Struts 2

Apache Struts 2 is an elegant, extensible framework for creating enterprise-ready Java web applications. The framework is designed to streamline the full development cycle, from building, to deploying, to maintaining applications over time.

MVC Architecture

Model **V**iew **C**ontroller or MVC as it is popularly called, is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

- **Model** - The lowest level of the pattern which is responsible for maintaining data.
- **View** - This is responsible for displaying all or a portion of the data to the user.
- **Controller** - Software Code that controls the interactions between the Model and View.

The model

The model is responsible for managing the data of the application. It responds to the request from the view and it also responds to instructions from the controller to update itself.

The view

A presentation of data in a particular format, triggered by a controller's decision to present the data. They are script based templating systems like JSP, ASP, PHP and very easy to integrate with AJAX technology.

The controller

The controller is responsible for responding to user input and perform interactions on the data model objects. The controller receives the input, it validates the input and then performs the business operation that modifies the state of the data model.

Struts 2

Components & Description

1. Action

Create an action class which will contain complete business logic and control the interaction between

the user, the model, and the view.

2. Interceptors

Create interceptors if required, or use existing interceptors. This is part of Controller.

3. View

Create a JSPs to interact with the user to take input and to present the final messages.

4. Configuration Files

Create configuration files to couple the Action, View and Controllers. These files are struts.xml, web.xml, struts.properties.

Steps:

1. Create Action Class – by implementing Action Interface or Extending ActionSupport class
2. Create a View JSP
3. Setup Configuration file : struts.xml

```
<?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.devMode" value="true" />
    <package name="helloworld" extends="struts-default">

        <action name="hello"
                class="com.h2kinfosys.struts2.HelloWorldAction"
                method="execute">
                <result name="success">/HelloWorld.jsp</result>
        </action>
    </package>
</struts>
```

4. Add entry in web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">

    <display-name>Struts 2</display-name>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <filter>
        <filter-name>struts2</filter-name>
        <filter-class>
            org.apache.struts2.dispatcher.FilterDispatcher
        </filter-class>
    </filter>

    <filter-mapping>
        <filter-name>struts2</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
</web-app>
```

Run the code now.

Struts 2 - Actions

Actions are the core of the Struts2 framework, as they are for any MVC (Model View Controller) framework. Each URL is mapped to a specific action, which provides the processing logic necessary to service the request from the user.

The only requirement for actions in Struts2 is that there must be one no-argument method that returns

either a String or Result object and must be a POJO. If the no-argument method is not specified, the default behavior is to use the execute() method.

Optionally you can extend the **ActionSupport** class which implements six interfaces including**Action** interface.

Struts 2 - Interceptors

Interceptors are conceptually the same as servlet filters or the JDKs Proxy class. Interceptors allow for crosscutting functionality to be implemented separately from the action as well as the framework. You can achieve the following using interceptors:

- Providing preprocessing logic before the action is called.
- Providing postprocessing logic after the action is called.
- Catching exceptions so that alternate processing can be performed.

Struts2 Framework Interceptors

Interceptor & Description

1 alias

Allows parameters to have different name aliases across requests.

2 checkbox

Assists in managing check boxes by adding a parameter value of false for check boxes that are not checked.

3 conversionError

Places error information from converting strings to parameter types into the action's field errors.

4 createSession

Automatically creates an HTTP session if one does not already exist.

5 debugging

Provides several different debugging screens to the developer.

6 execAndWait

Sends the user to an intermediary waiting page while the action executes in the background.

7 exception

Maps exceptions that are thrown from an action to a result, allowing automatic exception handling via redirection.

8 fileUpload

Facilitates easy file uploading.

9 i18n

Keeps track of the selected locale during a user's session.

10 logger

Provides simple logging by outputting the name of the action being executed.

11 params

Sets the request parameters on the action.

12 prepare

This is typically used to do pre-processing work, such as setup database connections.

13 profile

Allows simple profiling information to be logged for actions.

14 scope

Stores and retrieves the action's state in the session or application scope.

15 ServletConfig

Provides the action with access to various servlet-based information.

16 timer

Provides simple profiling information in the form of how long the action takes to execute.

17 token

Checks the action for a valid token to prevent duplicate formsubmission.

18 validation

Provides validation support for actions

How to use Interceptors?

```
<?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.devMode" value="true" />
    <package name="helloworld" extends="struts-default">
        <action name="hello"
            class="com.h2kinfosys.struts2.HelloWorldAction"
            method="execute">
            <interceptor-ref name="params"/>
            <interceptor-ref name="timer" />
            <result name="success">/HelloWorld.jsp</result>
        </action>
    </package>
</struts>
```

Create Custom Interceptors

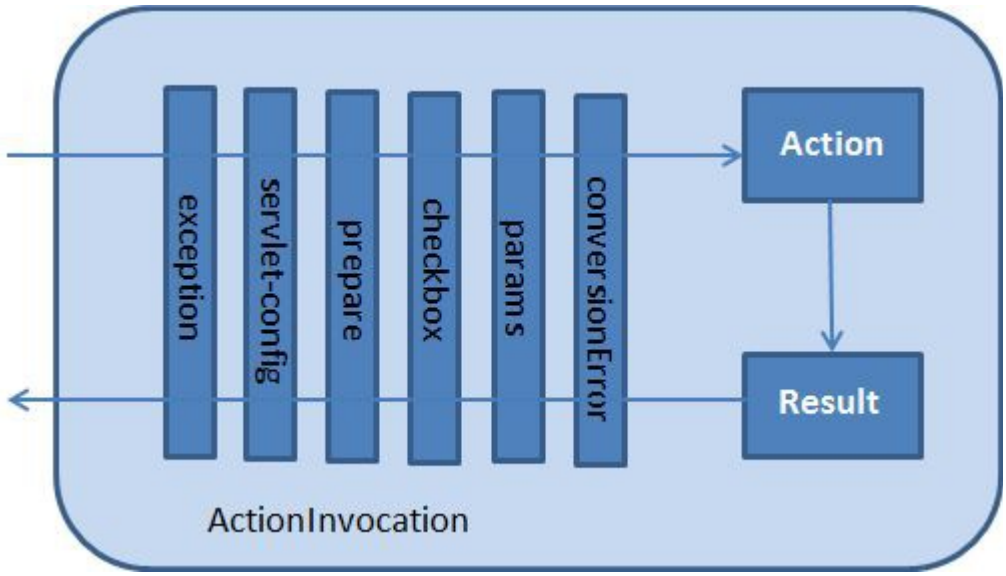
Using custom interceptors in your application is an elegant way to provide cross-cutting application features. Creating a custom interceptor is easy; the interface that needs to be extended is the following **Interceptor** interface:

```
public interface Interceptor extends Serializable{
    void destroy();
    void init();
    String intercept(ActionInvocation invocation)
    throws Exception;
}
```

If you have no need for initialization or cleanup code, the **AbstractInterceptor** class can be extended. This provides a default no-operation implementation of the init() and destroy() methods.

```
import java.util.*;
```

```java
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.AbstractInterceptor;

public class MyInterceptor extends AbstractInterceptor {

    public String intercept(ActionInvocation invocation)throws Exception{

        /* let us do some pre-processing */
        String output = "Pre-Processing";
        System.out.println(output);

        /* let us call action or next interceptor */
        String result = invocation.invoke();

        /* let us do some post-processing */
        output = "Post-Processing";
        System.out.println(output);

        return result;
    }
}
```



```
<?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.devMode" value="true" />
    <package name="helloworld" extends="struts-default">

        <interceptors>
            <interceptor name="myinterceptor"
                class="com.h2kinfosys.struts2.MyInterceptor" />
        </interceptors>

        <action name="hello"
            class="com.h2kinfosys.struts2.HelloWorldAction"
            method="execute">
            <interceptor-ref name="params"/>
            <interceptor-ref name="myinterceptor" />
            <result name="success">/HelloWorld.jsp</result>
        </action>

    </package>
</struts>
```

Stacking multiple Interceptors

```
<interceptor-stack name="basicStack">
    <interceptor-ref name="exception"/>
    <interceptor-ref name="servlet-config"/>
    <interceptor-ref name="prepare"/>
    <interceptor-ref name="checkbox"/>
    <interceptor-ref name="params"/>
    <interceptor-ref name="conversionError"/>
</interceptor-stack>
```

Struts 2 - Value Stack

The value stack is a set of several objects which keeps the following objects in the provided order:

Objects & Description

1 **Temporary Objects**

There are various temporary objects which are created during execution of a page. For example the current iteration value for a collection being looped over in a JSP tag.

2 **The Model Object**

If you are using model objects in your struts application, the current model object is placed before the action on the value stack

3 **The Action Object**

This will be the current action object which is being executed.

4 **Named Objects**

These objects include #application, #session, #request, #attr and #parameters and refer to the corresponding servlet scopes

Using Value Stack:

```
import java.util.*;

import com.opensymphony.xwork2.util.ValueStack;
import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;

public class HelloWorldAction extends ActionSupport{
    private String name;

    public String execute() throws Exception {
        ValueStack stack = ActionContext.getContext().getValueStack();
        Map<String, Object> context = new HashMap<String, Object>();
```

```
        context.put("key1", new String("This is key1"));
        context.put("key2", new String("This is key2"));
        stack.push(context);

        System.out.println("Size of the valueStack: " + stack.size());
        return "success";
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Create View to access them:

```
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<head>
<title>Hello World</title>
</head>
<body>
    Entered value : <s:property value="name"/><br/>
    Value of key 1 : <s:property value="key1" /><br/>
    Value of key 2 : <s:property value="key2" /> <br/>
</body>
</html>
```

Validations Framework

Now we will look into how Struts's validation framework. At Struts's core, we have the validation framework that assists the application to run the rules to perform validation before the action method is executed.

Client side validation is usually achieved using Javascript. But one should not rely upon client side validation alone. Best practise suggests that the validation should be introduced at all levels of your application framework. Now let us look at two ways of adding validation to our Struts project.

Create a View:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>Employee Form</title>
</head>

<body>
    <s:form action="empinfo" method="post">
        <s:textfield name="name" label="Name" size="20" />
        <s:textfield name="age" label="Age" size="20" />
        <s:submit name="submit" label="Submit" align="center" />
    </s:form>
</body>
</html>
```

Create Action

```
import com.opensymphony.xwork2.ActionSupport;

public class Employee extends ActionSupport{
    private String name;
    private int age;

    public String execute()
```

```
    {
         return SUCCESS;
    }
    public String getName() {
         return name;
    }
    public void setName(String name) {
         this.name = name;
    }
    public int getAge() {
         return age;
    }
    public void setAge(int age) {
         this.age = age;
    }

    public void validate()
    {
        if (name == null || name.trim().equals(""))
        {
            addFieldError("name","The name is required");
        }
        if (age < 28 || age > 65)
        {
            addFieldError("age","Age must be in between 28 and 65");
        }
    }
}
```

Struts.xml

```
<?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.devMode" value="true" />
    <package name="helloworld" extends="struts-default">

        <action name="empinfo"
            class="com.h2kinfosys.struts2.Employee"
            method="execute">
            <result name="input">/index.jsp</result>
            <result name="success">/success.jsp</result>
        </action>

    </package>

</struts>
```

Struts 2 - Exception Handling

Struts provides an easier way to handle uncaught exception and redirect users to a dedicated error page. You can easily configure Struts to have different error pages for different execeptions.

Struts.xml

```
<?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.devMode" value="true" />
    <package name="helloworld" extends="struts-default">

        <action name="hello"
            class="com.h2kinfosys.struts2.HelloWorldAction"
            method="execute">
            <exception-mapping exception="java.lang.NullPointerException"
```

```
            result="error" />
            <result name="success">/HelloWorld.jsp</result>
            <result name="error">/Error.jsp</result>
        </action>

    </package>
</struts>
```