5. Intercepting HTTP Request with Filters

Contents

[1. Introduction 1](#_Toc11027)

[2. Servlet Filter Methods 1](#_Toc27217)

[3. Demo: Servlet Filter Methods 1](#_Toc19673)

[4. Filter Mapping in web.xml 1](#_Toc28967)

[5. Summary 2](#_Toc23164)

# 1. Introduction

=>slides: Pg. 1

In this module, we shall understand one of the powerful APIs provided by the Servlets called as filters. We shall understand what filters are, why do we need filters in Servlets, lifecycle of filters, and how to add filters to the Servlet application practically.

=>slides: Pg. 2

Now let us understand what are filters. In simple, non-technical words to understand, we use the term filters regularly in our day-to-day life. The most common usage of the word filters is while filling the water from a tap, if you fill the water directly from the tap, then there will be some additional impurities that will come along with the water. So we use filters to filter the unwanted impurities coming from the tap. And we will take only the purified water which is required.

=>slides: Pg. 3

In simple terms, using Servlets filters, we can stop the request, and we can modify the request from the client before the request accesses the back-end resources. And, also, using filters, we can manipulate the response from the server before it reaches to the client. Now let us understand more in detail.

=>slides: Pg. 4

It is a new feature introduced in Servlets 2. 3 specification. And originally it was a part of the J2EE 2. 3 specification and also available in the current Java EE specification. Application servers have provided this particular feature in the past but not in a standardized way. So the use of Servlet filters allows us to replace something called as Servlet chaining or chaining Servlets. What Servlet chaining means, one Servlet calling the other Servlet where the logic needs to be modified at the normal process of a request response chain.

=>slides: Pg. 5

So what are Servlet filters? Technically speaking, a Servlet filter is a small amount of code that executes before or after serving up a web resource. A Servlet filter does not create its own response, but it can manipulate a Servlet request or a Servlet response. Filters follow the decorator design pattern. It dynamically extends the Servlet behavior without adding the Servlet code or necessarily extending the Servlet source code. We can map one filter to any number of web resources, and also many filters can map to one web resource. Whenever a request is given, then it will be sent to the Servlet filter where the code for preprocessing the request will be written, and then it will be submitted to the Servlet.

=>slides: Pg. 6

For example, let us assume that we have three Servlets--Servlet1, Servlet2, and Servlet3--and also three filters--Filter1, Filter2, and Filter3. Now let us say Servlet1 is mapped with Filter1 and Filter2. And Servlet2 is mapped with Filter1, Filter2, and Filter3. And Servlet3 is mapped with Filter3. So whenever that client sends a request to the Servlet1, then the request will be first passed to the Filter1 and then a request will be given to the Filter2. And once the processing is done, then the request will be submitted to the Servlet for processing. And after the Servlet has processed, then the response will be given to the Filter2, and then it will be passed to Filter1. And, finally, from Filter1, the response will be given to the client. And whenever the client sends a request for the Servlet2, then the request will be passed to the Filter1, Filter2, and then Filter3. After processing the code presenting all the three filters, the request will be given to the Servlet2 for processing, and then the response will also be passed across all the three filters, and the response will be given to the client. And whenever that client sends a request for the Servlet3, then Filter3 alone will be processed, and then the request will be given to the Servlet3. And once the processing is done, then the response will be given to that client. Once we have an idea on what Servlet filters do,

=>slides: Pg. 7

the next question that comes to our mind is, Why do we need a Servlet filter? A filter can perform application task unrelated to the usual operation of the Servlet. Moving a task from a Servlet to a filter gives us the ability to reuse this filter without rebuilding the application. Servlet filters are applied by configuration settings rather than by changing the Java code, that is, the original Java source of the Servlet to which the filter will be applied. By using the filter interface, we can implement a task as a filter allowing to enable or disable the task without rebuilding the entire application. Filters' default nature is they can be chained together. Filters have initialization parameters, so behavior can be parameterized and controlled through the deployment descriptor. Filters have the access to the ServletRequest so they can manipulate the headers, they can add objects to the request attributes, they can add objects to the session. Filters also have the access to the ServletContext and all the resources which are accessed by the Servlet. Once we have an idea on what a Servlet filter is and why we need a Servlet filter,

=>slides: Pg. 8

now let us understand where we can use the Servlet filters. Using the filters, we can record all the incoming requests, log the IP addresses of the computers from which the request originates, conversions, data compressions, encryption and decryptions, input validations, authentication and authorizations, audit access to sensitive resources, email system administrators on every application error. We could compress the response to reduce our bandwidth to make the application perform better. In the next section, we shall understand the Servlet filter's lifecycle and ServletFilter method.

# Servlet Filter Methods

=>slides: Pg. 9

Now let us understand the lifecycle of Servlet filters. The lifecycle of filters is very much similar to that of a Servlet lifecycle. First, in the web container or Servlet container, we load the filter class. Then the web container creates an instance of the filter. Once the instance is created, then the web container will call the init method of the filter where the init method will be executed once. Then the doFilter method will be invoked, which is the main method to perform the actual task. The doFilter method of the filter is called by the container each time a request or response \_\_\_\_\_ is passed through the chain due to client request for a resource. At the end of the chain, the destroy method is called by the web container to indicate a filter is being taken out of service.

=>slides: Pg. 10

Now let us understand the ServletFilter method. ServletFilter is actually declared as an interface in the Java language in the same package as a web Servlet, that is, javax. servlet. The filter interface provides us a couple of methods,

=>slides: Pg. 11

the least of which is its own initialization method, which, of course, is called when the filter is called by the server.

=>slides: Pg. 12

The real need of the work of the logic of our filter can be provided by overriding the doFilter method.

=>slides: Pg. 13

The doFilter method from the filter interface provides us the ServletRequest object,

=>slides: Pg. 14

ServletResponse object,

=>slides: Pg. 15

and something called as a Servlet chain where the FilterChain gives us a handle to the order of filters that are applied to a particular resource without actually knowing what the order is.

=>slides: Pg. 16

The doFilter method and the init method both throw exceptions, so we have to be careful when handling these methods.

=>slides: Pg. 17

The interface also provides us with a destroy method, which will be called by the server when the filter is being taken out of service.

=>slides: Pg. 18

Now let us understand how a filter works. The web deployment descriptor, that is, the web. xml, declares that a filter should be invoked for a particular Servlet. Before the web container invokes a Servlet, it applies all the filters associated with a particular Servlet. For each filter, the web container invokes init method if it is overridden, the doFilter method which gets injected to do a common request or response chain, and the destroy method if we have implemented it if the filter is being taken out of service. Within the filter class, we need to call the doFilter method on the chain object that is passed into the doFilter method to ensure that any remaining filters are invoked without writing any filter code knowing whether or not other filters are applied. The filter decides whether its own processing should come before or after the rest of the filters by controlling the order or the processing order app when the doFilter method is actually called on the chain object. In the next section, we shall understand how to apply filters with the support of ServletFilter method.

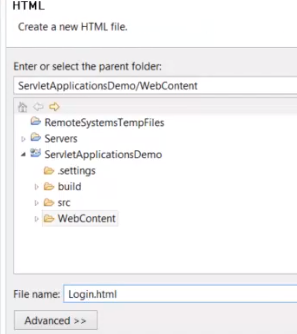
# Demo: Servlet Filter Methods

=>slides: Pg. 19

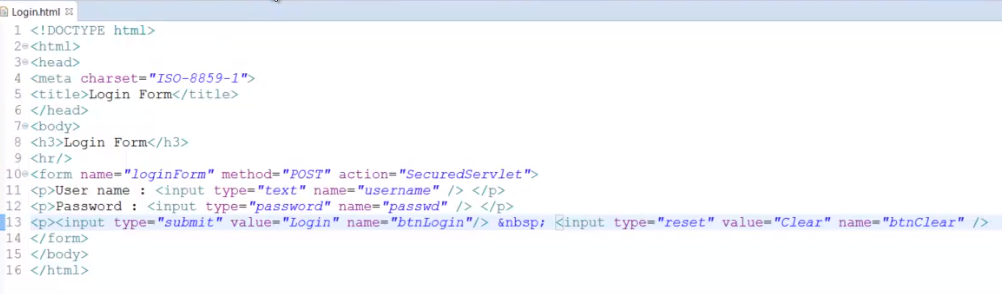
Now let us understand how to apply a single filter to the Servlet. In this demo, we shall create an authentication filter which will validate the user credentials and log the IP address of the client machine along with the login time. Whenever the client sends a request to the secured Servlet,

=>slides: Pg. 20

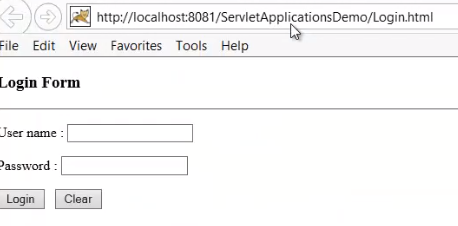
then the request will be first passed to the authentication filter, and then it will pass the request to the secured Servlet. And once the processing is completed, then the response will be passed to the authentication filter. And from there, the final response will be given to the client.



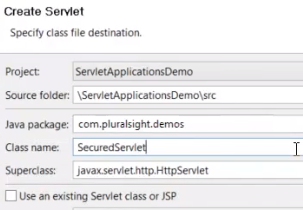
Now let me first create an HTML page. To do, let me right-click on the WebContent folder and click on New, HTML File. And let me provide a meaningful name for the file, for example, Login. html. And let me click on Finish button.



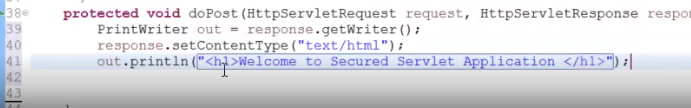
Now let us write the code for a simple login form. First, let me set the title for the page as Login Form. And then within the body tag, let me add a header, h3 Login Form. And then just for beautification, let me add a horizontal rule, hr. As we require form elements, let me add a form tag, form name="loginForm" method="POST" action="SecuredServlet". Let me close the form tag. Now let me add a form element to accept the username. Paragraph Username : input type="text" name="username". And let me close the paragraph. Similarly, let us add another form element to accept the password. Paragraph Password : input type="password" name="passwd". And let me close the paragraph. Now let me add two buttons to submit the data and clear the form data. So let me type in paragraph input type="submit" value="Login" name="btnLogin", and a space, &nbsp;. And then input type="reset" value="Clear" name="btnClear". Let me execute this form.



We can observe a very simple login form. If we need to beautify, we can use CSS, Bootstrap, or Foundation classes. But to make sure we are not deviating from the concept, I created a simple form. But if you are aware of CSS or Bootstrap, feel free to create an --login form. Now let me come back to the Esectionse and add a Servlet page.



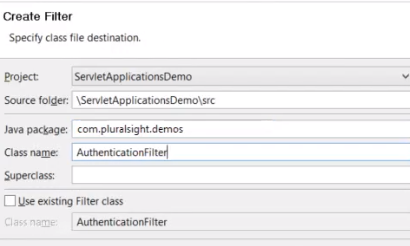
So let me right-click on the application and click on Add, New, Servlet. And then provide a meaningful package name. For example, once again, let me provide com. dxc. demos, and the class name as SecuredServlet, and let us keep the superclass as HttpServlet itself. And let me click on Finish button to create the Servlet page.



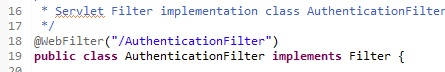
Now within the doPost method, let me write a simple code to display a message. So let me type in PrintWriter out = response. getWriter. Then let me set the content type, response. setContentType("text/html"). And then let me write a simple message, out. println("Welcome to Secured Servlet Application"). Now once again, let me execute the login page, and let me provide the values and click on login button.

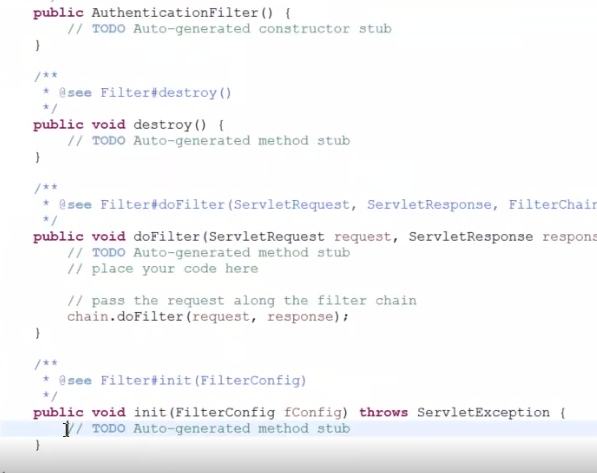


We can observe the browser is displaying the content without any validations or credential checks. Now to add the authentication code, it is not advisable to write the code within this Servlet. Instead, we can take the support of a filter. Always make sure the Servlet has only the exact business logic to be processed. Now let us add a filter. To do, let me right-click on the application once again, and click on New, Filter.



And then let me provide a meaningful package name, com. dxc. demos, and the class name as AuthenticationFilter.

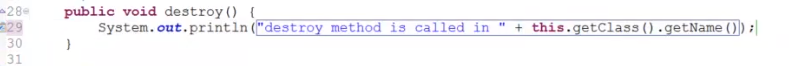




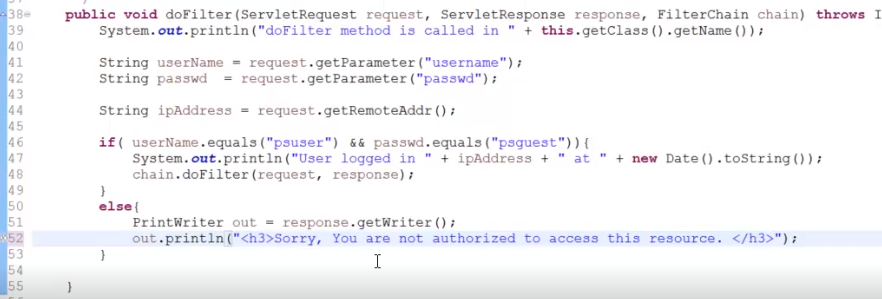
We can observe there are three methods--destroy, doFilter, and the init method. Now let me first add some log messages to be written on the server console.



So within the init method, let me add System. out. println("init method is called in " + this. getClass. getName.



And within the destroy method, let me add System. out. println("destroy method is called in " + this. getClass. getName.



And within doFilter method, let me add System. out. println("doFilter method is called in " + this. getClass. getName. Now let us write the main logic to be processed at the doFilter method. Let me first collect the username and the password. We already have a way that a filter can access all the resources exactly same as that other Servlet. Let me define a variable to hold the value of username. So let me type in String username = request. getParameter("username"). And then to hold the value of password, let me type in String passwd = request. getParameter("passwd"). Since I would like to track the IP address of the user logged into our application, let me define another string variable for holding that IP address of the user. String ipAddress = request. getRemoteAddr. Once we have the values of username and the password, let us do a simple validation. If you have a prior knowledge on database programming using Java, then feel free to use that knowledge while writing this filter. And if you're not aware of database programming using Java, then feel free to watch my course at dxc, Java Platform: Working with Databases, to make yourself comfortable with the database programming using Java. And you can apply that knowledge. For now, let me make it simple with a hard-coded validation--if(username. equals("psuser") && passwd. equals("psguest")), then I consider it as a valued user. I would like to add a log entry. And then I'll load the user for accessing my web resource. So let me type in system. out. println("User logged in " + ipAddress + " at + new Date. toString). Instead of displaying the log information at the server console, we can also add an entry at the database also. And then let me chain for accessing the next web resource. So let me type in chain. doFilter(request, response). And if the user credentials are invalid, then I would like to just display an error message, You're not authorized to access this resource. So let me type in PrintWriter out = response. getWriter, out. println("Sorry, You are not authorized to access this resource. "). We can also send an email to that relevant user or admin whenever an unauthorized action occurs. Now we have created the login page, the secure Servlet, and then we have added the filter. But if we observe, we are not invoking the authentication filter anywhere, and there is no relationship existing between the filter and the Servlet. So how does authentication filter be invoked when the client requests to the secured Servlet page? So let us find the answer in the next section, Filter Mapping on web. xml.

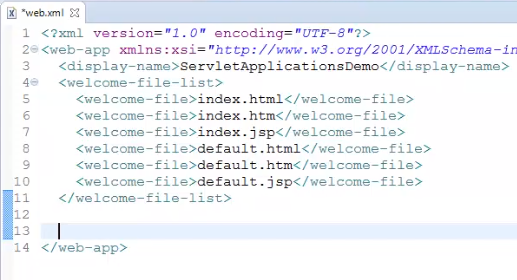
# Filter Mapping in web.xml

=>slides: Pg. 21

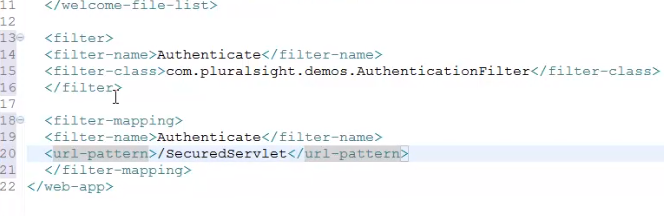
Filters are defined and then mapped to a URL or Servlet in much the same way as a Servlet is defined and then mapped to a URL pattern. In order to map a filter to the Servlet, we need to add an entry for a filter in the deployment descriptor file that does web. xml.

=>slides: Pg. 22

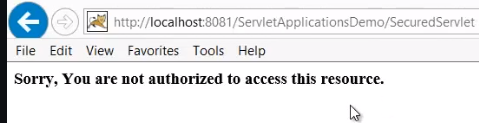
We need to use a tag, filter. Within that, we need to add a filter-name to provide the name for the filter. And then the filter-class where we have to specify the name of the filter class. And if that filter requires any initialization parameters, then we have to set a using init-param tag. To define a parameter, we have to use param-name and then the param-value tags. We can add any number of parameters to the filter. Once the filter is defined within the web. xml, then we need to add a filter mapping to the Servlet. To do, we need to use filter-mapping tag. Then, we have to specify the name of the filter-name and then the url-pattern to be mapped. The above code will help the filter to be invoked whenever the url-pattern matches. In order to use a filter that we apply to all the Servlets, we can use /\*. Or if it has to be invoked for a specific Servlet, we can provide that particular Servlet part. In order to add multiple filters, we need to add different mapping entries within the web. xml file. Now let us understand how to map our authentication filter with the secure Servlet.



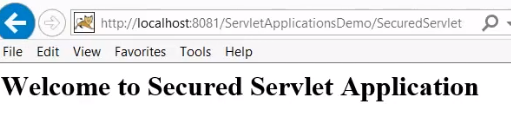
So let me flip to Esectionse, and then let me open the web. xml file present under WEB-INF folder. We can observe some welcome-file entries.



Now let us add our filter--filter, filter-name, Authenticate, filter-class, com. dxc. demos. AuthenticationFilter. Once we have added the entry for the filter, now we need to add the filter mapping. To do, let me type in filter-mapping. Now we need to provide the name of the filter to be invoked. So let me type in filter-name, Authenticate, and then we need to specify for which web resource this filter has to be applied. So let me type in url-pattern/SecuredServlet. Finally, save the web. xml file.



Now, once again, let me execute the login. html file. And let me provide an invalid username and password and click on Login. We can observe an error, invalid username or password. And now let me click on the Back button on the browser and provide a valid username and password.



We can observe, we have been navigated successfully to the Servlet page. Let me flip over to Esectionse. We can observe every time the web resource was requested, we can observe the doFilter method of the filter is also getting executed implicitly.

# Summary

=>slides: Pg. 23

In this module, we have understood the importance of filters, when to use filters, how to create a filter, and map the filter with the Servlets. In the next module, we shall understand how to handle the exceptions in Servlets.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*