Java Servlet Filter

Java Servlet Filter is used to intercept request and do some pre-processing and can be used to intercept response and do post-processing before sending to client in web application.

- **Why do we have Servlet Filter?**

Servlet Filters are **pluggable** java components that we can use to intercept and process requests *before* they are sent to servlets and response *after* servlet code is finished and before container sends the response back to the client.
Some common tasks that we can do with servlet filters are:

- Logging request parameters to log files.
- Authentication and autherization of request for resources.
- Formatting of request body or header before sending it to servlet.
- Compressing the response data sent to the client.
- Alter response by adding some cookies, header information etc.

Servlets and filters both are unaware of each other and we can add or remove a servlet filter just by editing web.xml.
We can have multiple filters for a single resource and we can create a chain of filters for a single resource in web.xml. We can create a Servlet Filter by implementing javax.servlet.Filter interface.

- **Servlet Filter interface**

Servlet Filter interface is similar to Servlet interface and we need to implement it to create our own servlet filter. Servlet Filter interface contains lifecycle methods of a Filter and it's managed by servlet container.
Servlet Filter interface lifecycle methods are:

1. **void init(FilterConfig paramFilterConfig)** – When container initializes the Filter, this is the method that gets invoked. This method is called only once in the lifecycle of filter and we should initialize any resources in this method. **FilterConfig** is used by container to provide init parameters and servlet context object to the Filter. We can throw ServletException in this method.
2. **doFilter(ServletRequest paramServletRequest, ServletResponse paramServletResponse, FilterChain paramFilterChain)** – This is the method invoked every time by container when it has to apply filter to a resource. Container provides request and response object references to filter as argument. **FilterChain** is used to invoke the next filter in the chain. This is a great example of **Chain of Responsibility Pattern**.
3. **void destroy()** – When container offloads the Filter instance, it invokes the destroy() method. This is the method where we can close any resources opened by filter. This method is called only once in the lifetime of filter.

- **Servlet WebFilter annotation**

javax.servlet.annotation.WebFilter was introduced in Servlet 3.0 and we can use this annotation to declare a servlet filter. We can use this annotation to define init parameters, filter name and description, servlets, url patterns and dispatcher types to apply the filter. If you make frequent changes to the filter configurations, its better to use web.xml because that

will not require you to recompile the filter class.

- **Servlet Filter configuration in web.xml**

We can declare a servlet filter in web.xml like below.

```
<filter>
  <filter-name>RequestLoggingFilter</filter-name> <!-- mandatory -->
  <filter-class>com.journaldev.servlet.filters.RequestLoggingFilter</filter-class>
  <init-param> <!-- optional -->
  <param-name>test</param-name>
  <param-value>testValue</param-value>
  </init-param>
</filter>
```

We can map a Filter to servlet classes or url-patterns like below.

```
<filter-mapping>
  <filter-name>RequestLoggingFilter</filter-name> <!-- mandatory -->
  <url-pattern>/*</url-pattern> <!-- either url-pattern or servlet-name is mandatory -->
  <servlet-name>LoginServlet</servlet-name>
  <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

While creating the filter chain for a servlet, container first processes the url-patterns and then servlet-names, so if you have to make sure that filters are getting executed in a particular order, give extra attention while defining the filter mapping.
Servlet Filters are generally used for client requests but sometimes we want to apply filters with **RequestDispatcher** also.