the method returns the URL unchanged.

Finalizing a servlet

The web container may determine that a servlet should be removed from service (for example, when a container wants to reclaim memory resources or when it is being shut down). In such a case, the container calls the destroy method of the Servlet interface. In this method, you release any resources the servlet is using and save any persistent state. The destroy method releases the database object created in the init method .

A servlet's service methods should all be complete when a servlet is removed. The server tries to ensure this by calling the destroy method only after all service requests have returned or after a server-specific grace period, whichever comes first. If your servlet has operations that may run longer than the server's grace period, the operations could still be running when destroy is called. You must make sure that any threads still handling client requests complete.

Following things are doe to finalize a servlet:

- Keep track of how many threads are currently running the service method.
- Provide a clean shutdown by having the destroy method notify long-running threads of the shutdown and wait for them to complete.
- Have the long-running methods poll periodically to check for shutdown and, if necessary, stop working, clean up, and return.