**Struts 2 FrameWork Tutorial**

A framework tries to automate the common tasks and provides a platform for the users to build applications quickly.

Struts 2 is based on the OpenSymphony **Web Works Framework**.

Struts 2 framework implements the Model-View-Controller (**MVC**) design pattern.

In Struts 2 the model, view and controller are implemented by the **action**, **result** and **FilterDispatcher** respectively.

The controller's job is to map the user request to appropriate action.

In Struts 2 FilterDispatcher does the job of Controller.

Model contains the data and the business logic.

In Struts 2 the model is implemented by the Action component.

View is the presentation component of the MVC Pattern.

In Struts 2 the view is commonly implemented using JSP, Velocity Template, Freemaker or some other presentation-layer technology.

The controller receives the user request and determine which Struts 2 action to invoke.

The framework creates an instance of this action and associate it with the newly created instance of the ActionInvocation.

In Struts 2 the invocation of action should pass through a series of interceptors as defined in the application's XML file.

The framework calls the ActionInvocations invoke() method to start the execution of the action.

Each time the invoke() method is called, ActionInvocation consults its state and executes whichever interceptor comes next.

ActionInvocation hands control over to the interceptor in the stack by calling the interceptors intercept() method.

The intercept() method of the interceptor inturn calls the invoke() method of the ActionInvocation till all the interceptors are invoked, in the end the action itself will be called and the corresponding result will be returned back to the user.

Some interceptor do work before the action is executed and some do work after the action is executed. It's not necessary that it should do something each time it is invoked.

These interceptors are invoke both before and after the action.

First all the interceptors are executed in the order they are defined in the stack.

Then the action is invoked and the result is generated.

Again all the interceptors present in the stack are invoked in the reverse order.

The other important features of Struts 2 are **OGNL** and **ValueStack**.

Object-Graph Navigation Language (OGNL) is a powerful expression language that is used to reference and manipulate data on the ValueStack.

OGNL help in **data transfer** and **type conversion**.

OGNL expression language provides simplified syntax to reference java objects.

OGNL is used to bind the java-side data properties to the string-based view layer.

**What are Pull-MVC and push-MVC based architecture ? Which** **architecture does Struts2 follow ?**

Pull-MVC and Push-MVC are better understood with how the view layer is getting data i.e. Model to render. In case of Push-MVC the data( Model) is constructed and given to the view layer by the Controllers. Typical example is Spring MVC and Struts1. Pull-MVC on the other hand puts the model data typically constructed in Controllers are kept in a common place , which then gets rendered by view layer. Struts2 is a Pull-MVC based architecture, in which all data is stored in Value Stack and retrieved by view layer for rendering.

**Are Interceptors and Filters different ? , If yes then how ?**

**Which class is the front-controller in Struts2 ?**

The class "org.apache.struts2.dispatcher.FilterDispatcher " is the front controller in Struts2. In recent time *Struts 2.1.3* this class is deprecated and new classes are introduced to do the job. Refer:  <http://struts.apache.org/2.1.8/struts2-core/apidocs/org/apache/struts2/dispatcher/FilterDispatcher.html>

**Can I have more than one struts.xml file for a single Struts application?**

The struts.xml file allows to break big struts.xml file into small files and configuration files to be included as needed. Here is the example:

 <struts>  
.....  
......  
<include file="file1.xml"/>  
<include file="file2.xml"/>  
.....  
.....  
</struts>

|  |
| --- |
|  |

**Structure of the struts.xml file**

In the last section we developed and tested the Hello World application. Here is the sample struts.xml file from the last example.

*<?xml version="1.0" encoding="UTF-8" ?>  
<!DOCTYPE struts PUBLIC  
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"  
"http://struts.apache.org/dtds/struts-2.0.dtd">  
  
<struts>  
  
<constant name="struts.enable.DynamicMethodInvocation" value="false" />  
<constant name="struts.devMode" value="true" />  
  
<package name="roseindia" namespace="/roseindia" extends="struts-default">  
  
<action name="HelloWorld" class="net.roseindia.Struts2HelloWorld">  
<result>/pages/HelloWorld.jsp</result>  
</action>  
  
<!-- Add actions here -->  
</package>  
  
  
<!-- Add packages here -->  
  
</struts>*

It is possible to remove the “struts.xml” file from your application completely if the functionality of your application does not depends on it. There are few configurations that can be handled alternatively such as annotations, “web.xml” startup parameters, and alternate URL mapping schemes. Still, there are few configurations that always need the “struts.xml” file like the global results, exception handling, and the custom interceptor stacks.

***Link for more detail:*** <http://www.roseindia.net/struts/struts2/struts-xml.shtml>

**Struts2 Validation:**

Validation depends on both the validation and workflow interceptors (both are included in the default interceptor stack). The validation interceptor does the validation itself and creates a list of field-specific errors. The workflow interceptor checks for the presence of validation errors: if any are found, it returns the "input" result (by default), taking the user back to the form which contained the validation errors.

If we're using the default settings *and* our action doesn't have an "input" result defined *and* there are validation (or, incidentally, type conversion) errors, we'll get an error message back telling us there's no "input" result defined for the action.

If you are defining your own interceptor don’t forget to include workflow and validation interceptor from struts-defaut.xml, and if you want to ignore validation for a particular method use *SkipValidation* interceptor.

There are basically two ways to validate a form in Struts2.

1. Add validate Method

The workflow interceptor controls the flow of cotrol.

The workflow interceptor checks whether the action implements the Validateable interface , if it does, the workflow interceptor will invoke the validate() method of the Action class.

In the validate() method we validate the user name and the password. If the validation fails an error is added using the addFiledError() method.

The validate() method doesn't return any errors, instead it stores all the errors with the help of the ValidationAware interface.

Now the workflow interceptor will check any validation errors has occured. If any error has occured the workflow interceptor will stop the request processing and transfer the control to the input page with the appropriate error messages.

1. Manual validation using XML or Annotation

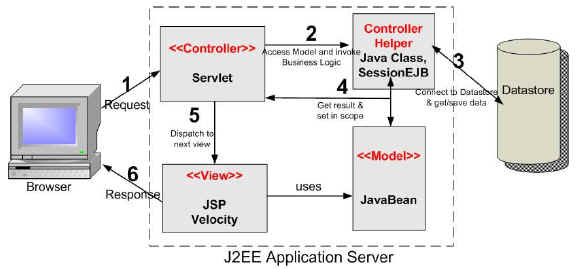
|  |  |  |
| --- | --- | --- |
| **Feature** | **Struts 1** | **Struts 2** |
| Action classes | Struts 1 requires Action classes to extend an abstract base class. A common problem in Struts 1 is programming to abstract classes instead of interfaces. | An Struts 2 Action may implement an Action interface, along with other interfaces to enable optional and custom services. Struts 2 provides a base ActionSupport class to implement commonly used interfaces. Albeit, the Action interface is **not** required. Any POJO object with a execute signature can be used as an Struts 2 Action object. |
| Threading Model | Struts 1 Actions are singletons and must be thread-safe since there will only be one instance of a class to handle all requests for that Action. The singleton strategy places restrictions on what can be done with Struts 1 Actions and requires extra care to develop. Action resources must be thread-safe or synchronized. | Struts 2 Action objects are instantiated for each request, so there are no thread-safety issues. (In practice, servlet containers generate many throw-away objects per request, and one more object does not impose a performance penalty or impact garbage collection.) |
| Servlet Dependency | Struts 1 Actions have dependencies on the servlet API since the HttpServletRequest and HttpServletResponse is passed to the execute method when an Action is invoked. | Struts 2 Actions are not coupled to a container. Most often the servlet contexts are represented as simple Maps, allowing Actions to be tested in isolation. Struts 2 Actions can still access the original request and response, if required. However, other architectural elements reduce or eliminate the need to access the HttpServetRequest or HttpServletResponse directly. |
| Testability | A major hurdle to testing Struts 1 Actions is that the execute method exposes the Servlet API. A third-party extension, Struts TestCase, offers a set of mock object for Struts 1. | Struts 2 Actions can be tested by instantiating the Action, setting properties, and invoking methods. Dependency Injection support also makes testing simpler. |
| Harvesting Input | Struts 1 uses an ActionForm object to capture input. Like Actions, all ActionForms must extend a base class. Since  other JavaBeans cannot be used as ActionForms, developers often create redundant classes to capture input. DynaBeans can used as an alternative to creating conventional ActionForm classes, but, here too, developers may be redescribing existing JavaBeans. | Struts 2 uses Action properties as input properties, eliminating the need for a second input object. Input properties may be rich object types which may have their own properties. The Action properties can be accessed from the web page via the taglibs. Struts 2 also supports the ActionForm pattern, as well as POJO form objects and POJO Actions. Rich object types, including business or domain objects, can be used as input/output objects. The ModelDriven feature simplifies taglb references to POJO input objects. |
| Expression Language | Struts 1 integrates with JSTL, so it uses the JSTL EL. The EL has basic object graph traversal, but relatively weak collection and indexed property support. | Struts 2 can use JSTL, but the framework also supports a more powerful and flexible expression language called "Object Graph Notation Language" (OGNL). |
| Binding values into views | Struts 1 uses the standard JSP mechanism for binding objects into the page context for access. | Struts 2 uses a "ValueStack" technology so that the taglibs can access values without coupling your view to the object type it is rendering. The ValueStack strategy allows reuse of views across a range of types which may have the same property name but different property types. |
| Type Conversion | Struts 1 ActionForm properties are usually all Strings. Struts 1 uses Commons-Beanutils for type conversion. Converters are per-class, and not configurable per instance. | Struts 2 uses OGNL for type conversion. The framework includes converters for basic and common object types and primitives. |
| Validation | Struts 1 supports manual validation via a validate method on the ActionForm, or through an extension to the Commons Validator. Classes can have different validation contexts for the same class, but cannot chain to validations on sub-objects. | Struts 2 supports manual validation via the validate method and the XWork Validation framework. The Xwork Validation Framework supports chaining validation into sub-properties using the validations defined for the properties class type and the validation context. |
| Control Of Action Execution | Struts 1 supports separate Request Processors (lifecycles) for each module, but all the Actions in the module must share the same lifecycle. | Struts 2 supports creating different lifecycles on a per Action basis via Interceptor Stacks. Custom stacks can be created and used with different Actions, as needed. |

[**http://struts.apache.org/2.1.6/docs/comparing-struts-1-and-2.html**](http://struts.apache.org/2.1.6/docs/comparing-struts-1-and-2.html)

**MVC Architecture:**

The main aim of the MVC architecture is to separate the business logic and application data from the presentation data to the user.

They are **resuable** : When the problems recurs, there is no need to invent a new solution, we just have to follow the pattern and adapt it as necessary.

****

1).  **Model:** The model object knows about all the data that need to be displayed. It is model who is aware about all the operations that can be applied to transform that object. It only represents the data of an application. The model represents enterprise data and the business rules that govern access to and updates of this data. Model is not aware about the presentation data and how that data will be displayed to the browser.

2). **View :** The view represents the presentation of the application. The view object refers to the model. It uses the query methods of the model to obtain the contents and renders it. The view is not dependent on the application logic. It remains same if there is any modification in the business logic. In other words, we can say that it is the responsibility of the view's to maintain the consistency in its presentation when the model changes.

3). **Controller:**  Whenever the user sends a request for something then it always go through the controller. The controller is responsible for intercepting the requests from view and passes it to the model for the appropriate action. After the action has been taken on the data, the controller is responsible for directing the appropriate view to the user. In  GUIs, the views and the controllers often work very closely together.

**Features of MVC1:**

1. Html or jsp files are used to code the presentation. To retrieve the data JavaBean can be used.
2. In mvc1 archictecture all the view, control elements are implemented using Servlets or Jsp.
3. In MVC1 there is tight coupling between page and model as data access is usually done using Custom tag or through java bean call.

**Features of MVC2:**

1. The MVC2  architecture removes the page centric property of MVC1 architecture by separating Presentation, control logic and the application state.
2. In MVC2 architecture there is only one controller which receives all the request for the application and is responsible for taking appropriate action in response to each request.

The basic diff bwMVC1 and MVC2 is MVC1 combines controller and View together and MVC2 or Model two separates them out.

**Struts 2 Namespace**

Struts 2 Namespace is a new concept to handle the multiple modules by given a namespace to each module. In addition, it can used to avoid conflicts between same action names located at different modules.

**<struts>**

**<package** name="default" namespace="/" extends="struts-default"**>**

**<action** name="SayWelcome"**>**

**<result>**pages/welcome.jsp**</result>**

**</action>**

**</package>**

**<package** name="common" namespace="/common" extends="struts-default"**>**

**<action** name="SayWelcome"**>**

**<result>**pages/welcome.jsp**</result>**

**</action>**

**</package>**

**<package** name="user" namespace="/user" extends="struts-default"**>**

**<action** name="SayWelcome"**>**

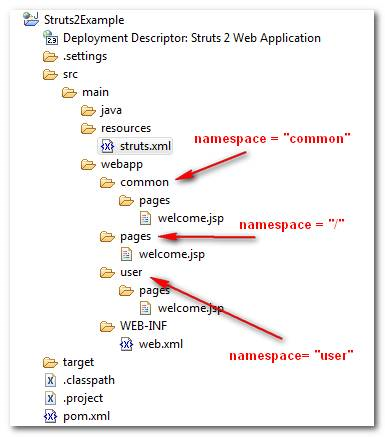
**<result>**pages/welcome.jsp**</result>**

**</action>**

**</package>**

**</struts>**

Struts 2 action namespace map to folder structure.



**Struts 2 – Development mode**

To enable the Struts 2 development mode, you can significantly increase your Struts 2 development speed by giving **auto configuration and properties files reload** and **extra logging and debugging** feature.

The auto reload feature is really a convenient feature.

Each time i made changed in properties or XML configuration file, the application is no longer need to restart to take effect.

By default, the Struts 2 development mode is disabled.

Set the “**struts.devMode**” value to true, either in Struts properties file or XML configuration file.

**struts.properties**

struts.devMode = true

**struts.xml**

**<struts>**

**<constant** name="struts.devMode" value="true" **/>**

**</struts>**

**Action suffix extension in Struts 2**

In Struts 2, all action class has a default suffix **.action** extension. For example,

**<struts>**

**<package** name="default" namespace="/" extends="struts-default"**>**

**<action** name="SayStruts2"**>**

**<result>**pages/printStruts2.jsp**</result>**

**</action>**

**</package>**

**</struts>**

To access the “SayStruts2″ action class, use the following URL :

Action URL: <http://localhost:8080/Struts2Example/SayStruts2.action>

Struts 2 is allow to configure the action extension easily, to change it, just declare a constant “**struts.action.extension**” value :

**<struts>**

**<constant** name="struts.action.extension" value="html"**/>**

**<package** name="default" namespace="/" extends="struts-default"**>**

**<action** name="SayStruts2"**>**

**<result>**pages/printStruts2.jsp**</result>**

**</action>**

**</package>**

**</struts>**

Now you can access the “SayStruts2″ action class via

Action URL: http://localhost:8080/Struts2Example/SayStruts2.html

We can also make the **value as “”** than this action can be addressed by Action name only.

Action URL: [http:**//**localhost:8080**/**Struts2Example**/**SayStruts2](http://localhost:8080/Struts2Example/SayStruts2)