6. Working with Standard Actions

Contents

[1. Introduction 1](#_Toc13760)

[2. jsp:include 1](#_Toc7537)

[3. jsp:forward 1](#_Toc25071)

[4. jsp:param 2](#_Toc28346)

[5. jsp:useBean 2](#_Toc4795)

[6. jsp:setProperty 2](#_Toc31745)

[7. jsp:getProperty 2](#_Toc25348)

[8. jsp:plugins and jsp:fallback 2](#_Toc7530)

[9. jsp:element, jsp:attribute, and jsp:body 2](#_Toc22553)

[10. Demo: Working with Standard Actions 3](#_Toc478)

[11. Summary 3](#_Toc28870)

# 1. Introduction

=>slides: Pg. 1

 In this last module we have understood how we can handle the exceptions within the JSP page. And in this module we will learn one of the important concepts in JSP working with standard actions.

=>slides: Pg. 2

JSP standard action tags are used to remove or eliminate scriptlet code from our JSP page because scriptlet code are technically not recommended. It is considered to be a bad practice to put Java code directly inside the JSP page.

=>slides: Pg. 3

JSP standard action tags begins with the jsp: prefix

=>slides: Pg. 4

and there are many JSP standard action tags which we are used to perform some specific task. Now in this module we shall understand these action tags in detail

=>slides: Pg. 5

and then later we will learn how to use the JSP standard actions effectively by extending our virtual training company application by providing the search option to search for a course, and if the course is present in the library, we shall display the details of the course. Now let us first understand the JSP action tags.

# jsp:include

=>slides: Pg. 7

Include action allows us to include either a static or dynamic resource in a JSP file. When the source JSP page receives a request the results of including static and dynamic resources are quite different.

=>slides: Pg. 8

If the resource is static, then its content is inserted into the calling JSP file. And if the resource is dynamic, then the request is sent to the included resource, the included page is executed and then the result is included in the response from the calling JSP page.

=>slides: Pg. 9

The syntax for the JSP include action is jsp:include page= the relative URL of the page to be included flush=true or false. The Boolean attribute value determines whether the included resource has its buffer flushed before it is included.

=>slides: Pg. 10

For example, let us assume that we have two files, date. jsp and the other one is main. jsp. Within data. jsp file let us say I wanted to display the current date, so to do we can just type in paragraph Date: Java expression of new java. util. Date. toLocaleString, and within the main. jsp file let me add some code to create a simple JSP page. Now let us say that we wanted to include the date. jsp file within the main. jsp file. So we need to type in jsp:include page=date. jsp flush=true.

=>slides: Pg. 11

Now if we execute the main. jsp file, we can observe that the date present within the data. jsp file has been included along with the output of main. jsp file. We already observed the include directive.

=>slides: Pg. 12

So what is the difference between the include action and the include directive? Let us understand now. Include directive will copy the code of one page into another page, whereas the include action will copy the result of a page into another page.

=>slides: Pg. 13

The include directive will copy the code during the translation time, whereas the include action will copy the output during the runtime. Include directive will generate only one equivalent servlet of the source and the include action will generate two equivalent servlets for both the source and the included page. Include directive will perform static binding and the include action performs dynamic binding. In the next section we shall understand forward action.

# jsp:forward

=>slides: Pg. 14

Forward action tag

=>slides: Pg. 15

is used to effectively terminate the execution of the current page and discards the output and dispatches a request to a new page. The requested page can be either an HTML, JSP or a servlet. The forward action is equivalent to getRequestDispatcher. foward concept of servlets. When we use the forward action the control will be completely transferred from the source to destination and it is the responsibility of the destination page to provide the output to the client. The output of the source page will be lost or discarded. One important point we need to remember is that the JSP page must be buffered to use a JSP forward tag. We cannot set buffer equal to none in a page directive.

=>slides: Pg. 16

In order to use the forward action, we need to type in jsp:forward page=URL of the page to be forwarded.

=>slides: Pg. 17

Now let us understand the forward action practically. Let me use the same JSP pages used for the include action, that is main. jsp and the date. jsp. Now instead of include action, let me update the code to use the forward action

=>slides: Pg. 18

and once the code is saved and executed, then we can observe the output from the date. jsp page.

=>slides: Pg. 19

The content what is displayed earlier from main. jsp page in included example is not displayed for the forward action sample. In order to redirect the user from one page to the other, we can use the sendRedirect method,

=>slides: Pg. 20

so now let us understand the difference between forward action and the sendRedirect method. In forward action, the source and the destination will have direct communication, whereas in sendRedirect the source and the destination page will communicate indirectly by performing an extra round trip to the client. In forward action, the source and the destination will use the same request and response object, whereas in sendRedirect the source and the destination will use separate request and response objects. In forward action, both source and the destination must belong to the same application, whereas in sendRedirect the source and the destination can be in the same application or within a different application or within the same server or different servers. In web application development, if you transfer the control from one JSP to another JSP multiple times in a sequence by using include or forward actions, then it is said to be JSP chaining and if there is a requirement such that we need to pass some data from one page to the other page, then we need to use param action, which we will observe in the next section.

# jsp:param

=>slides: Pg. 21

In the web application development, when we are transferring the control from one JSP page to the other JSP page using either include or forward action, along with the control, the request object will also be transferred from the source to the destination page. When the request object is transferred from the source to the destination, if we have a requirement to modify the value of any parameter available in the request object or if we want to add additional parameters into the request object, then we can use param action.

=>slides: Pg. 22

In order to define the param action, jsp:param name= name of the parameter and the value= value to be passed to the other page. Jsp:param action has to be used as a subtag for include forward or plugin actions and I will explain about plugin action in the later part of the course.

=>slides: Pg. 23

Now let us understand how do we use this param action. Let us assume that we have two JSP pages, one is main. jsp and the other one is course. jsp. Now within the main. jsp file, let me add the code for the forward action, jsp:forward page=course. jsp. And now let me define some parameters that I wanted to pass from the main. jsp page to course. jsp page. To do that we can type in jsp:param name=courseName, value=JAVA, jsp:param name=trainer, value=Sekhar Srinivasan. Once we have added the code to the main. jsp file, now let us update the course. jsp file. I just wanted to read the values from the parameters passed from the main. jsp and wanted to display the details, so let me provide a header, Course Details, and then a paragraph Course Name expression request. getParameter of courseName, paragraph Trainer expression request. getParameter of trainer.

=>slides: Pg. 24

If we execute the main. jsp page, we can observe the result with the course name as JAVA and the trainer is Steve Samuels. In the next section we shall understand one of the most important features for efficient JSP application development, how to use JavaBeans in JSP with the support of useBean action.

# jsp:useBean

=>slides: Pg. 25

UseBean action is a way for declaring and initializing the actual JavaBean component object. If bean object of the bean class is already created, then it does not create the bean, depending on the scope, but if object of bean is not then it will instantiate the bean. Once the bean is declared, then the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP.

=>slides: Pg. 26

Any Java class can be considered as a JavaBean if that class follows the following specification. There is no special keyword to define the bean explicitly. The specifications to be followed are the class must be declared as public, class must have 0 argument constructor, the variables declared inside the bean are called as bean properties, and all the bean properties should be declared private. For every bean property we must specify a setter and a getter methods, and these methods should be declared as public. The class must implement serializable interface.

=>slides: Pg. 27

Now let us understand how to use the useBean action.

=>slides: Pg. 28

The syntax will be jsp:useBean followed by the attributes.

=>slides: Pg. 29

The attributes of the useBean action are id. This attribute will specify the name of the reference which will refer to the object. Class, this attribute will specify the name of the class who's object has to be instantiated, and the name of the class should be specified as a fully qualified name. Scope, this attribute will specify the location where the object can be accessed. UseBean supports various scopes, page scope is used to specify that the object can be accessed only within the current page. Request scope is used to specify that the object can be accessed in all those JSPs where the same request is used. Session scope is used to specify that the object can be accessed from all the JSP pages until the session is not closed. Application scope is used to specify that the object can be accessed from all the JSP pages in all the request in all the sessions and everywhere. Note, the default value of the scope attribute is page. The next attribute of useBean we need to remember is type. This attribute is used to specify the name of the class who's reference has to be declared. The class name should be specified as fully qualified name. BeanName, this attribute is used to specify a logical name for the bean.

=>slides: Pg. 30

Now let us understand how to use the useBean action. Let us assume that we have a class name of bean, which implements serializable interface and defines a bean property firstName and lastName. In order to access the firstName we need to define a method to set the value and a method to retrieve the value from the firstName. Similarly we require methods to set the lastName and also to retrieve the value from the lastName, and as per the JavaBean specifications, we require a 0 arg constructor and once we have created the JavaBean class,

=>slides: Pg. 31

in order to use the JavaBean within the JSP application, we need to use jsp:useBean, id=name, class=com. psdemos. model. Name, scope=request. In the next section, we shall understand how to set the values for the JavaBean object using the setProperty action.

# jsp:setProperty

=>slides: Pg. 32

Once the JavaBean object has been created using the useBean action, in order to set the values for the JavaBean property, we can use setProperty action. We can use the setProperty action in two ways, setProperty can be used within the body of the useBean action or it can be used outside of the useBean action.

=>slides: Pg. 33

Whenever the setProperty action is defined within the body of useBean action, then the property setting is conditional, that is, in simple terms the property values will be set only if the new bean is created. If an existing bean with that scope and id are found, the body of the tag will never execute so that the property won't be rest from the JSP code.

=>slides: Pg. 34

And if the setProperty action is defined outside of the useBean property, then the value will be set irrespective whether a new bean was instantiated or an existing bean was found.

=>slides: Pg. 35

In order to set the bean property using the setProperty action, we need to use jsp:setProperty name=id of the JavaBean instance, property=propertyName or \*, value= value for the property. And if the value for the property is going to be assigned from a control or from the request object parameters, we need to use param= request object parameter name to be used for providing the value. One important point we need to remember while setting the bean property using the setProperty action, that is if the property attribute is \*, then the list of the beans settable properties is compared with the list of parameters in the current request. Whenever a match occurs, the set method is invoked with the corresponding request parameter.

=>slides: Pg. 36

Now let us understand practically how to set the values for the JavaBean, which we have created in the previous section. Let us say we have a JSP page for accepting the values for firstName and lastName and when the user clicks on the Submit button, then the data should be submitted to setName. jsp where I would like to set the values of the JavaBean using setProperty action. Now within the setName. jsp page, let us first instantiate the instance for the JavaBean using the useBean action. So let us type in useBean action id=nameBean, class=com. psdemos. model. Name, scope=request and within the useBean action body, let us set the JavaBean property. To do, we can type in setProperty action, name=nameBean, property=firstName. Since the value for the property will be assigned from the request object parameter, we need to set param=firstName and similarly we set the lastName property also. SetProperty action name=nameBean, property=lastName, param=lastName.

=>slides: Pg. 37

Whenever we know that the request object param name matches with the JavaBean property value, then instead of setting the individual property, we can also set using setProperty action name=nameBean, property= \*. As we have got a fair idea on how to set the value for the JavaBean property using the setProperty action, in the next section we shall understand how to read the value from the JavaBean property using the getProperty action.

# jsp:getProperty

=>slides: Pg. 38

Once the bean has been declared with the useBean action, if we wanted to access the exposed properties of the bean, we need to use getProperty action tag where getProperty tag inserts the string value from the primitive type or object into the output string. For primitive types, the value will be converted automatically to a string value and for the object types, the toString method of the object will be invoked.

=>slides: Pg. 39

In order to use the getProperty action tag, we need to use jsp:getProperty name= the id of the bean instance specified in the useBean action, property= the name of the property to retrieve the value. One important point we need to remember while using the getProperty action tag is that it allows us to access only the properties of the bean attribute, it doesn't have a capability to access the nested properties, that is property of a property attribute.

=>slides: Pg. 40

For example, assume that we have a class Person, which has a string personName property and also has a property Car of type Car class where the Car class has a String carName property. Now if we wanted to retrieve the value of the person name, we can access using paragraph Person name: getProperty action, name=person, property=personName, and in order to access the carName, if we use paragraph Car name, getProperty action, name=person, Property=car,

=>slides: Pg. 41

then all we get will be the result of the cars toString method because car is a property of the attribute, but we cannot navigate to the carName property of the car class. Now let us understand practically how to use getProperty action tag for reading the values set using the previous section.

=>slides: Pg. 42

Within the setName. jsp, let us first read the values of the JavaBean. To do, let me type in paragraph First Name getProperty action name=nameBean, id=firstName, and similarly to retrieve the values from the last name, paragraph Last Name, getProperty action, name=nameBean, id=lastName, and now let me add an anchor tag to navigate for getName. jsp page. So we need to type in anchor href =getName. jsp Read Values and let me close the anchor tag.

=>slides: Pg. 43

Now within the getName. jsp page, in order to read the values from the JavaBean, first we need to instantiate the JavaBean class. So to do, we need to type in useBean action id=nameBean, class=com. psdemos. model. Name. Now within the useBean property, let me add the code to read the values of the First Name and Last Name, and now let me add the code to read the values of the First Name and Last Name outside of the useBean body tag.

=>slides: Pg. 44

Now let us observe what will happen if we execute the application. Let me provide the username as Sekhar and the last name as Srinivasan.

=>slides: Pg. 45

Let me click on Submit button. We can observe the values set for the JavaBean.

=>slides: Pg. 46

Now let me click on the read values link. We can observe null values at both places.

=>slides: Pg. 47

Now let me change the scope of the useBean to session in both setName. jsp

=>slides: Pg. 48

and getName. jsp. Now let me execute the application

=>slides: Pg. 49

and once again let me type in the user name as Sekhar and the last name as Srinivasan.

=>slides: Pg. 50

We can observe the values at the setName. jsp. And now let me click on the read values link. We can observe that the code present within the useBean body is not executed again

=>slides: Pg. 51

and we can observe the values of First Name and Last Name displayed from the code present outside of useBean body. Hope you got a fair idea on how to work with useBean, setProperty and getProperty action tags. In the next section we shall understand JSP plugins action tag.

# jsp:plugins and jsp:fallback

=>slides: Pg. 52

JSP plugin action is used to insert Java components into our JSP page. It is used to identify the browser type and inserts the object or embed tag accordingly. If the required plugin is not present, then the JSP plugin action tag downloads the plugin and then executes the Java component where the Java component can be either an applet or a JavaBean.

=>slides: Pg. 53

In order to use the JSP plugin action, the syntax will be jsp:plugin type=bean or applet, code= object code, codebase=object codebase where the code attribute is used to specify the class name of the applet or JavaBean in the form of packageName. className. And the codebase is used to provide the base URL that consists the class files of the component. Plugin action tag additionally supports the following optional attributes type, align, height, width, name, title, et cetera. If the plugin requires any value, then that value can be passed using the params action tag. This action isn't optional and the params action is a direct child of the plugin action. If you define a params action, then it is mandatory to have one or more param actions to provide the additional parameters required for the component. In order to use the params action within the plugins action, we need to use params. Within the params body to define the parameter, we can use jsp:param name= parameter name and value= value for the parameter. We can define any number of parameters as per the requirement. In order to use the plugin action tag efficiently, we need to understand the usage of another action tag called as fallback action. =>slides: Pg. 54-55

Fallback action is a direct child of the plugin action and this action tag is optional, which is used to provide some meaningful information to the client's browser if the requested plugin cannot be started. The most common usage of this action tag is to show some text that indicates there is a problem while loading the component.

=>slides: Pg. 56

In order to use the fallback action within the plugin action body, we need to define fallback. Within the fallback action body we can write any valid content to be displayed for the user. For example, paragraph Error while loading the applet. Now let us understand how to use the plugin and the fallback action tag practically.

=>slides: Pg. 57

Say that we have a simple applet code where we have defined a label and we have two methods, one is init and the other one is start. Within the init method we have set the properties of the label and within the start method we are accepting the firstName and lastName values, and that value we have set as a text for the label. Now my requirement is to use this applet within the JSP page.

=>slides: Pg. 58

So within the JSP page, in order to use this applet, we need to type in jsp:plugin type=applet, code=com. psdemos. applet. MyApplet. class, codebase=appletCode. Since this applet is accessing the parameters to set the values for the applet parameters, we need to use params, so let me type in jsp:params. We need to provide the value for firstName, so let me type in jsp:param name=firstName, value=Sekhar. And similarly I need to provide the value for the lastName, so I need to type in jsp:param name=lastName and value=Srinivasan. Once we have defined the parameters in case if the applet is not loaded properly, then the user should be informed about that. To do, we can use the fallback. So let me type in jsp:fallback. Within the fallback body, paragraph Error while loading applet.

=>slides: Pg. 59

Once we save the JSP file and execute the page we can observe an applet is displayed with the name Sekhar Srinivasan. In the next section we shall understand some of the JSP actions which helps in generating the XML documents.

# jsp:element, jsp:attribute, and jsp:body

=>slides: Pg. 60

Element action is used to dynamically create an XML element and adds it to the response. This element is useful primarily in JSP pages, which are created using the XML syntax where other approaches cannot be used because of the \_\_\_\_\_ requirement.

=>slides: Pg. 61

JSP element action can be defined in two ways, without a body, with a body.

=>slides: Pg. 62

In order to define a JSP element without a body, syntax will be jsp:element name=elementName, and in order to define a JSP element with a body we can type in jsp:element name= elementName.

=>slides: Pg. 63

Within the JSP element we can use jsp:attribute action to set the attribute for the element and also we can use jsp:body action in order to provide the body for the element. Now let us understand how to use the element attribute and the body actions to generate an anchor tag.

=>slides: Pg. 64

Within the body tag I wanted to provide a heading, so let me type in jsp:element name=h1 and in order to provide the content for the h1 we need to use the body action. So let me type in jsp:body Demo on Element, Attribute, and Body actions, and then I want it to have a horizontal row. To provide let me type in jsp:element name=hr. Since hr tag doesn't require any body or attribute, we can simply close the element by typing /> and then I wanted to provide an anchor tag to allow the user to navigate to some page. To do, we can type in jsp:element name=a. As we know that anchor tag requires a hyper reference, href attribute, in order to add the attribute for the element we need to use the jsp:attribute action. So let me type in jsp:attribute name=href somepage. jsp, and in order to provide the hypertext where the user clicks on it, we need to use the JSP body action. So to provide the hypertext we can type in jsp:body Click here.

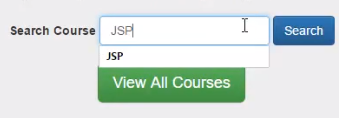
=>slides: Pg. 65

Now once we execute the above page we can observe a header, then a horizontal row, and a hyperlink with the text Click here, and when the user clicks on that link, the user will navigate to some page. Hope you got a fair idea on how to use element attribute and the body actions. In the next section we shall understand how to use the JSP actions in our virtual training web application and such for a course and display the course details if the course is present in the library.

# Demo: Working with Standard Actions

=>slides: Pg. 66

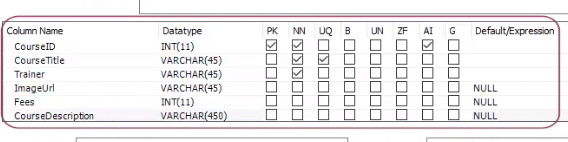
Before we start developing the application, let us first observe what we have completed until the previous module, and what we need to develop from the finished application. And then we shall continue with our recording. We have completed the home page for our application and also we have completed the registration and the login form, and in the last module we have extended our application to support custom error pages. Now in this section we shall use JSP standard actions and extend our virtual training web application to add the search feature,

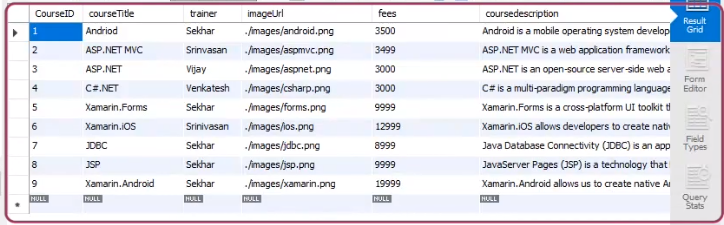


that is one the user types in a course name and clicks on the search,

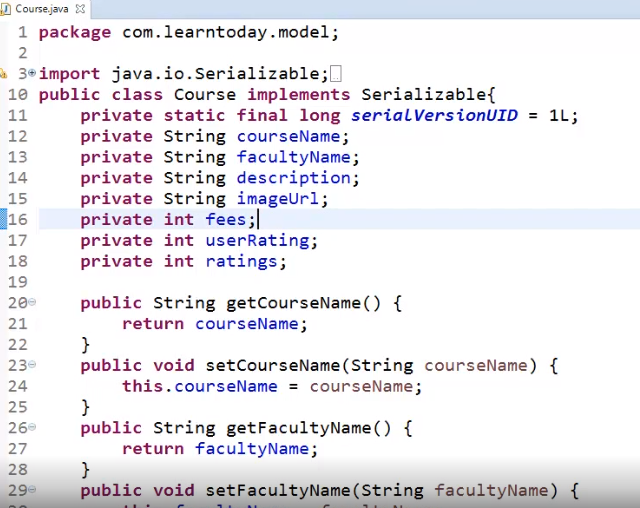
=>slides: Pg. 67

then if the course exists then I wanted to display the details of that course and if the course is not present, show a valid error message. Now let us get started.

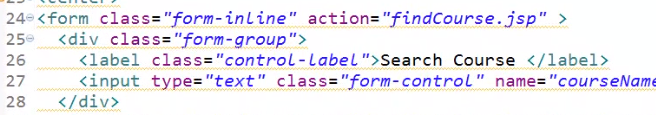


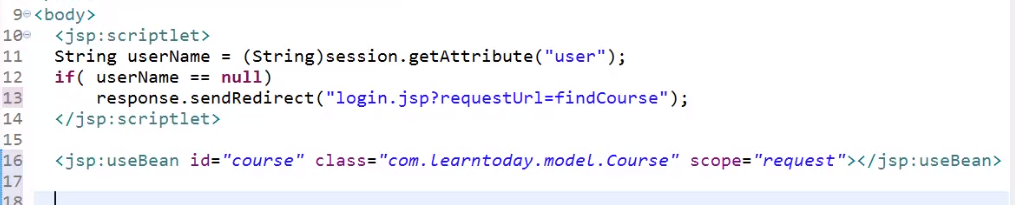


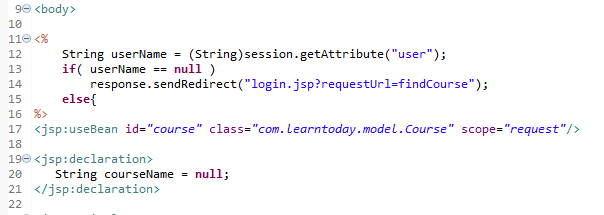
So, create a table with the name Course with the following columns and add some records as the courses.

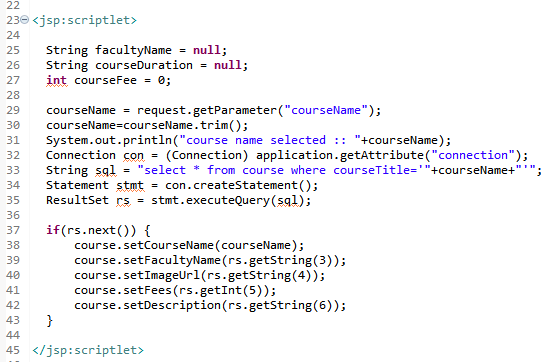


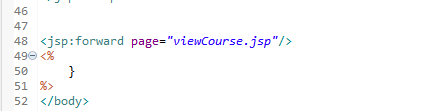
Create the course JavaBean. I have defined courseName, facultyName, description, imageUrl, fees, and ratings bean properties, and also have defined the setter and getter methods to set and get the values from the above bean properties. Once the course bean is created, now let me first open the index page.

We can observe when the user clicks on button search we are submitting the data to the find course JSP page. So let me add the find course JSP page.

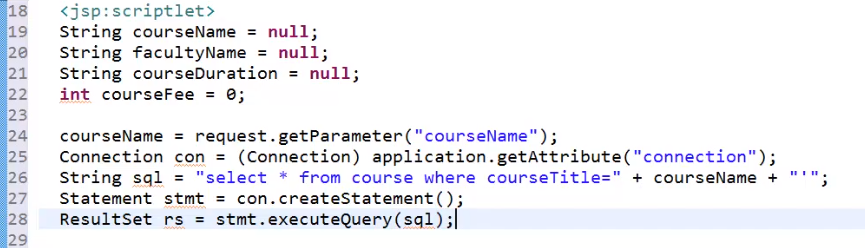








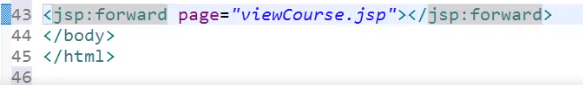
Once the page is added, now let me add a scriptlet and my requirement is in order to search a course the user should log in to our application. So let me type in String userName = session. getAttribute of user. And let me typecast the value to a String. If the session is not set, the value for the username will be null. So I need to redirect the user to login page. So let me type in if of userName = null, response. sendRedirect of login. jsp? requestUrl=findCourse. If the user is already logged in, then we need to search the course and display the course details with the support of course JavaBean. So let me type in else useBean action id=course class=com. learntoday. model. Course and the scope I would like to provide as request, so let me type in scope=request.



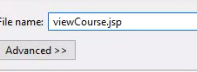
Now let me add a scriptlet and write the code to read the course details and set to the JavaBean. So let me type in scriptlet and let me define a variable to hold the course name. So let me type in string courseName = request. getParameter of courseName. Now I need a connection to interact with the database. So let me type in Connection con = Connection application. getAttribute of connection. And now I require a select statement to retrieve the course details based on the course name. So let me type in String sql = select \* from course where courseTitle= courseName. Once the query is prepared, let me define an object for the statement. So let me type in Statement stmt = con. createStatement, and we need to collect the result. So let me type in ResultSet rs = stmt. executeQuery of sql. If there is some data, then I would like to set the details to our course bean.



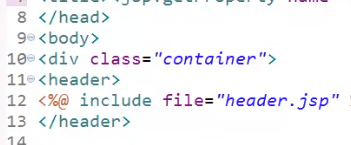
So let me add the code to verify the condition and set the values for the JavaBean. Once the bean property has been set I would like to redirect the user to another view for viewing the course details.



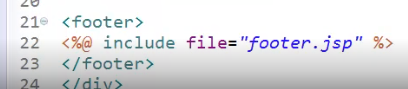
So let me type in jsp:forward page=viewCourse. jsp.



Now let me add viewCourse. jsp page and also to update the page with the



header and



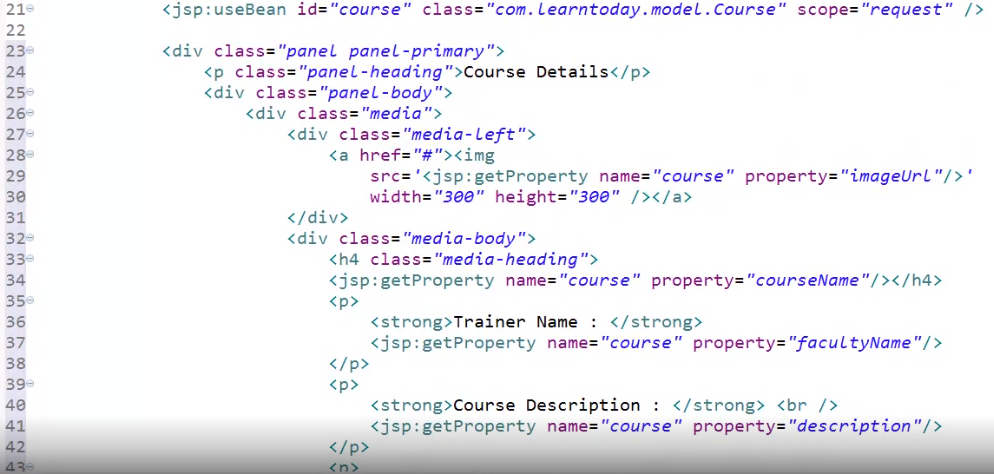
footer sections. Now let me update the title for the page.

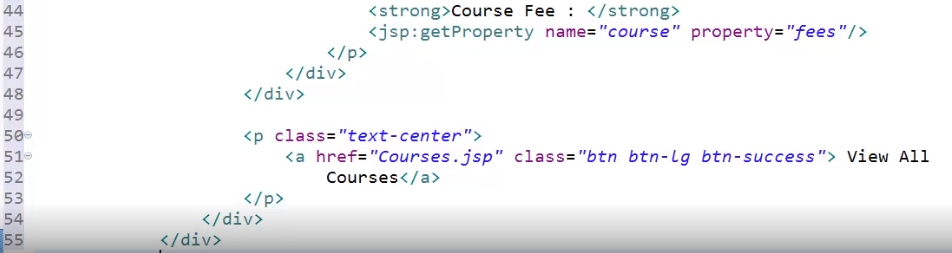


So let me type in jsp:getProperty name=course property=courseName details.



Within the article section let me type in the scriptlet and let me define the variable to hold the username login. So let me type in String user = String session. getAttribute of user. And I would like to have a welcome message, so let me type in String message = Welcome + user. Now let me display the welcome message. So let me type in bold jsp:expression message. I would like to use the JavaBean object to display the course detail, so we need the useBean action, so let me type in useBean id=course class=com. Learntoday. model. Course scope=request.



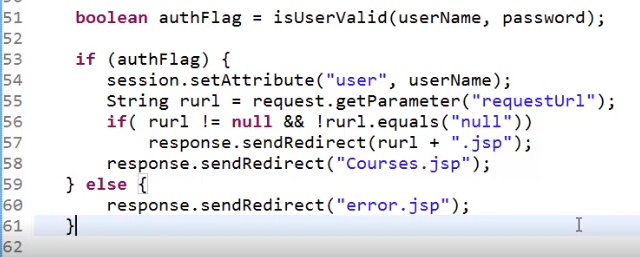




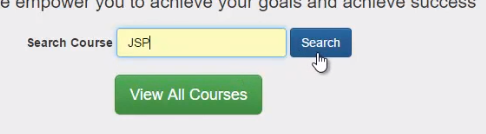
Now let us add the code to display the course details. I have added a row where we have displayed the course image and then I have used bootstrap panels to display the course details and within the panel body I have returned the code to get the value of the course name, faculty name, description, fees, and also I have provided a hyperlink to view all the courses. In the findCourse. jsp page we have verified if the user hasn't logged in, then we are redirecting the user to the login page. Once the user has logged in, I want the user to navigate back to the requested page.



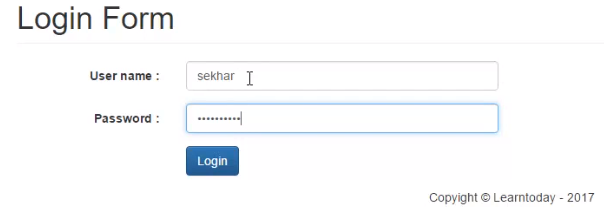
So let me open the authenticate. jsp page to update the code accordingly. Let me scroll down to the authenticate flag condition and let me add the following code,



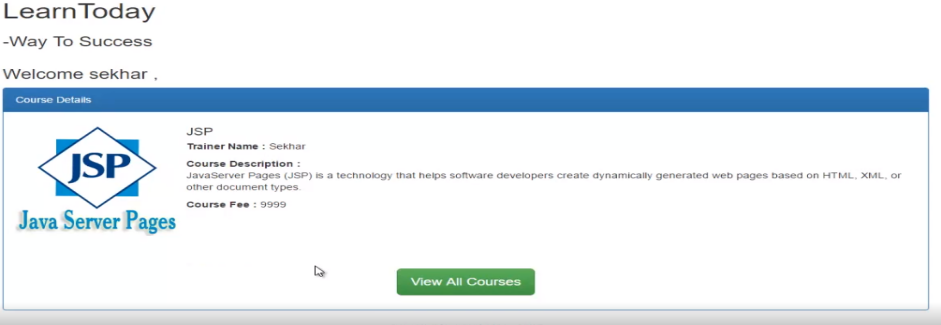
I have collected URL value to a string variable and verifying a condition if the written URL value is not null, then I wanted to redirect the user to the requested page instead of the defaultCourses. jsp page. Now let the save the file and execute the application.



Let me type in the course JSP within the search text and let me click on the Search button.



We can observe the login page. So let me provide my credentials, the username as Sekhar, and let me provide the password and click on Login.



We can observe the course details of the JSP page. In this section I have used the code to interact with the database within the JSP page directly. But remember it is advisable not to write the code directly within the JSP page. In the next modules I will explain how to write the code efficiently.

# Summary

=>slides: Pg. 68

In this module we have understood various standard actions provided by the JSP and also understood how to use them effectively. In the next module we shall understand one of the powerful concepts, expression language for JSP development, and how it will help in reducing the code and write the code effectively by updating our virtual training application.

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