9. Providing Servlet Metadata Using Annotations

Contents

[1. Introduction 1](#_Toc30305)

[2. @WebServlet 1](#_Toc844)

[3. @WebInitParam 1](#_Toc12538)

[4. @WebFilter 1](#_Toc22608)

[5. Summary 2](#_Toc2075)

# 1. Introduction

=>slides: Pg. 1

In this module, we shall understand one of the new featured annotations introduced in Servlet 3 for providing the metadata information for the Servlet. We have been using this feature from the very first Servlet page in our course, but we haven't discussed it in detail, and I informed you that I will explain about the annotations in a later module of our course. Now let us understand some key points about providing Servlet metadata using annotations.

=>slides: Pg. 2

Servlet API 3 has introduced annotations. The package javax. servlet. annotation provides the annotation types that can be used for annotating a Servlet class. The annotations can replace equivalent XML configuration present in the web deployment descriptor file, web. xml. Annotations help in developing Servlet applications easily.

=>slides: Pg. 3

Now in this module, we shall understand providing Servlet metadata using WebServlet, WebInitParams, and WebFilter annotations by comparing with the XML configuration required at web deployment descriptor file. Now let us get started.

# @WebServlet

=>slides: Pg. 4

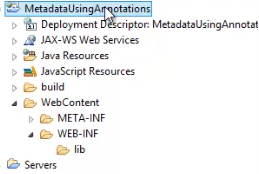
WebServlet annotation is used to declare a Servlet. This annotation is processed by the container at the deployment time, and the corresponding Servlet made available at the specified URL-pattern. First let us understand what we use to perform before annotations are introduced in Servlet, and then let us observe how we can use annotations to provide Servlet metadata.

=>slides: Pg. 5

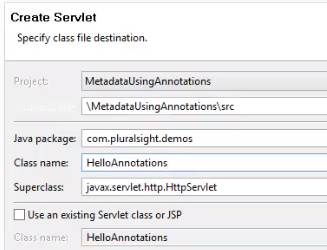
Let us assume we have a class HelloServlet which extends HttpServlet. Now in order to run the Servlet page, we need to configure the Servlet information within the web deployment descriptor file. We have used the Servlet configuration in web. xml in prior modules also. But at that time, our concentration was on the topic that we were discussing. And we haven't given much priority for the configuration settings. Now let us do so in detail. In order to execute the Servlet page, we need to configure the Servlet. To do, we need to define a Servlet. That way, we have to provide a name for the Servlet using servlet-name tag. And then we need to provide the Servlet class definition. For that, we need to use servlet-class tag. Once the Servlet has been defined, then it has to be mapped. To do, we need to use the servlet-mapping tag where we have to provide the name of the Servlet to be mapped using the servlet-name tag. And then we need to specify the URL-pattern to be used for sending the request to the Servlet using url-pattern tag. Once the above configuration details are set within the web. xml file, then we can execute the Servlet page using the URL-pattern configured within the file. I informed you that from Servlet 3 instead of configuring within the web. xml file, we can use annotations. In order to use WebServlet annotations, we still require a class which extends HttpServlet. And we need to add @WebServlet name attribute, which is used to provide the name for the Servlet. And if the name attribute is not specified, then fully qualified name of the class will be used. We need to provide at least one URL-Pattern either in the value attribute or within the urlPattern attribute of the annotation, but not both. The value attribute is recommended for use when the urlPattern is the only attribute being set. Otherwise, the urlPattern attribute should be used. Other than name, value, and urlPattern, WebServlet annotations support other elements which are optional.

=>slides: Pg. 6

Now let us have an idea on the optional elements of WebServlet annotation. Description returns the description of the Servlet if present. displayName returns the display name of the Servlet if present. asyncSupported--it is used to specify if the asynchronous operation is supported by the Servlet or not. We shall understand in detail about asynchronous processing Servlets and how it will increase the scalability of the application in the next module. loadOnStartup--this element tells the Servlet container in what sequence the Servlet should be loaded. The lower numbers are loaded first. If the value is negative or unspecified, the Servlet container can load the Servlet at any time. initParams--it is used to specify the initialization parameters for the Servlet. I will explain about this element in detail in the next clip. smallIcon--it is used to return the small icon of the Servlet if present. largeIcon--it is used to return the large icon of the Servlet if present. Now let us understand how to use WebServlet annotation practically using a simple demo. Now let me open the Eclipse and understand the WebServlet annotation.



I have already created a dynamic website within the Eclipse.



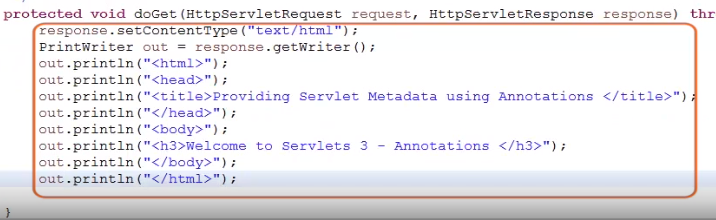
Now let me add a Servlet, so I right-click on the application and click on Add, New, Servlet. And let me provide a meaningful name for the package--com. Pluralsight. demos. And the class name as HelloAnnotations. And let me click on the Finish to create the Servlet. We have observed these steps a couple of times in our course.



Now let us observe the statement above our class, which we never concentrated on. We can observe @WebServlet annotations with the value of URL-patterns.



We can explicitly specify the element value="/HelloAnnotations".



Now let me add some code to set the content type, create an object for the PrintWriter class, and a couple of out statements to generate the response content.



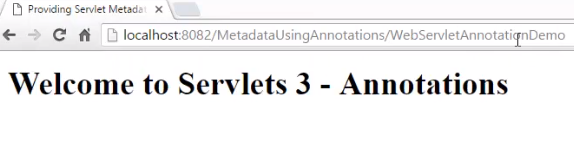
Now let me execute the Servlet page. We can observe the result. Now let me flip to Eclipse again and set additional optional elements values for the WebServlet annotations.



So let me type in name="HelloAnnotations", description="Demo: @WebServlet Annotation". Let me change the value element to urlPatterns, and let me add other urlPatterns to access this Servlet. In order to support multiple parts, we need to use flower brackets, and URL should be separated using comma. So let me type in {"/HelloAnnotations", "/WebServletAnnotationDemo"). Now let us execute the page.



We can observe the URL-pattern HelloAnnotations.



Now let me change the URL-pattern to WebServletAnnotationDemo and press Enter. We can observe the result. We can use any of the URL-patterns to access the Servlet. It is very much easier to register the Servlet using WebServletAnnotation than configuring Servlet details in web deployment descriptor file. In the next clip, we shall understand how to initialize the Servlet with the support of WebInitParams annotation.

# @WebInitParam

=>slides: Pg. 8

WebInitParams annotation is used to declare an initialization parameter on a Servlet or filter within the WebServlet or web filter annotation. We can pass parameters to a Servlet from the web deployment descriptor file also. The init parameters of a Servlet can only be accessed by that Servlet.

=>slides: Pg. 9

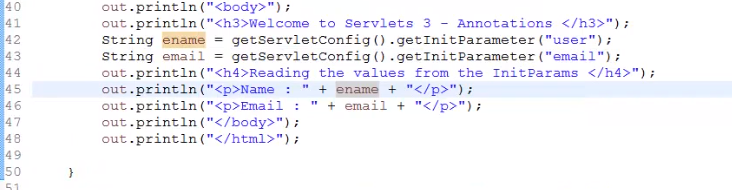
Let us first observe how to configure the init params in the Web. xml file. In order to pass the parameters to the Servlet, we need to use init-param tag. Then to define the parameter, we need to use param-name. And to provide the value, we need to use param-value tag. To read the value of the parameter from the Servlet, we need to use getServletConfig. getInitParameter("name of the parameter"), for example, ("user"). Whatever the configuration we have performed at Web. xml file for the init params, we can achieve the same in a much simpler way using the @WebInitParams annotations. In order to pass the parameters to the Servlet page using the WebInitParams annotation, we need to use the WebServlet annotation. Within that, we have an optional element, init-param, which has to be defined using the @WebInitParams by passing the name and value elements to the WebInitParams annotation.

=>slides: Pg. 10

Now let us understand how to use WebInitParams practically using a simple demo. I have opened the HelloAnnotations Servlet page. Let me add some parameters to be passed for the Servlet page. To do, we need to add the optional element InitParams for the WebServlet annotation.



So within the WebServlet annotation, let me type in initParams=, let me open our flower brackets, and then we need to define the parameters using the WebInitParam annotation. So let me type in @WebInitParam(name, ) let me provide the value as "Sekhar". Optionally, if we want, we can specify the description for this parameter using the description element. We can pass any number of parameters for the Servlet using InitParams. So once again, let me type in @WebInitParam(name="email"). Let me provide the value for the email, for example, sekharonline4u@gmail. com. Once we have defined the parameters, now let us read the values from the parameters passed to the Servlet.



So within the doGet method, let me define string variables to collect the value of name and email. And we know that in order to retrieve the values from the parameters, we can use getInitParameter method of getServletConfig. So let me type in String ename= getServletConfig. getInitParameter("user"), String email = getServletConfig. getInitParameter("email"). Now let me add an out statement to display the value of the user and the email. Let us save the file and execute the Servlet page. We can observe the values passed to the Servlet are collected and displayed at the Servlet page. In the next clip, we shall understand another important annotation to replace the configuration that we perform while working with the Servlet filters, that is, WebFilter annotation.

# @WebFilter

=>slides: Pg. 11

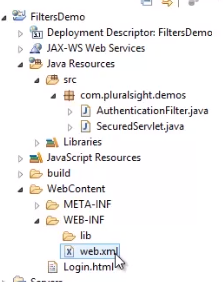
We have already discussed in detail about what are filters, how the filters will work, advantage of filters, and also we have observed the practical implementation of filters by configuring the filters within the web deployment description file. Now let us understand the filter annotation for providing the metadata information for Servlet filters. WebFilter annotation is used to define a filter within the web application. Now let us quickly recap how the filter configuration is performed at the web deployment descriptor file. First, we need to define a class which implements javax. servlet. filter interface. Once the filter task is defined, then we need to configure the filter details within that web. xml file. To do, we use a filter tag in which we provide a name for the filter and the filter class. Once the filter is defined, then we need to map the filter and Servlet. To do, we use a filter-mapping tag in which we need to specify the filter-name and the url-pattern of the Servlet for which the filter should be applied. Now let us understand how to use WebFilter annotation to provide the metadata. The first step we need to perform is add the WebFilter annotation for the class, which implements javax. servlet. filter interface. This annotation contains metadata about the filter being declared. Like WebServlet annotation, WebFilter annotation must specify at least one URL-pattern. This is done by using the urlPattern or value attribute of the annotation. All other attributes are optional with the default settings. The above step will register a filter for the URL-pattern/users. If a filter has to be applied for all the Servlets, then we can use /\*. And if a filter has to be defined for a specific Servlet, then we can use ServletNames="servlet1". And if the filter has to be applied for multiple Servlets, then we can specify ServletNames=, we need to provide a flower bracket and then specify the name of the Servlets separated by a comma.

=>slides: Pg. 13

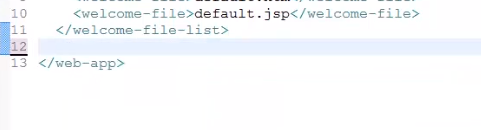
And if you require any initial parameters for the filter, then we can specify the urlPattern for the filter, and then we need to set the optional attribute, initParams, with the WebInitParam annotation. By providing the details for the name and value attributes, other than these attributes, the filter annotation will support the optional attributes such as displayName, filterName, description, asyncSupported, smallIcon, and largeIcon, etc., similar to WebServlet annotation.

=>slides: Pg. 14

Now let us understand practically how to use WebFilter annotation.



Let me open the application that we have developed for the module intercepting HTTP request with filters.



Let me open the web. xml file and remove all the tags present within that file. And then let me open the AuthenticationFilter. And then above the class AuthenticationFilter definition, let me add the WebFilter annotation.



So let me type in @WebFilter(filterName="Authenticate", urlPattern="/SecuredServlet"). Let us save the file and execute the login. html file. And let me first provide an invalid username and password and click on Login button. We can observe an error stating Sorry, You are not authorized to access the page. Now let me click on Back button on browser, and let me provide the valid user credentials and click on Login button. We can observe the user has been navigated successfully to the Servlet page.

# Summary

=>slides: Pg. 15

In this final module, we have understood the importance of annotations in Servlets and how they can be used as a replacement for XML configurations we used to perform \_\_\_\_\_ for providing that metadata details. And also discussed in detail about some of the annotations like WebServlet, WebInitParams, and WebFilter.

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