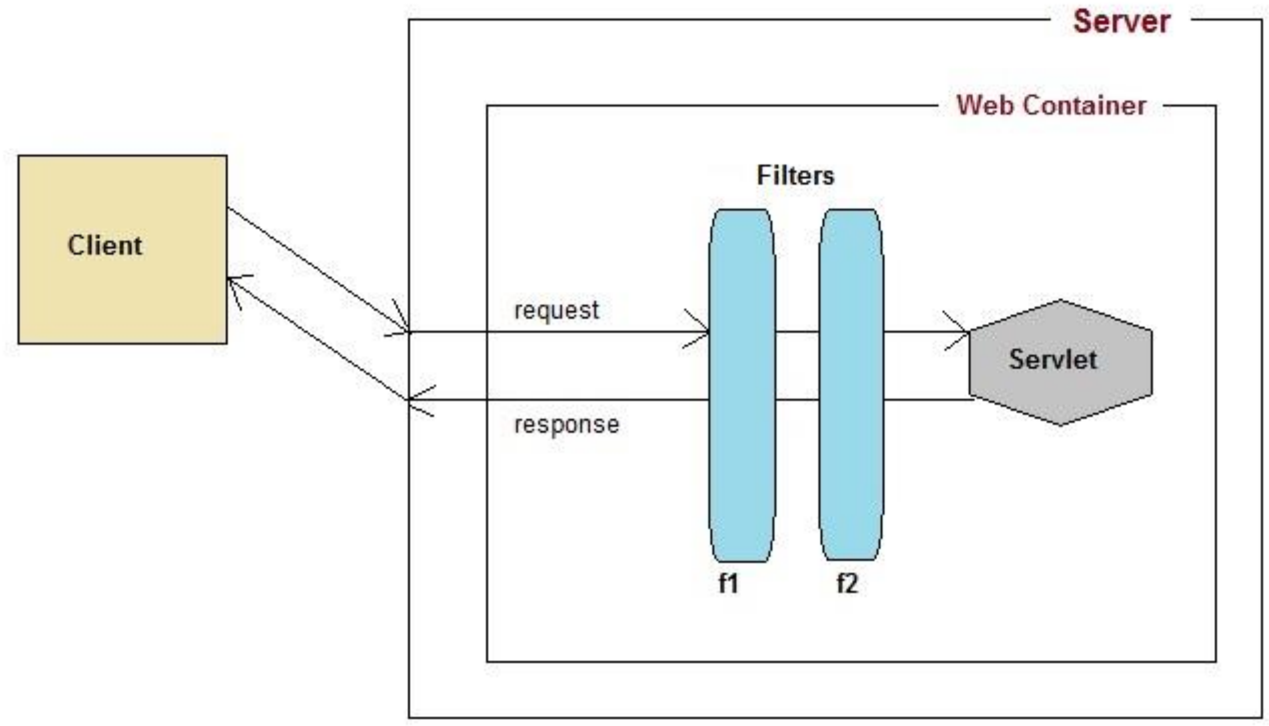
Introduction to Filter API

Filters are compontents that you can use and configure to perform some filtering tasks. Filter is used for pre-processing of requests and post-processing of responses. You can have any number of filters for pre-processing of a request and post-processing of a response. Filters are configured in the deployment descriptor of a web application.



How Filters Works?

- 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.

More about Filter API

Filter API is part of **Servlet API**. Filter interface is found in the **javax.servlet** package.

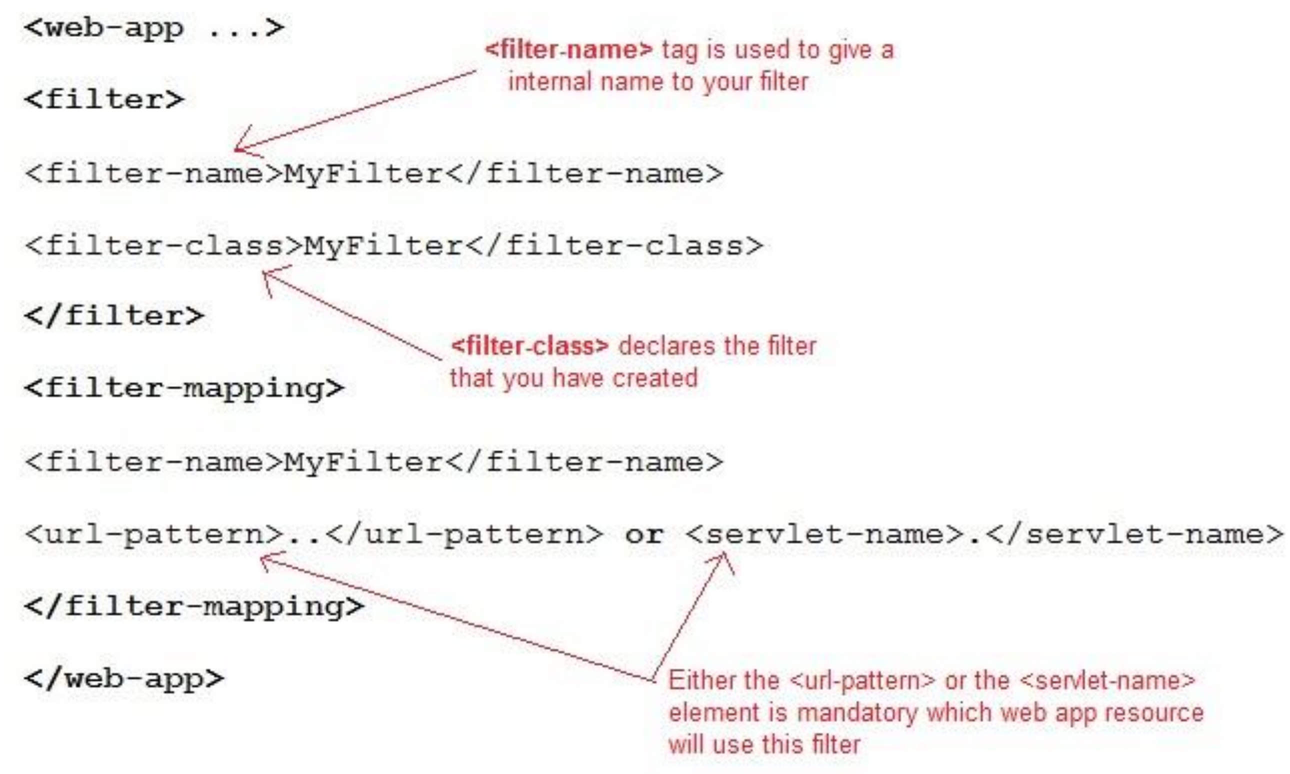
For creating a filter, we must implement Filter interface. Filter interface gives the following life cycle methods for a filter:

1. void `init(FilterConfig filterConfig)`: invoked by the web container to indicate to a filter that it is being placed into service.
2. void `doFilter(ServletRequest request, ServletResponse response, FilterChain chain)`: invoked 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.
3. void `destroy()`: invoked by the web container to indicate to a filter that it is being taken out of service.

What is FilterChain Interface?

FilterChain object is used to invoke the next filter in the chain, or if the calling filter is the last filter in the chain then the rosource at the end of the chain invoked. The resources at the end of Filter chain can either be a target Servlet(in case of request flow) or the Client(in case of response flow) as described in the diagram above.

Declaring a 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>
```

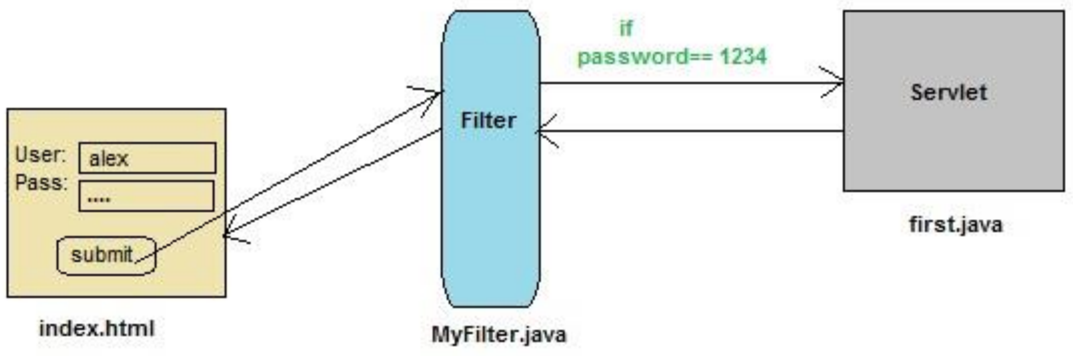
<filter-name> tag is used to give a internal name to your filter
<filter-class> declares the filter that you have created
Either the <url-pattern> or the <servlet-name> element is mandatory which web app resource will use this filter

---

Example demonstrating Filter usage

In this example we are using Filter to authenticate(check correct username and password). Here **index.html**will ask username and password from the user, **MyFilter** will validate the password entered by the user, if the user has entered "1234" as password, then he will be forwarded to **first** servlet else the index.html will be shown again to the user.

This is exactly what we used to do earlier using two servlet classes earlier, one for validation and the other to Welcome the user. Now we will insert a Filter for validating the user.



index.html

```
<form method="post" action="first">

    Name:<input type="text" name="user" /><br/>

    Password:<input type="text" name="pass" /><br/>

    <input type="submit" value="submit" />

</form>
```

web.xml

```
<web-app..>

    <filter>

        <filter-name>MyFilter</filter-name>

        <filter-class>MyFilter</filter-class>

    </filter>

    <filter-mapping>

        <filter-name>MyFilter</filter-name>

        <servlet-name>first</servlet-name>
    </filter-mapping>

    <servlet>

        <servlet-name>first</servlet-name>

        <servlet-class>first</servlet-class>

    </servlet>

    <servlet-mapping>

        <servlet-name>first</servlet-name>

        <url-pattern>/first</url-pattern>

    </servlet-mapping>

    <welcome-file-list>

        <welcome-file>index.html</welcome-file>

    </welcome-file-list>

</web-app>
```

MyFilter.java

```
import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;


public class MyFilter implements Filter {


    public void init(FilterConfig fc) throws ServletException {}


    public void doFilter(ServletRequest request, ServletResponse response,

            FilterChain chain) throws IOException, ServletException {

        PrintWriter out = response.getWriter();

        String pass = request.getParameter("pass");

        if(pass.equals("1234"))

        {

         chain.doFilter(request, response);
        }
        else
        {
            out.println("You have enter a wrong password");
            RequestDispatcher rs = request.getRequestDispatcher("index.html");
            rs.include(request, response);
        }
    }
   public void destroy() { }
}
```

first.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class first extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse respons
e)
            throws ServletException, IOException
     {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        String user = request.getParameter("user");
        out.println("Wellcome "+user);
     }
}
```