**Servlet Filters**

Table of Contents

[Configuration 3](#_Toc100555750)

[API 5](#_Toc100555751)

[Filter 5](#_Toc100555752)

[FilterConfig 6](#_Toc100555753)

[FilterChain 7](#_Toc100555754)

[Use Cases 8](#_Toc100555755)

We previously mentioned a problem where a user could directly access some resource by changing the URL in their browser, thus **bypassing the verification process** of servlets. To deal with this problem, the proposed solution was to generate the HTML code using a PrintWriter object directly in the servlet. However, this quickly becomes a troublesome process as our resources will not always be so easy to generate like this.

A better solution would be to **intercept all new requests** and verify the user before giving them access to resources or servlets. This is achieved by using **servlet filters**. Servlet filters have other use cases as well, such as to log the IP addresses of incoming requests, input validation, etc. However, filters are not used for input validation nowadays because it can be handled easily on the client-end using JavaScript.

Servlet filters can be used for **pre-processing** when we receive requests as well as **post-processing** before sending a response. However, we will only be looking into pre-processing.

## Configuration

We can configure servlet filters using the web.xml file or using **annotations**, just like servlets.

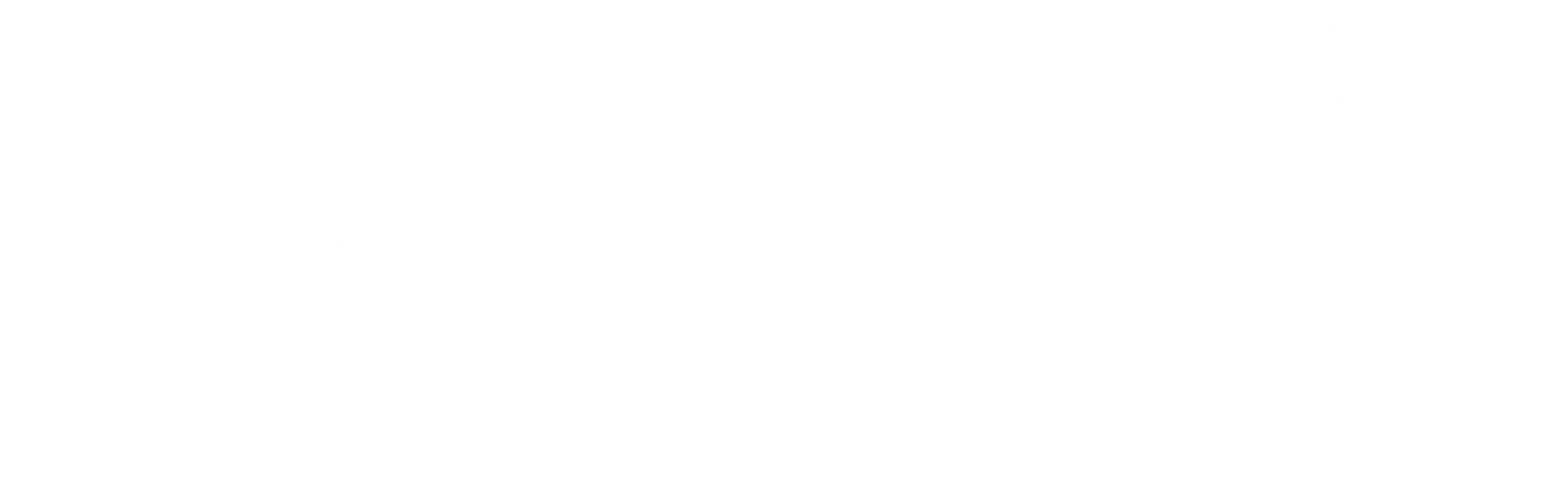
<web-app>  
 <servlet>  
 <servlet-name>FilteredServlet</servlet-name>  
 <servlet-class>FilteredServlet</servlet-class>  
 </servlet>

<servlet>  
 <servlet-name>FilteredServlet2</servlet-name>  
 <servlet-class>FilteredServlet2</servlet-class>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>FilteredServlet</servlet-name>  
 <url-pattern>/FilteredServlet</url-pattern>  
 </servlet-mapping>  
 <servlet-mapping>  
 <servlet-name>FilteredServlet2</servlet-name>  
 <url-pattern>/FilteredServlet2</url-pattern>  
 </servlet-mapping>  
 <filter>  
 <filter-name>BasicFilter</filter-name>  
 <filter-class>BasicFilter</filter-class>  
 </filter>  
 <filter-mapping>  
 <filter-name>BasicFilter</filter-name>  
 <servlet-name>FilteredServlet</servlet-name>  
 </filter-mapping>  
 <filter-mapping>  
 <filter-name>BasicFilter</filter-name>  
 <servlet-name>FilteredServlet2</servlet-name>  
 </filter-mapping>  
 <filter-mapping>  
 <filter-name>BasicFilter</filter-name>  
 <url>/\*</url>  
 </filter-mapping>  
</web-app>

XML

Just like servlets, we declare a filter element where we declare its name along with the name of the class which will contain the code for the filter. Next, we have the filter-mapping element, which maps the filter to the servlet or resource that will use it. Notice that we can have multiple mappings for a single filter, since the filter can work with multiple resources.

Consider the diagram below:



When the servlet container receives a new request from some client, the request may go through a series of filters in a specified order. This series of filters is defined by a *FilterChain* object, which we will look into in more depth later. The **order** in which the filters are applied is defined by the order in which the filters are declared in the web.xml file. **Which filters** are applied to each servlet or resource is defined by the mappings.

However, note that the filters for a servlet are only applied when a **client** makes a request. They are not applied if another servlet forwards the request. In that situation, if we want to use a filter, we must [apply it manually](https://stackoverflow.com/questions/31065687/servlet-filter-forwarded-request-from-a-servlet-will-go-to-servlet-filter-or-n).

## API

We will mainly be using three interfaces when working with servlet filters, *Filter*, *FilterConfig* and *FilterChain*.

### Filter

*Filter* objects have three methods, init(), doFilter() and destroy().

The init() method basically initializes the *Filter* object. It takes a single parameter, a *FilterConfig* object. This contains the **initialization parameters** for the *Filter* object.

public void init(*FilterConfig* config) throws ServletException {  
 devName = config.getInitParameter("devName");  
 devPhone = config.getInitParameter("devPhone");  
}

JAVA

The initialization parameters can be defined in the web.xml file when declaring the filter.

<filter>  
 <filter-name>BasicFilter</filter-name>  
 <filter-class>BasicFilter</filter-class>  
 <init-param>  
 <param-name>devName</param-name>  
 <param-value>John Doe</param-value>  
 </init-param>  
 <init-param>  
 <param-name>devPhone</param-name>  
 <param-value>0123456789</param-value>  
 </init-param>  
</filter>

XML

The doFilter() method is called every time a client makes a request for some resource to which the filter is applied. This is where the actual **filtering** happens.

The destroy() method is called at the end of the *Filter* object’s lifecycle.

### FilterConfig

We already discussed the use of *FilterConfig* objects above. These objects have several methods, such as getFilterName(), getServletContext(), getInitParameterNames() and getInitParameter("someParameter").

### FilterChain

A *FilterChain* object contains the **chain of filters** which must be applied before the client gets the actual requested resource. It has a single method, doFilter(), which invokes the next filter in the chain or the resource as required.

Note that this is different from the doFilter() method of a *Filter* object. A *Filter* object uses the doFilter() method of the *FilterChain* object that is passed to it as a parameter in its own doFilter() method to invoke the next filter. Failure to call the doFilter() method of the *FilterChain* object will cause the program to stop.

public void doFilter(*ServletRequest* request, *ServletResponse* response, *FilterChain* chain) throws ServletException, IOException {  
 // logging client's IP address  
 *ServletContext* servletContext = request.getServletContext();  
 servletContext.log(request.getRemoteHost());  
 chain.doFilter(request, response);  
}

JAVA

Also note that the doFilter() method of a *Filter* object uses *ServletRequest* and *ServletResponse* objects as parameters, which need to be **typecast** to *HttpServletRequest* and *HttpServletResponse* objects respectively before invoke the doFilter() method of the *FilterChain* object.

## Use Cases

Filters allow us to encapsulate recurring tasks in reusable units. They allow us to change the format of a response from a servlet or JSP page, since clients are increasingly demanding formats other than HTML, such as WML. Filters can perform a variety of functions, including by not limited to:

* Authentication
* Logging and Auditing
* Image Conversion
* Data Compression
* Encryption
* Tokenization
* Triggering Resource