Developing Applications Using Java Web Frameworks

Session - 3

Struts 2 - Interceptors and Tags

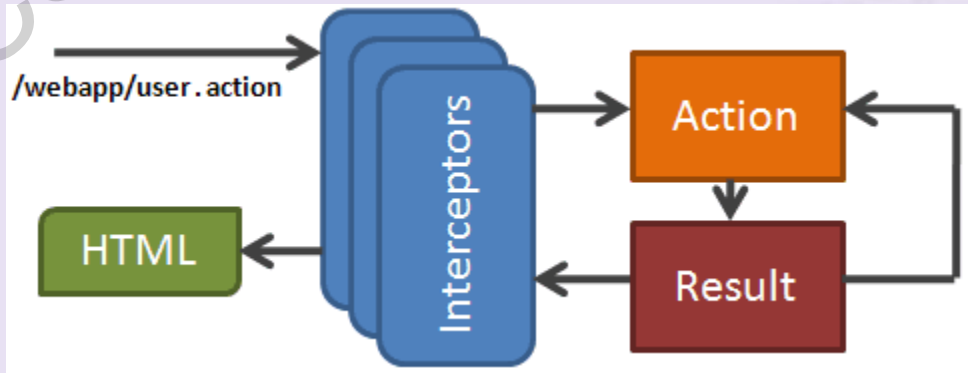
Objectives

- Describe the purpose of Interceptors in Struts 2 framework
- Explain different types of built-in Interceptors
- Explain how to configure Interceptors in Struts 2 application
- Describe Interceptors Stacks
- Explain the process of creating custom Interceptors in Struts 2 framework
- List the different types of tags used in Struts 2 View page
- Explain various Data tags
- Explain various Control tags
- Explain various User Interface (UI) tags

Introduction

- Interceptor is an important feature introduced in the Struts 2 framework.
- Interceptors sit between controller and action.
- Interceptors intercepts the request and response, processes the request before and after invoking action.

The main aim of having Interceptors is to separate the core functionality that may be applicable to multiple actions.



Interceptors 1-2

- An Interceptor is an object which intercepts an action dynamically.
- A stack of interceptors:
  - Can be configured for an action.
  - Are executed before the execution of the mapped action, to provide all the pre-processing functionalities to the request.
  - Are a set of built-in Interceptors provided by the Struts 2 framework, which can be used to provide the required functionalities to action.

Interceptors 2-2

The Struts 2 framework also allows the developers to develop custom Interceptors for defining the specific functionalities for the Web application.

- A custom Interceptor can be created:
  - By implementing the `com.opensymphony.xwork2.interceptor.Interceptor` interface.
  - By extending the `com.opensymphony.xwork2.interceptor.AbstractInterceptor` class.

Request and Interceptors

- Every request that is received by the Struts 2 framework passes through each Interceptor.
- Once the request is received, the Interceptors can:
  - Ignore the request
  - Act on the request data
  - Short-circuit the request and prevent the Action class method to be executed

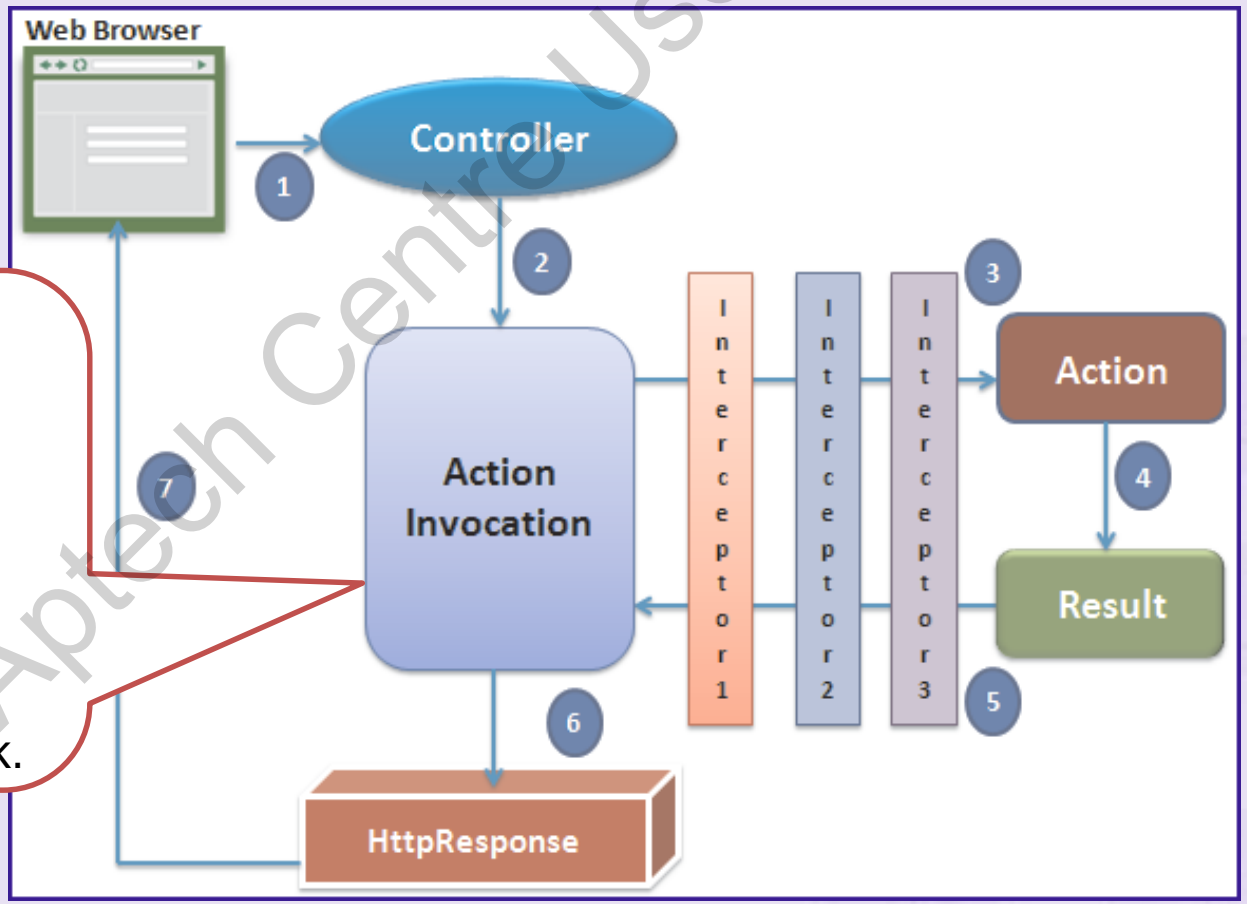
In Struts 2, the default Interceptors are configured in the `struts-default.xml` file.

Working of Interceptor 1-2

- The `ActionInvocation` class is responsible for the execution of the action including the sequential invocation of the Interceptor stack.
- `ActionInvocation` class:
  - Encapsulates the processing details associated with the execution of a particular action.
  - On receiving a request, the framework decides to which action the URL maps.
  - Stores the references of the configured Interceptors.

Working of Interceptor 2-2

- Following figure shows the invocation of interceptors by the `ActionInvocation` class in the Struts 2 framework.



The `ActionInvocation` class starts the invocation process by executing the first Interceptor in the stack.

Configuring Interceptors 1-2

- The Interceptors can be configured using any one of the following approaches:
  - By declaring all the Interceptors in the `struts.xml` configuration file. The Interceptor classes are defined using a name-class pair in the configuration file.
  - By declaring all the Interceptors that are used for the Web application in an XML file and including that XML file in `struts.xml` configuration file.
- All the Interceptors that are required to perform the pre-processing functionalities of the request for a given action should be defined in the action mapping for that specific action.

Configuring Interceptors 2-2

- Following code snippet shows the declaration of the Interceptors:

```
...
<include file="struts-default.xml">
<package name="default" extends="struts-default">
<interceptors>
   <interceptor name="timer1" class="timer1_class">
   <interceptor name="timer2" class="timer2_class">
</interceptors>
...
<action name="Login" class="example.Login">
   <interceptor-ref name="timer1" />
   <interceptor-ref name="timer2" />
    <result name="success">/example/DisplayDetails.jsp</result>
    <result name="error">/example/error.jsp</result>
</action>
</package>
...
```

- The `<interceptor-ref name="...">` element is used to declare the list of Interceptors that will intercept the action request.

Default Interceptor Configuration 1-2

The `struts-default.xml` file is the base configuration file with default settings of the components in the Struts 2 framework.

- `struts-default.xml` file:
  - Is automatically included into `struts.xml` file to provide default configuration settings to the Web applications.
  - Contains the definitions of all Interceptors and Interceptor stacks.
  - Can be included in the `struts-default.xml` in the `struts.xml` configuration file by extending the package from the `struts-default` package.

Default Interceptor Configuration 2-2

- Following figure shows the list of interceptors defined in the `struts-default.xml` file:

```
<interceptors>
    <interceptor name="alias" class="com.opensymphony.xwork2.interceptor.AliasInterceptor"/>
    <interceptor name="autowiring" class="com.opensymphony.xwork2.spring.interceptor.ActionAutowiringInterceptor"/>
    <interceptor name="chain" class="com.opensymphony.xwork2.interceptor.ChainingInterceptor"/>
    <interceptor name="conversionError" class="org.apache.struts2.interceptor.StrutsConversionErrorInterceptor"/>
    <interceptor name="cookie" class="org.apache.struts2.interceptor.CookieInterceptor"/>
    <interceptor name="cookieProvider" class="org.apache.struts2.interceptor.CookieProviderInterceptor"/>
    <interceptor name="clearSession" class="org.apache.struts2.interceptor.ClearSessionInterceptor" />
    <interceptor name="createSession" class="org.apache.struts2.interceptor.CreateSessionInterceptor" />
    <interceptor name="debugging" class="org.apache.struts2.interceptor.debugging.DebuggingInterceptor" />
    <interceptor name="execAndWait" class="org.apache.struts2.interceptor.ExecuteAndWaitInterceptor"/>
    <interceptor name="exception" class="com.opensymphony.xwork2.interceptor.ExceptionMappingInterceptor"/>
    <interceptor name="fileUpload" class="org.apache.struts2.interceptor.FileUploadInterceptor"/>
    <interceptor name="i18n" class="com.opensymphony.xwork2.interceptor.I18nInterceptor"/>
    <interceptor name="logger" class="com.opensymphony.xwork2.interceptor.LoggingInterceptor"/>
    <interceptor name="modelDriven" class="com.opensymphony.xwork2.interceptor.ModelDrivenInterceptor"/>
    <interceptor name="scopedModelDriven" class="com.opensymphony.xwork2.interceptor.ScopedModelDrivenInterceptor"/>
    <interceptor name="params" class="com.opensymphony.xwork2.interceptor.ParametersInterceptor"/>
    <interceptor name="actionMappingParams" class="org.apache.struts2.interceptor.ActionMappingParametersInteceptor"/>
    <interceptor name="prepare" class="com.opensymphony.xwork2.interceptor.PrepareInterceptor"/>
    <interceptor name="staticParams" class="com.opensymphony.xwork2.interceptor.StaticParametersInterceptor"/>
    <interceptor name="scope" class="org.apache.struts2.interceptor.ScopeInterceptor"/>
    <interceptor name="servletConfig" class="org.apache.struts2.interceptor.ServletConfigInterceptor"/>
    <interceptor name="timer" class="com.opensymphony.xwork2.interceptor.TimerInterceptor"/>
    <interceptor name="token" class="org.apache.struts2.interceptor.TokenInterceptor"/>
    <interceptor name="tokenSession" class="org.apache.struts2.interceptor.TokenSessionStoreInterceptor"/>
    <interceptor name="validation" class="org.apache.struts2.interceptor.validation.AnnotationValidationInterceptor"/>
    <interceptor name="workflow" class="com.opensymphony.xwork2.interceptor.DefaultWorkflowInterceptor"/>
    <interceptor name="store" class="org.apache.struts2.interceptor.MessageStoreInterceptor" />
    <interceptor name="checkbox" class="org.apache.struts2.interceptor.CheckboxInterceptor" />
    <interceptor name="datetime" class="org.apache.struts2.interceptor.DateTextFieldInterceptor" />
    <interceptor name="profiling" class="org.apache.struts2.interceptor.ProfilingActivationInterceptor" />
    <interceptor name="roles" class="org.apache.struts2.interceptor.RolesInterceptor" />
    <interceptor name="annotationWorkflow" class="com.opensymphony.xwork2.interceptor.annotations.AnnotationWorkflowInterceptor" />
    <interceptor name="multiselect" class="org.apache.struts2.interceptor.MultiselectInterceptor" />
    <interceptor name="deprecation" class="org.apache.struts2.interceptor.DeprecationInterceptor" />
```

Struts 2 Framework Interceptors

- Each Interceptor implements logic for a specific function which is common to all Web applications.
- Some of the common Interceptors are as follows:
  - Chaining Interceptors
  - Checkbox Interceptors
  - Conversion Interceptors
  - Create Session Interceptors
  - Debugging Interceptors
  - Servlet-config Interceptors
  - Exception Interceptors
  - Parameters Interceptor
  - Other Interceptor

Chaining Interceptors 1-2

- Extends `AbstractInterceptor` class.
- Main requirement in action chaining is copying the parameters from ValueStack of one action to the ValueStack of the next action class.
- User can define a collection of include and exclude elements to control which parameter is to be copied and how it is to be copied.
- Following table lists the methods available in Chaining Interceptor class:

Method	Description
`Collection getExcludes()`	Returns a collection of excluded parameters.
`Collection getIncludes()`	Returns a collection of included parameters.
`String intercept (ActionInvocation action)`	Implements the code to handle interception.
`Avoid setExcludes (Collection excludes)`	Sets a collection of excluded parameters.
`void setExcludes (Collection excludes)`	Sets a collection of included parameters.

Chaining Interceptors 2-2

- Following code snippet shows the implementation of the Chaining Interceptors:

```
...
<package>
<action name="Login" class="example.Login">
 <interceptor-ref name="custom_stack" />
   <result name="success" type="chain">action1</result>
   <result name="error">/example/error.jsp</result>
</action>

<action name="action1" class="example.action1">
 <interceptor-ref name="custom_stack" />
 <result name="success" >/example/DisplayDetails.jsp</result>
</action>
</package>
...
```

Checkbox Interceptors

- Defined in the `CheckboxInterceptor` class.
- Implements the `Interceptor` interface.
- Has a field named uncheckedValue of String data type and a method `void setUncheckedValue(String unchecked)` to set value to this field.
- Is included in the default Interceptor stack such as `basicStack`, `defaultStack`, and `paramsPrepareParamsStack` which is defined in the `struts-default.xml` file.

Conversion Error Interceptors 1-2

- Defined in the `ConversionErrorInterceptor` class.
- Is the subclass of the `XWork 2` conversion error interceptor.
- Is used when the action implements the `ValidationAware` interface or the `ActionSupport` class.
- The value of all the fields is stored so that during subsequent requests, the values will be available for display.
- Maps errors as a field error.
- Contains the method, `ActionContext.getConversionErrors()` which will return a map containing all the conversion errors.

Conversion Error Interceptors 2-2

- Is included in the default Interceptor stack such as `basicStack` and `defaultStack` which is defined in the `struts-default.xml` file.
- Following code snippet demonstrates the implementation of the `ConversionError` Interceptor:

```
...
<action name="Login" class="example.Login">
   <interceptor-ref name="params" />
   <interceptor-ref name="conversionError" />
<result name="success">/example/DisplayDetails.jsp </result>
</action>
...
```

Create Session Interceptors

- Defined in the `CreateSession` class.
- Extends the `AbstractInterceptor` class.
- Is used for creating an HttpSession.
- Defines the logic to be executed when the Interceptor invokes the `intercept()` method.
- Following code snippet shows the implementation of the `CreateSessionInterceptor` Interceptor:

```
. . .
<action name="Login" class="example.Login">
<interceptor-ref name="create-session" />
<interceptor-ref name="defaultStack" />
<result name="success">/example/DisplayDetails.jsp </result>
</action>
...
```

Debugging Interceptors

- Defined in the `DebuggingInterceptor` class.
- Implements the `Interceptor` interface.
- Provides developer with different debugging screens.
- Methods present in this Interceptor class are as follows:
  - `String getParameter(String key)`
  - `String intercept(ActionInvocation action)`
  - `void printContext()`
  - `void setDevMode(String mode)`

- The **devMode** should be enabled in the `struts.properties` file for the Interceptor to intercept the request.

Servlet-config Interceptors

- Defined in the `ServletConfig` class.
- Allows the developer to inject various objects from the Servlet environment in the `Action` class.
- Found in the `org.apache.struts2.Interceptor` package.
- Supported interfaces are as follows:
  - `ServletContextAware` sets the `ServletContext`.
  - `ServletRequestAware` sets the `HttpServletRequest`.
  - `ServletResponseAware` sets the `HttpServletResponse`.
  - `ParameterAware` sets a map of the request parameters.
  - `RequestAware` sets a map of the request attributes.
  - `Session Aware` sets a map of session attributes.
  - `ApplicationAware` sets a map of application scope properties.
  - `PrincipalAware` sets the `Principal` object used for applying security.

Exception Interceptors 1-2

- Defined in the `ExceptionMapping` class.
- Extends the `AbstractInterceptor` class.
- Provides the functionality of exception handling by displaying a page describing the real problem.
- Enables the mapping of the exception to a result code and does not throw an exception.
- Wrapped exception within an `ExceptionHandler` and pushed on the stack.
- Contains `Intercept()` method and getter/setter methods for its fields such as `logCategory`, `logEnabled`, and `logLevel`.

Exception Interceptors 2-2

- Following code snippet shows the implementation of the Exception Interceptor in the `struts.xml` file:

```
...
<global-exception-mappings>
    <exception-mapping exception="java.lang.Exception"
result="exception" />
</global-exception-mappings>
<action name="Login" class="example.Login ">
  <interceptor-ref name="exception" />
  <interceptor-ref name="prepare" />
  <interceptor-ref name="debugging" />
  <interceptor-ref name="params" />
  <interceptor-ref name="defaultStack" />
   <result name="success">/example/DisplayDetails.jsp </result>
   <result name="exception">/example/exception.jsp </result>
</action>
...
```

Parameters Interceptor

- Defined in the `ParametersInterceptor` class.
- Sets the values of all the parameters on the ValueStack.
- Provides `ActionContext.getParameters()` method that obtains all the parameters from the ValueStack.
- On invocation of the interceptor, the value in three flags are set: Those are as follows:
  - `XWorkMethodAccessor.DENY_METHOD_EXECUTION` restricts the invocation of methods when it is set ON.
  - `InstantiatingNullHandler.CREATE_NULL_OBJECTS` automatically creates null reference when it is set ON.
  - `XWorkConverter.REPORT_CONVERSION_ERRORS` reports errors when conversion of data type takes place provided it is set ON.

Other Interceptors 1-2

- Following table lists some of the other Interceptors provided by Struts 2 framework:

Interceptor	Description
Alias Interceptor	Aliases a named parameter to a different parameter name.
Execute and Wait Interceptor	Displays an intermediary waiting page to the user while the action is executed in the background.
Message Store Interceptor	Stores and retrieves error messages, field errors, and action errors in the session for actions. These are used for actions implementing ValidationAware interface.
Roles Interceptor	Allows the execution of action provided the user belongs to one of the configured roles.

Other Interceptors 2-2

Interceptor	Description
Validation Interceptor	Provides validation support for actions by checking the action against all the validation rules declared in the Validation framework configuration files such as Action-validation.xml.
Scope Interceptor	Looks for the specified parameters and pulls these parameters from the given scope.
Timer Interceptor	Logs the amount of time elapsed between the execution of action and the execution time of the Interceptors in the Interceptor stack of the action.
Model Driven Interceptor	This interceptor pushes the Model result on the ValueStack. However, to do so, the Action class must implement ModelDriven interface.

Interceptor Stacks 1-3

- Set of Interceptors that can be grouped of Interceptors that are commonly used and required.
- The <interceptor-stack> element is used to define the stack of Interceptors.
- Following table lists some of the Interceptor stacks:

Interceptor Stack	Description
validationWorkflowStack	Adds to the basic stack feature the validation and workflow features.
fileUploadStack	Adds to the basic stack feature the file uploading feature.
chainStack	Adds to the basic stack feature the chaining feature.

Interceptor Stacks 2-3

Interceptor Stack	Description
`defaultStack`	Provides a complete stack, including debugging and profiling.
`executeAndWaitStack`	Provides an execute and wait stack by displaying a waiting page to the user. This is useful when file is being uploaded.
`i18nStack`	Handles the settings for the specified locale for the current action request..
`jsonValidationWorkflowStack`	Serializes validation and action errors into JSON.

Interceptor Stacks 3-3

- Following code snippet shows the declaration of Interceptor stack:

```
...
<include file="struts-default.xml">
<package name="default" extends="struts-default"> <interceptors>
   <interceptor name="timer1" class="timer1_class">
   <interceptor name="timer2" class="timer2_class">
   <interceptor-stack name="custom_stack">
   <interceptor-ref name="timer1" />
   <interceptor-ref name="timer2" />
</interceptor-stack> </interceptors>
...
   <action name="Login" class="example.Login ">
   <interceptor-ref name="custom_stack" />
   <result name="success">/example/DisplayDetails.jsp</result>
   <result name="error">/example/error.jsp</result> </action>
</package>
...
```

- The `defaultStack` is defined in the `struts-default.xml` using `<default-interceptor-ref name="defaultStack" />` element.

Custom Interceptor 1-5

- The developer can create a custom Interceptor class by extending the class from the Interceptor class.
- The class should define `init()`, `destroy()`, and `intercept()` methods.
- The description of these methods are as follows:
  - `void destroy()` method is used to clean up resources allocated by the interceptor.
  - `void init()` method is called at the time of intercept creation, but before the request processed using intercept, and to initialize any resource needed by the Interceptor.
  - `String intercept(ActionInvocation invocation)` method allows the Interceptor to intercept processing and to do some processing before and/or after the processing `ActionInvocation`.

Custom Interceptor 2-5

- Following code snippet shows a simple example of creating an interceptor for the Struts 2 Web application:

```
public class MyFirstInterceptor implements Interceptor{
    @Override
    public void destroy() {   }
    @Override
    public void init() {          }
    @Override
    public String intercept(ActionInvocation actionInvocation)
throws Exception {
        String startInterceptor="   Start Interceptor 1";
        System.out.println(startInterceptor);
        String result=actionInvocation.invoke();
        String endInterceptor="   End Interceptor 1";
        System.out.println(endInterceptor);
        return result;       }
}
```

Custom Interceptor 3-5

- Following code snippet demonstrates creation of the `Action` class:

```
public class MyAction extends
ActionSupport{
public String execute()
 {
 System.out.println("In Action");
  return SUCCESS;
}
}
```

- Following code snippet shows the configuration of custom interceptor in `struts.xml` file:

```
. . .
<package name="default" extends="struts-
default" namespace="/">
    <interceptors>   <interceptor
name="myfirstInterceptor"
class="com.example.MyFirstInterceptor" />
<interceptor name="secondInterceptor"
class="com.example.MySecondInterceptor" />
</interceptors>
<action name="Action1"
class="com.example.MyAction">
  <interceptor-ref
name="myfirstInterceptor"/>
  <interceptor-ref
name="mysecondInterceptor"/>
  <result
name="success">Welcome.jsp</result>
  <result name="input">login.jsp</result>
    </action>  </package> </struts>
```

Custom Interceptor 4-5

- Following code snippet shows the `index.jsp` page to call the appropriate action:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Performed Action</title>
</head>
<body bgColor="lightBlue">
  <s:form action="Action1">
    <s:submit value="For calling Interceptor" align="center"/>
  </s:form>
</body>
</html>
```

Custom Interceptor 5-5

- Following code snippet demonstrates the remaining part of `struts.xml` file:

```
. . .
<welcome-file-list>
 <welcome-file><b>F</b>index.jsp</welcome-file>
 </welcome-file-list>
. . .
```

- The **output** of the custom interceptor application is as follows:

```
Start Interceptor 1
Start Interceptor 2
In Action
End Interceptor 2
End Interceptor 1
```

Struts 2 Tag Library

- Struts 2 framework uses a set of tags for data reference.
- Before using the Struts 2 tags, the developer must have the statement, `<%@ taglib prefix="s" uri="/struts-tags" %>` that assigns the 's' prefix by which the Struts 2 tags will be identified.
- Struts 2 tags is divided into two types:
  - Struts Generic Tags
  - UI Tags

Control Tags 1-4

- The control tags are used to control the flow of page execution.
- The most commonly used control tags are namely, `if`, `elseIf`, `else`, `append`, `merge`, and `iterator`.
- **Iterator Tag**
  - Is used to loop over collection of objects.
  - Can iterate over any Collection object, Map, Enumeration or Iterator, and Array object.
  - Attributes supported are as follows:
    - Value
    - Status

Control Tags 2-4

- **If and else Tag**
  - Is similar to `if-else` control structure provided in languages.
  - Attribute supported is, `Test` - contains the Boolean expression which is evaluated and tested. It returns either true or false.
- Following code snippet shows the `Action` class for accepting country:

```
public class CountryAction extends ActionSupport {
private String country;
  public String getCountry() {
   return country;
  }
  public void setCountry(String country) {
   this.country = country;
  }
```

Control Tags 3-4

```
  @Override
  public String execute() throws Exception {
   return SUCCESS;
  }
}
```

- Following code snippet shows the `index.jsp` page which displays message to the user based on the condition:

```
<form action="countryAction" method="post">
  Select Country : <select name="country">
    <option value="France">France</option>
    <option value="Nepal">Nepal</option>
    <option value="Russia">Russia</option>
    <option value="China">China</option>
    <option value="USA">USA</option>
   </select><br> <input type="submit">
</form>
```

Control Tags 4-4

```
<hr>
<s:if test="country!=null">
  <s:if test="country=='Russia'">
  <s:property value="country" /> is the selected country.
</s:if>
<s:elseif test="country=='USA'">
 <s:property value="country" /> is the selected country.
</s:elseif>
<s:elseif test="country=='China' or country=='Nepal'">
  <s:property value="country" /> is the selected country.
</s:elseif>
<s:else> <s:property value="country" /> is not the selected country.
</s:else>
</s:if>
<hr>
<a href="index.jsp">Select Country Again</a>
</body>
</html>
```

- Code displays the form with the selection list. Then, depending on the selected option, a message is displayed to the user, based on the evaluation of the condition.

Data Tags 1-11

- Data manipulation or creation is done with the help of Data Tags.
- The most commonly used data tags are action tag, include tag, bean tag, date tag, param tag, property tag, push tag, set tag, text tag, and url tag.
- **Action Tag**
  - Helps the users to call actions directly from a JSP page by specifying the action name and an optional namespace.
  - The `executeResult` parameter must be specified in a program, otherwise any result processor defined for this action in `struts.xml` will be ignored.

Data Tags 2-11

- Following code snippet demonstrates the `Action` tag:

```
public class ActionTagAction extends ActionSupport {
 public String execute() throws Exception {
     return "done";
 }

 public String doDefault() throws Exception {
  ServletActionContext.getRequest().setAttribute("stringByAction",
"This is a String put in by the action's doDefault()");
     return "done";
 }
}
```

Data Tags 3-11

- Following code snippet shows the `struts.xml` file:

```
<!—struts.xml -->
    ....
  <action name="actionTagAction1"
class="example.testing.ActionTagAction">
      <result name="done">success.jsp</result>
  </action>
   <action name="actionTagAction2"
class="example.testing.ActionTagAction" method="default">
      <result name="done">success.jsp</result>
  </action>
    ....
```

Data Tags 4-11

- Following code snippet demonstrates the `actiontag.jsp`:

```
. . .
<s:action name="actionTagAction" executeResult="true" />
<div> … </div>
. . .
<s:action name="actionTagAction!specialMethod" executeResult="true">
<div> … </div>
. . .
<div> … </div>
<s:action name="actionTagAction!default" executeResult="false" />
<s:property value="#attr.stringByAction" />
```

- The jsp page invokes two methods namely, `execute()` and `doDefault()`.

Data Tags 5-11

Include Tag

- Is used to include a JSP file in another JSP page.

Following code snippet demonstrates the use of `Include` tag:

```
 ...
<-- First Syntax -->
<include value="AptechJsp.jsp">
<-- Second Syntax -->
<include value="AptechJsp.jsp">
<param name="param1" value="value2">
<param name="param2" value="value2">
     </include>
<-- Third Syntax -->
<include value="AptechJsp.jsp">
<param name="param1">value1</param>
<param name="param2">value2</param>
</include>
...
```

Data Tags 6-11

- **Bean Tag**
  - Represents a class that applies to JavaBeans specification.
- Following code snippet demonstrates an example for `bean` tag:

```
...
<bean name="org.apache.struts2.util.number" var="number">
<param name="first" value="100">
<param name="last" value="125">
</bean>
...
```

Data Tags 7-11

- **Date Tag**
  - Helps to modify a date. User can use a custom format or can use the predefined format in the properties file.
- Following code snippet demonstrates an example for `date` tag:

```
<s:date name="person.birthday" format="dd/MM/yyyy" />
<s:date name="person.birthday" format="%{getText('some.i18n.key')}"
/>
<s:date name="person.birthday" nice="true" />
<s:date name="person.birthday" />
```

Data Tags 8-11

- **Param Tag**
  - Is used to parameterize other tags.
  - Has two parameters - `name` (String) shows the name of the parameter and `value` (Object) shows the value of the parameter.
- Following code snippet demonstrates an example for `param` tag:

```
<pre>
<ui:component>
 <ui:param name="key"     value="[0]"/>
 <ui:param name="value"   value="[1]"/>
 <ui:param name="context" value="[2]"/>
</ui:component>
</pre>
```

Data Tags 9-11

- **Property Tag**
  - Gets the property of a value, which will default to the top of the stack if none is specified
- Following code snippet demonstrates an example for `property` tag:

```
<s:push value="myBean">
    <!-- Example 1: -->
    <s:property value="myBeanProperty" />

    <!-- Example 2: -->TextUtils
    <s:property value="myBeanProperty" default="a default value"
/>
</s:push>
```

Data Tags 10-11

- **Push Tag -** Pushes value on stack for easier usage.
- Following code snippet demonstrates an example for `push` tag:

```
...
<push value="employee">
<property value="userName1">
<property value="userName2">
</push>
...
```

- **Set Tag** - Allocates a value to a variable in a more definite scope.
- Following code snippet demonstrates an example for `set` tag:

```
...
<set name="Aptech" value="environment.name">
<property value="Aptech">
...
```

Data Tags 11-11

- **url Tag** - Is used to create a URL.
- Following code snippet demonstrates an example for `url` tag:

```
<-- Example 1 -->
<s:url value="example.action">
    <s:param name="id" value="%{selected}" />
</s:url>
<-- Example 2 -->
<s:url action="myexample">
    <s:param name="id" value="%{selected}" />
</s:url>

<-- Example 3-->
<s:url includeParams="get">
    <s:param name="id" value="%{'22'}" />
</s:url>
```

Form UI Tags 1-2

- Struts UI tags display the data on the HTML page and use the data from value stack or from Data tags.
- Struts UI tags are divided into three types Form Tags, Non-Form tags, and Ajax tags.
- **Form Tags:**
  - The most commonly used Form tags are checkbox, checkboxlist, combobox, doubleselect, head, file, form, hidden, label, and password.
  - These tags provide user interface for the Struts Web applications.
  - Form tags are categorised into three types namely, Simple UI tags, Group UI tags, and Select UI tags.

Form UI Tags 2-2

- Following code snippet demonstrates the use of `form` tags:

```
<%@ taglib prefix="s" uri="/struts-tags"%>
. . .
<body>   <s:div>Email Form</s:div>
   <s:text name="Please fill in the form :" />
   <s:form action="hello" method="post" enctype="multipart/form-
data">
   <s:hidden name="secret" value="secretvalue"/>
   <s:textfield key="email.from" name="from" />
   <s:password key="email.password" name="password" />
   <s:textfield key="email.to" name="to" />
   <s:textfield key="email.subject" name="subject" />
   <s:textarea key="email.body" name="email.body" />
   <s:label for="attachment" value="Attachment"/>
   <s:file name="attachment" accept="text/html,text/plain" />
   <s:token />    <s:submit key="submit" />    </s:form>
</body> </html>
```

Non-Form UI Tags 1-4

- The most commonly used Non-Form tags are as follows:
  - **Actionerror**
    - It is tag is used to send the error feedback message to user.
  - **Actionmessage**
    - It is tag is used to send information feedback message to user.

Non-Form UI Tags 2-4

- **Component** tag renders custom UI widget using the specified templates.
- Following code snippet shows an example for `component` tag:

```
<!-- JSP -->
   <s:component template="/my/custom/component.vm"/>
     or
   <s:component template="/my/custom/component.vm">
     <s:param name="key1" value="value1"/>
     <s:param name="key2" value="value2"/>
   </s:component>
```

Non-Form UI Tags 3-4

```
<!- Velocity -->
    #s-component( "template=/my/custom/component.vm" )
      or
    #s-component( "template=/my/custom/component.vm" )
      #s-param( "name=key1" "value=value1" )
      #s-param( "name=key2" "value=value2" )
    #end
<!- Freemarker -->
   <@s..component template="/my/custom/component.ftl" />
     or
   <@s..component template="/my/custom/component.ftl">
      <@s..param name="key1" value="%{'value1'}" />
      <@s..param name="key2" value="%{'value2'}" />
   </@s..component>
```

Non-Form UI Tags 4-4

- **Div** generates an HTML div that loads its content using an ajax call, via the jQuery framework.
- Following code snippet shows an example for `div` tag:

```
  <%@ taglib prefix="s" uri="/struts-tags"%>
<%@ taglib prefix="sj" uri="/struts-jquery-tags"%>
<html>
  <head>
    <sj:head/>
  </head>
  <body>
    <div id="div1">Div 1</div>
    <s:url var="ajaxTest" value="/AjaxTest.action"/>
    <sj:div href="%{ajaxTest}">Initial Content</sj:div>
  </body>
</html>
```

- **Fielderror** tag renders field errors if they exists.

Ajax UI Tags 1-6

- Ajax tag is the new tag implemented in Struts 2 framework.
- In Struts 2, DOJO framework is used for the Ajax tag implementation.
- Commonly used Ajax tags are autocompleter, bind, head, div, submit, tree, and treenode.
- To use Ajax tags, you add the tag library in the JSP page as, `<%@ taglib prefix="sx" uri="/struts-dojo-tags" %>` to JSP page.
- The head tag included on the page, can be configured for performance or debugging purposes.

Ajax UI Tags 2-6

- **Autocompleter** tag is a combo box that can autocomplete text entered on the input box.
- To create autocompleter component in Struts 2, you have to follow two steps:
  - Add `struts2-dojo-plugin.jar` in your class path.
  - Include the `struts-dojo-tags` tag and its header in the jsp page.

Ajax UI Tags 3-6

- **Bind** tag generates event listeners for multiple events on multiple sources, making an asynchronous request to the specified `href`, and updating multiple targets.
- Following code snippet shows the example for `bind` tag:

```
<img id="loadingImage" src="images/loadingAnimation.gif"
style="display:none"/>
<s:div id="parentDiv">
    <s:form action="actionName">
        <s:submit id="btn" />
        <sx:bind src="btn" events="onclick" targets="parentDiv"
showLoadingText="false" indicator="loadingImage"/>
    </s:form>
</s:div>
```

Ajax UI Tags 4-6

- **Div** tag generates an HTML div that loads its content using an XMLHttpRequest call, via the dojo framework.
- Following code snippet shows the example for `div` tag:

```
<img id="indicator"
src="${pageContext.request.contextPath}/images/indicator.gif"
style="display:none"/>
<sx:div href="%{#url}" updateFreq="2000" indicator="indicator">
  Initial Content
</sx:div>
```

Ajax UI Tags 5-6

- **Submit** tag renders a submit button that can submit a form asynchronously.
- The `submit` tag have three different types of rendering:
  - `input` renders as html `<input type="submit"...>`
  - `image` renders as html `<input type="image"...>`
  - `button` renders as html `<button type="submit"...>`
- Following code snippet shows the example for `submit` tag:

```
<sx:submit type="image" value="%{'Submit'}" label="Submit the form"
src="submit.gif"/>
```

Ajax UI Tags 6-6

- The `tree` and `treenode` Ajax tags render a tree node within a tree widget with AJAX support.
- The `tree` and `treenode` of the two combinations are used depending on the requirement like the tree is needed to be constructed dynamically or statically.
- `tree` widget normally uses the '`id`' attribute. The 'id' attribute is required, if the 'selectedNotifyTopic' or the 'href' attribute is going to be used.
- `treenode` renders a tree node within a tree widget with AJAX support.

Summary

- In Struts 2 framework, an Interceptor intercepts the request and processes the request before and after the execution of action and result.
- Interceptor Stacks save time as user do not have to repeatedly write the same list of Interceptors in every action mapping.
- Interceptor class acts as a reusable component and is used in different Web applications.
- Struts 2 framework uses a set of tags for data reference.
- Struts generic tags control the execution flow when pages are rendered and extract the data.
- Control tags control the flow of page execution.
- Data manipulation is done with the help of Data tags.
- Struts UI tags display the data on the HTML page and use the data from value stack.