STRUTS2 INTRODUCTION

Struts 2.x is an open source invasive, MVC based framework which facilitate productive development of dynamic web applications.

MVC is a design pattern which is based on the concept of separation of concern and loose coupling. This design pattern states that functionality of an application should be divided into 3 loosely joined components called Model, View and Controller.

Model represent data or state, View represents presentation and Controller represents work flow.

Advantage of MVC :
1) Maintenance of the application is simplified because of separation of Concern.
2) Reusability is increased as same Model can be used with multiple views or vice versa.
3) Scalability of the application is improved.

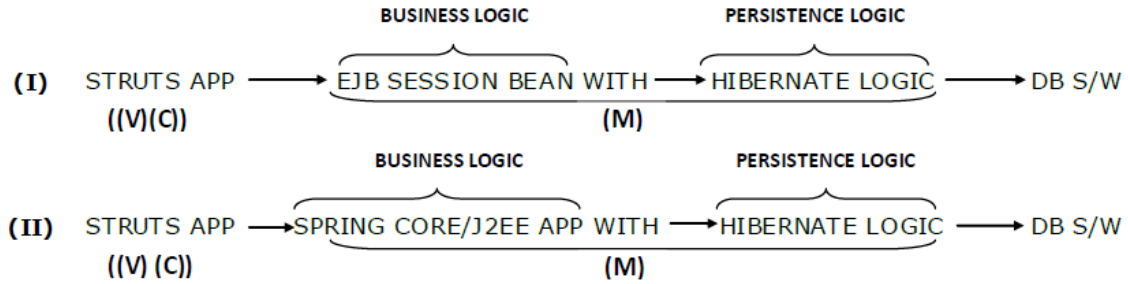
A framework such as struts provide implementation of the design pattern i.e framework provides pre-defined classes and interfaces which facilitate development of applications using a specific architecture.

Understanding MVC Implementation in Struts

1) Model in struts2 is represented by action. An Action is a POJO class. POJO (Plain Old Java Object) which contains data member to store request data and processing result, their getter and setter methods and one or more request processing methods.

2) View in struts2 support different presentation technology for presenting data to the end user. Common presenting technologies are JSP, Velocity, Freemarker, XSLT etc. To manage these different present technologies, predefined component called Result is provided by the framework.

3) Controller in struts is implemented as a filter. It acts as a single entry point into the application. i.e. it intercepts all the requests and controls the flow of their processing.

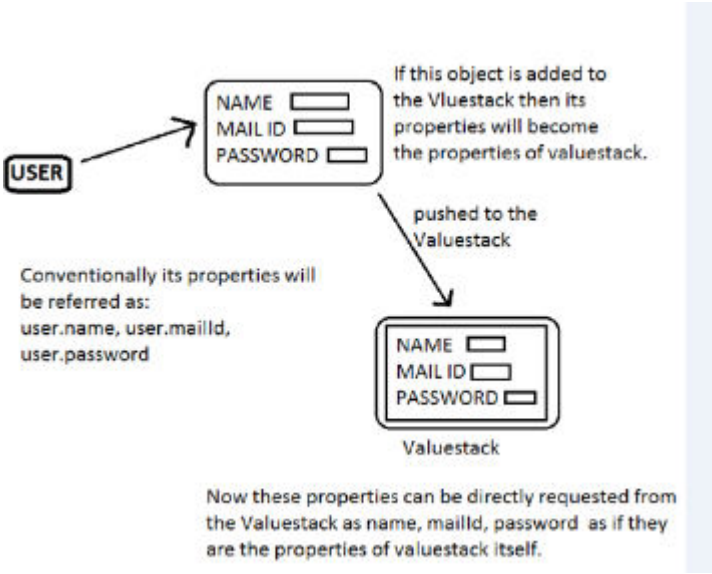


In addition to these main components Struts2 application uses following components as well.

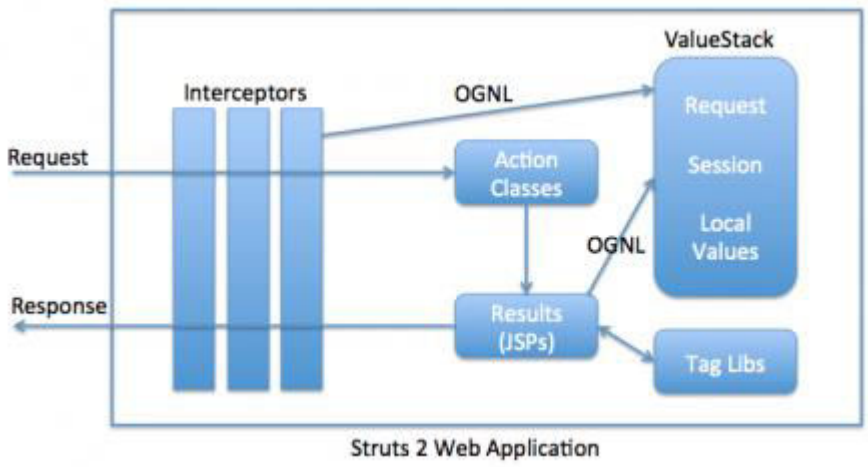
Value set is a helper component which facilitate sharing of data among the action, interceptors and view components. For each request a Value set Object is created by controller.

This Object has a special characterstic it exposes the properties of objects added to it as the properties of its own.

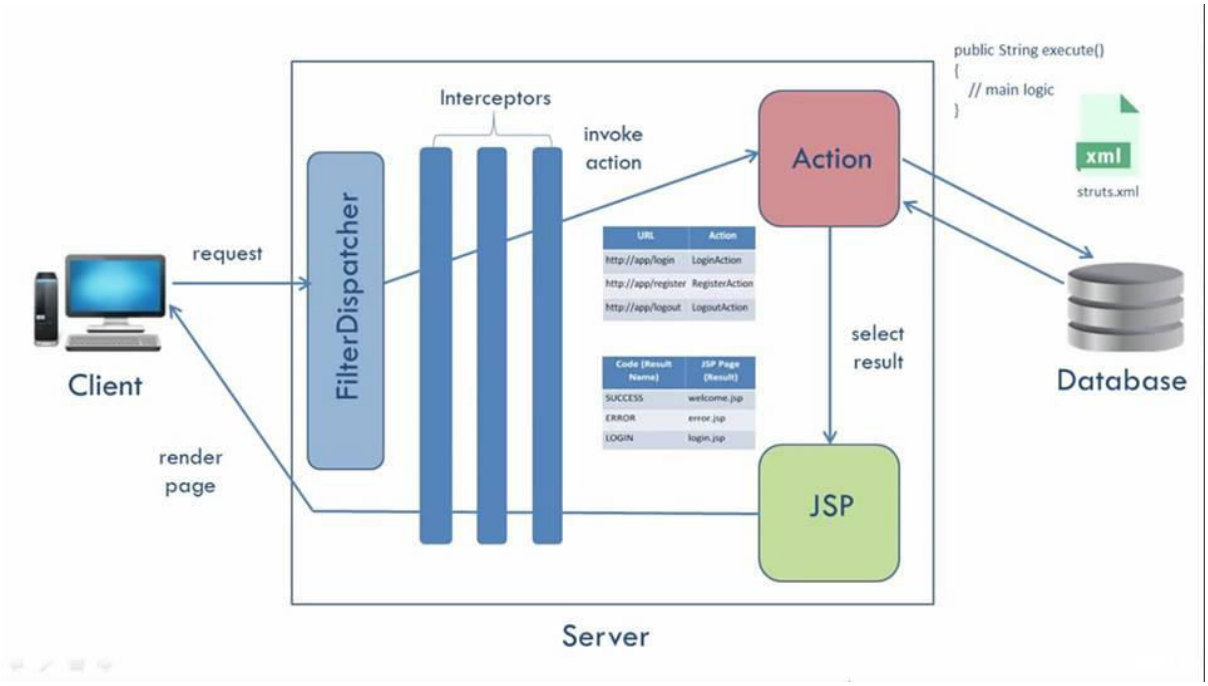
Let following be user object :



Struts 2 Architecture



Request Lifecycle in Struts 2 applications



- ✓ **User Sends request**: User sends a request to the server for some resource.
- ✓ **FilterDispatcher determines the appropriate action**: The FilterDispatcher looks at the request and then determines the appropriate Action.
- ✓ **Interceptors are applied**: Interceptors configured for applying the common functionalities such as workflow, validation, file upload etc. are automatically applied to the request.
- ✓ **Execution of Action**: Then the action method is executed to perform the database related operations like storing or retrieving data from the database.
- ✓ **Output rendering**: Then the Result renders the output.
- ✓ **Return of Request**: Then the request returns through the interceptors in the reverse order. The returning request allows us to perform the clean-up or additional processing.
- ✓ **Display the result to user**: Finally the control is returned to the servlet container, which sends the output to the user browser.

web.xml

```
<filter>
        <filter-name>struts2</filter-name>
        <filter-
class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filte
r-class>
    </filter>

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

In a development environment, there are a couple of properties that you might consider changing:

- ✓ struts.i18n.reload = true – enables reloading of internationalization files
- ✓ struts.devMode = true – enables development mode that provides more comprehensive debugging
- ✓ struts.configuration.xml.reload = true – enables reloading of XML configuration files (for the action) when a change is made without reloading the entire web
- ✓ application in the servlet container
- ✓ struts.url.http.port = 8080 – sets the port that the server is run on (so that generated URLs are created correctly)

Struts2.xml

- controls the execution flow of requests
- Input: URL
- Needs to map to
  - ✓ Action classes
  - ✓ JSPs

URL to Action class

http://localhost:8080/getTutorial -> TutorialAction

http://localhost:8080/getBook -> BookAction

Which method of Action class it executes?

- ✓ Default method of Action class is execute()
- ✓ Other methods can be configured

URL	Action Class	Code	JSP
http://mywebapp/getTutorial	TutorialAction	Success	success.jsp
		Failure	error.jsp
		NoSession	login.jsp

- ✓ JSP display depends on the result of method execution
- ✓ Same Action class may need to show different JSPs

Understanding namespaces

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts  Configuration
2.1//EN" "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>
	<package name="default" namespace="/tutorials" extends="struts-default">
		<action name="getTutorial"
class="com.praveen.action.TutorialAction">
			<result name="success">/success.jsp</result>
		</action>
	</package>
</struts>
```

URL will be **http://localhost:8080/applicationname/tutorials/getTutorial**

Action class data access in JSP

For example,

```
package com.praveen.action;

import com.praveen.service.TutorialFinderService;

public class TutorialAction{
	private static final long serialVersionUID = 3748973386879929793L;
	private String bestTutorialSite;

	public String execute() {
		TutorialFinderService tutfs= new TutorialFinderService();
		setBestTutorialSite(tutfs.getBestTutorialSite());
		return "success";
	}

	public String getBestTutorialSite() {
		return bestTutorialSite;
	}

	public void setBestTutorialSite(String bestTutorialSite) {
		this.bestTutorialSite = bestTutorialSite;
	}

	public static long getSerialversionuid() {
		return serialVersionUID;
	}

}


package com.praveen.service;

public class TutorialFinderService {

	public String getBestTutorialSite() {
		return "praveenoruganti";
	}
}
```

success.jsp

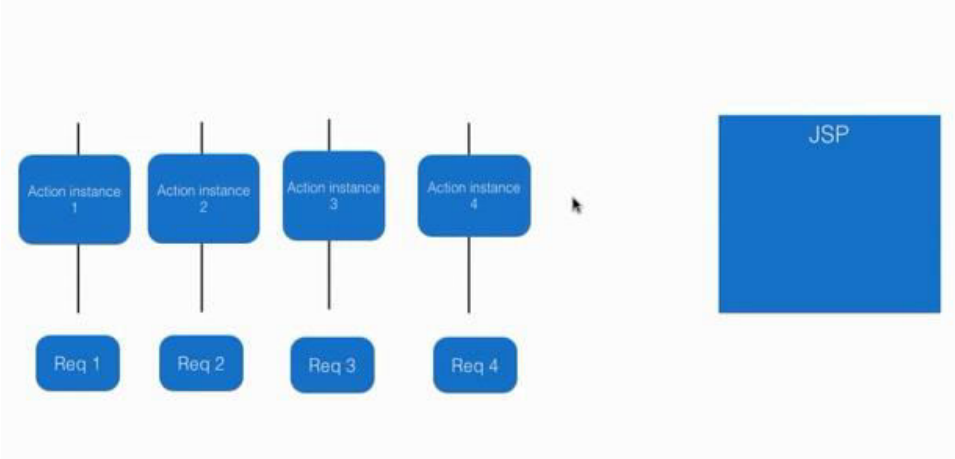
```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
 <%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html;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

```
<s:property value="bestTutorialSite"/>
</body>
</html>
```

Here in success.jsp, bestTutorialSite value present in Action is accessed through s:property tag.

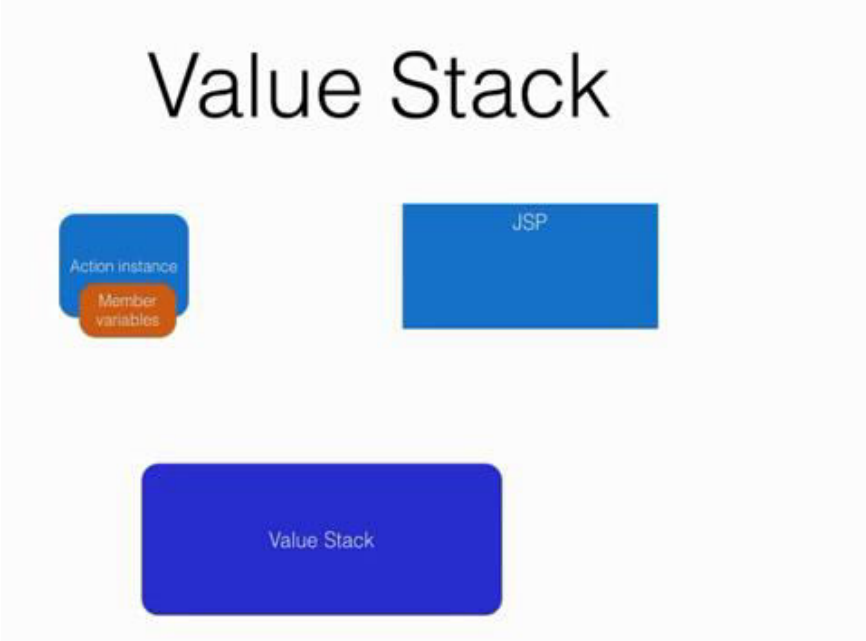
How this actually works?

Like Servlets, Struts2 don't have a threaded model so, new Action objects will be created for each request. Advantage of this model is we can have member variables in Action classes which there by can be accessed in JSP.



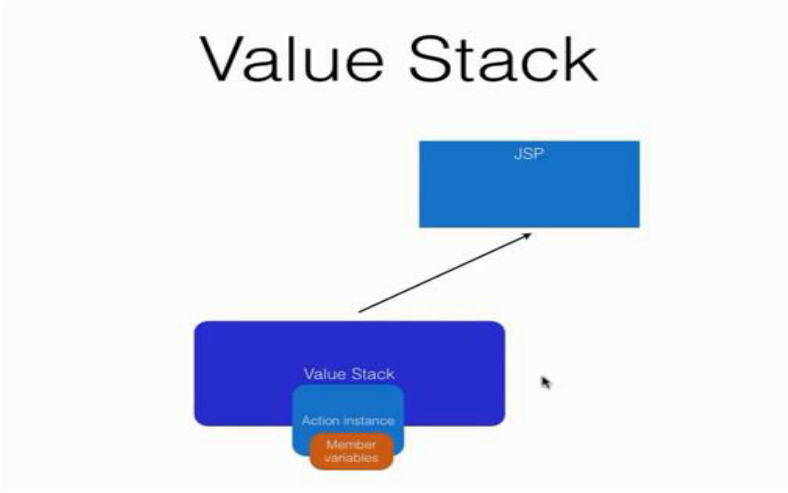
How this is actually accessed in JSP?

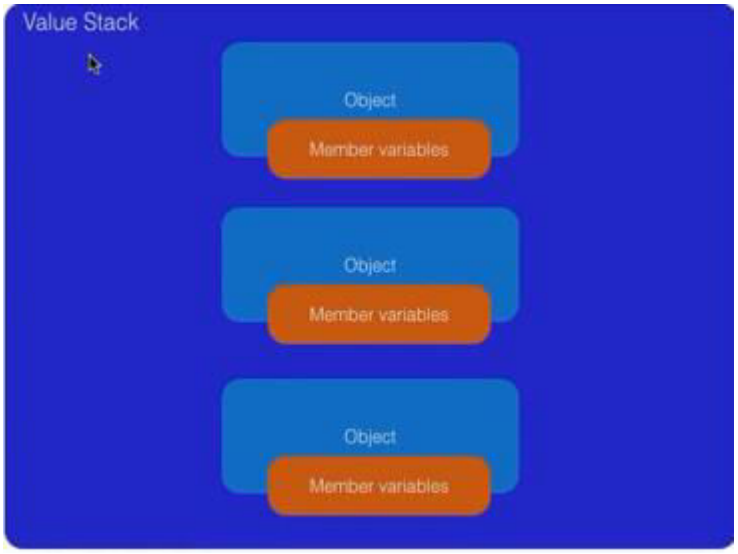
In Struts2, this happens using Value Stack.

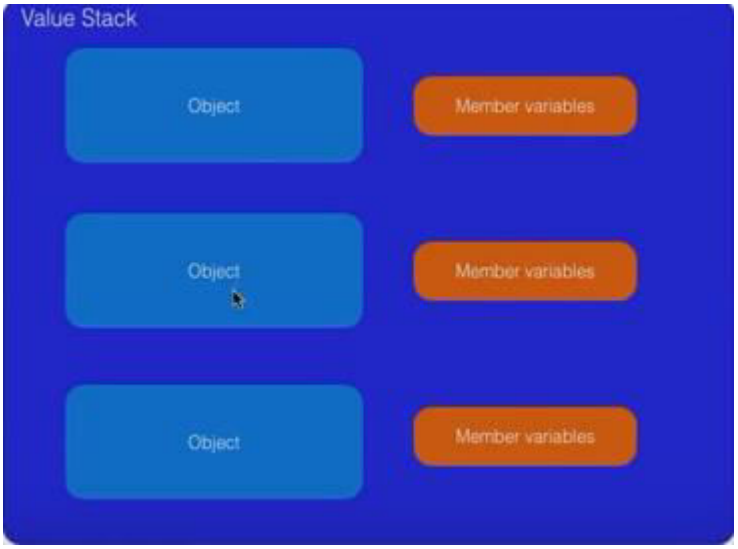


Struts 2 ValueStack and OGNL

ValueStack is the storage area where the application data is stored by Struts 2 for processing a client request. The data is stored in ActionContext objects that use ThreadLocal to have values specific to the particular request thread.







Value Stack has following objects:

- ✓ Temporary Objects
- ✓ Model Object
- ✓ Action Object
- ✓ Named Objects

Control Tags along with expression are used to access properties of objects in value stack.

Data Tags are used to manipulate the value stack.

Important points on Value Stack

- ✓ It is not really a stack in the traditional sense.
- ✓ It does stack the objects.
- ✓ Behaves like a virtual object.
- ✓ Makes it easier to refer the member variables.

It looks variables from top down(ValueStack.fileName), if it needs particular name in object then it goes ValueStack.Object.fileName

Methods of ValueStack interface

There are many methods in ValueStack interface. The commonly used methods are as follows:

- ✓ **public String findString(String expr)** finds the string by evaluating the given expression.
- ✓ **public Object findValue(String expr)** finds the value by evaluating the specified expression.

- ✓ **public Object findValue(String expr, Class c)** finds the value by evaluating the specified expression.
- ✓ **public Object peek()** It returns the object located on the top of the stack.
- ✓ **public Object pop()** It returns the object located on the top of the stack and removes it.
- ✓ **public void push(Object o)** It puts the object on the top of the stack.
- ✓ **public void set(String key, Object value)** It sets the object on the stack with the given key. It can be get by calling the findValue(key) method.
- ✓ **public int size()** It returns the number of objects from the stack.

Object-Graph Navigation Language (OGNL) is a powerful Expression Language that is used to manipulate data stored on the ValueStack. As you can see in architecture diagram, both interceptors and result pages can access data stored on ValueStack using OGNL.

OGNL uses dot notation to navigate object graphs.

OGNL also supports following features:

- ✓ type conversion
- ✓ calling methods
- ✓ collection manipulation and generation
- ✓ projection across collections
- ✓ expression evaluation
- ✓ lambda expressions

Accessing Input Parameters in Action Class

Generally in Servlets it happens via GET and POST requests.

In Struts2, this can be achieved using member variables in Action class. This is actually done by Interceptors in background. for example,

```
package com.praveen.action;

import com.praveen.service.TutorialFinderService;

public class TutorialAction{
	private String bestTutorialSite;
	private String language;

	public String execute() {
		TutorialFinderService tutfs= new TutorialFinderService();
		setBestTutorialSite(tutfs.getBestTutorialSite(language));

		return "success";
	}

	public String getBestTutorialSite() {
		return bestTutorialSite;
	}

	public void setBestTutorialSite(String bestTutorialSite) {
		this.bestTutorialSite = bestTutorialSite;
	}

	public String getLanguage() {
		return language;
```

```
	}

	public void setLanguage(String language) {
		this.language = language;
	}

}
package com.praveen.service;

public class TutorialFinderService {

	public String getBestTutorialSite(String language) {
		if(language.equalsIgnoreCase("java")) {
		  return "http://praveenoruganti.wikidot.com";
		}else {
			return "language not supported yet!";
		}
	}
}
```

URL:
http://localhost:8080/praveenstruts2sampleapp/tutorials/getTutorial?language=Java

Post requests to Action

searchForm.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<s:form action="tutorials/getTutorial">
<s:textfield label="Enter the language you want to search for" key="language"/>
<s:submit align="center"/>
</s:form>
```

URL: http://localhost:8080/praveenstruts2sampleapp/searchForm.jsp

Login Action Sample with Best Practices

1. Instead of hard coding as success we are implementing the Action interface so that we can use the constant SUCCCESS while returning.

```
package com.praveen.action;

import com.opensymphony.xwork2.Action;

public class LoginAction implements Action {
	private String userid;
	private String password;

	public String execute() {
```

```
		if (getUserid().equalsIgnoreCase("Praveen") &&
getPassword().equals("Password")) {
			return SUCCESS;
		}
		return LOGIN;
	}

	public String getUserid() {
		return userid;
	}

	public void setUserid(String userid) {
		this.userid = userid;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

}
```

struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts  Configuration
2.1//EN" "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>
	 <package name="product" extends="struts-default">
		<action name="product" class="com.praveen.model.Product">
			<result name="success">welcome.jsp</result>
		</action>
	</package>
	<package name="tutorial" namespace="/tutorials" extends="struts-default">
		<action name="getTutorial"
class="com.praveen.action.TutorialAction">
			<result name="success">/success.jsp</result>
		</action>
	</package>

	<package name="login" namespace="/" extends="struts-default">
		<action name="login" class="com.praveen.action.LoginAction">
			<result name="success">/searchForm.jsp</result>
			<result name="login">/login.jsp</result>
		</action>

	</package>
</struts>
```

2. We need to encapsulate the jsp instead of hard coding in result by creating dummy action like searchForm

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts  Configuration
2.1//EN" "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>
        <package name="product" extends="struts-default">
              <action name="product" class="com.praveen.model.Product">
                    <result name="success">welcome.jsp</result>
              </action>
       </package>
       <package name="tutorial" namespace="/tutorials" extends="struts-default">
              <action name="getTutorial"
class="com.praveen.action.TutorialAction">
                    <result name="success">/success.jsp</result>
              </action>
       </package>

       <package name="login" namespace="/" extends="struts-default">
              <action name="login" class="com.praveen.action.LoginAction">
               <result name="success" type="chain">searchForm</result>
                    <result name="login">/login.jsp</result>
              </action>

       </package>

       <package name="search" namespace="/" extends="struts-default">
              <action name="searchForm">
                    <result>/searchForm.jsp</result>

              </action>

       </package>
</struts>
```

3. For maintainability we can have individual xml's and there by refer the same in struts.xml

login.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts  Configuration
2.1//EN" "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>
       <package name="login" namespace="/" extends="struts-default">
              <action name="login" class="com.praveen.action.LoginAction">
                    <result name="success" type="chain">searchForm</result>
                    <result name="login">/login.jsp</result>
              </action>

       </package>
</struts>
```

struts.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts  Configuration
2.1//EN" "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>
       <package name="product" extends="struts-default">
```

```
		<action name="product" class="com.praveen.model.Product">
			<result name="success">welcome.jsp</result>
		</action>
	</package>
	<package name="tutorial" namespace="/tutorials" extends="struts-default">
		<action name="getTutorial"
class="com.praveen.action.TutorialAction">
			<result name="success">/success.jsp</result>
		</action>
	</package>

	<include file="login.xml"></include>

	<package name="search" namespace="/" extends="struts-default">
		<action name="searchForm">
			<result>/searchForm.jsp</result>

		</action>

	</package>
</struts>
```

Action Wild cards

Wildcards let us map multiple Action names with a single Action mapping. This simplifies the XML mapping configuration, especially if your application uses some sort of standard naming conventions for the Action and JSP names.

```
<package name="search" namespace="/" extends="struts-default">
		<action name="search*">
			<result>/search{1}.jsp</result>
		</action>

	</package>
```

http://localhost:8080/praveenstruts2sampleapp/searchTutorial.jsp

http://localhost:8080/praveenstruts2sampleapp/searchForm.jsp

ActionSupport class

we can write Struts 2 Actions by 3 ways

- ✓ POJO
- ✓ implements Action interface
- ✓ extends ActionSupport class

We can implement validations as a part of ActionSupport

for example,

```
package com.praveen.action;

import org.apache.commons.lang.xwork.StringUtils;

import com.opensymphony.xwork2.ActionSupport;

public class LoginAction extends ActionSupport {
	private static final long serialVersionUID = -1965878195011833360L;
	private String userid;
```

```
	private String password;

	public void validate() {
		if(StringUtils.isEmpty(getUserid())) {
			addFieldError(userid,"userid cannot be blank");
		}

		if(StringUtils.isEmpty(getPassword())) {
			addFieldError(password,"password cannot be blank");
		}
	}

	public String execute() {
		if (getUserid().equalsIgnoreCase("Praveen") &&
getPassword().equals("Password")) {
			return SUCCESS;
		}
		return LOGIN;
	}

	public String getUserid() {
		return userid;
	}

	public void setUserid(String userid) {
		this.userid = userid;
	}

	public String getPassword() {
		return password;
	}

	public void setPassword(String password) {
		this.password = password;
	}

}
```

login.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts  Configuration
2.1//EN" "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>
	<package name="login" namespace="/" extends="struts-default">
		<action name="login" class="com.praveen.action.LoginAction">
			<result name="success" type="chain">searchForm</result>
			<result name="login">/login.jsp</result>
			<result name="input">/login.jsp</result>
		</action>

	</package>
</struts>
```

login.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<s:form action="login">
<s:fielderror/>
<s:textfield label="Loginid" key="userid"/>
<s:password label="Password" key="password" />
<s:submit align="center"/>
</s:form>
```

Configuring methods in Action mappings

if i need to execute any other method(other than execute method)

We can have

```
<package name="tutorial" namespace="/tutorials" extends="struts-default">
		<action name="getTutorial" class="com.praveen.action.TutorialAction"
method="getTutorial">
			<result name="success">/success.jsp</result>
		</action>
		<action name="addTutorial" class="com.praveen.action.TutorialAction"
method="addTutorial">
			<result name="success">/success.jsp</result>
		</action>
	</package>
```

otherwise we can also have wild card

```
<package name="tutorial" namespace="/tutorials" extends="struts-default">
		<action name="*" class="com.praveen.action.TutorialAction"
method="{1}">
			<result name="success">/success.jsp</result>
		</action>
	</package>
```

```
package com.praveen.action;

import com.praveen.service.TutorialFinderService;

public class TutorialAction {
	private String bestTutorialSite;
	private String language;

	public String execute() {
		TutorialFinderService tutfs = new TutorialFinderService();
		setBestTutorialSite(tutfs.getBestTutorialSite(language));

		return "success";
	}

	public String getTutorial() {
		System.out.println("getTutorial");
		return "success";
	}

	public String addTutorial() {
		System.out.println("addTutorial");
		return "success";
	}
```

```
	public String getBestTutorialSite() {
		return bestTutorialSite;
	}

	public void setBestTutorialSite(String bestTutorialSite) {
		this.bestTutorialSite = bestTutorialSite;
	}

	public String getLanguage() {
		return language;
	}

	public void setLanguage(String language) {
		this.language = language;
	}

}
```

http://localhost:8080/praveenstruts2sampleapp/tutorials/addTutorial

http://localhost:8080/praveenstruts2sampleapp/tutorials/getTutorial

Using Model Objects

```
package com.praveen.model;

public class User {

	private String userid;
	private String password;
	public String getUserid() {
		return userid;
	}
	public void setUserid(String userid) {
		this.userid = userid;
	}
	public String getPassword() {
		return password;
	}
	public void setPassword(String password) {
		this.password = password;
	}

}
```

```
package com.praveen.service;

import com.praveen.model.User;

public class LoginService {

	public boolean verifyLogin(User user) {
		if(user.getUserid().equalsIgnoreCase("Praveen") &&
user.getPassword().equalsIgnoreCase("Password") ) {
			return true;
		}
```

```
			return false;
		}
}
```

```
package com.praveen.action;

import org.apache.commons.lang.xwork.StringUtils;

import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;
import com.praveen.model.User;
import com.praveen.service.LoginService;

public class LoginAction extends ActionSupport implements ModelDriven<User> {
	private static final long serialVersionUID = -1965878195011833360L;

	private User user= new User();

	public User getUser() {
		return user;
	}

	public void setUser(User user) {
		this.user = user;
	}

	public void validate() {
		if(StringUtils.isEmpty(user.getUserid())) {
			addFieldError( user.getUserid(),"userid cannot be blank");
		}

		if(StringUtils.isEmpty( user.getPassword())) {
			addFieldError( user.getPassword(),"password cannot be blank");
		}
	}

	public String execute() {
		LoginService loginService= new LoginService();
		if (loginService.verifyLogin(user)) {
			return SUCCESS;
		}
		return LOGIN;
	}

	@Override
	public User getModel() {
		return user;
	}
}
```

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<s:form action="login">
<s:fielderror/>
<s:textfield label="Loginid" key="userid"/>
<s:password label="Password" key="password" />
```

```
<s:submit align="center"/>
</s:form>


<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts  Configuration
2.1//EN" "http://struts.apache.org/dtds/struts-2.5.dtd">
<struts>
	<package name="login" namespace="/" extends="struts-default">
		<action name="login" class="com.praveen.action.LoginAction">
			<result name="success" type="chain">searchForm</result>
			<result name="login">/login.jsp</result>
			<result name="input">/login.jsp</result>
		</action>

	</package>
</struts>
```

INTERCEPTOR

Before going to interceptor, lets recap

We have seen automatic features of Struts2

- ✓ Param transfer to member variables
- ✓ Validation
- ✓ and More....

Who/What does this work? INTERCEPTORS!

Struts2 Interceptors are like Servlet Filters that executes before and after the request is being processed. In Struts 2, interceptor is used to perform operations such as validation, exception handling, internationalization, displaying intermediate result etc.

Advantage of interceptors:
Pluggable If we need to remove any concern such as validation, exception handling, logging etc. from the application, we don't need to redeploy the application. We only need to remove the entry from the struts.xml file.

Struts 2 default interceptors stack:(present in struts-default.xml)

1. ***alias:*** As the name suggests this interceptor allows parameters to have different name between requests.
2. ***autowiring:*** The autowiring interceptor is very useful as it auto wires action classes to Spring beans in your application.
3. ***chain:*** This Interceptor is used to make the properties of previous action available in the current action.
4. ***checkbox:*** This helps to add automatic checkbox handling code that detect an unchecked checkbox.
5. ***cookie:*** This is used to add a cookie to the current action with a certain configurable name or value into action.
6. ***conversionError:*** It adds conversion errors from ActionContext to the action's field errors.
7. ***createSession:*** Automatically creates an HttpSession object if it doesn't exists.
8. ***clearSession:*** This interceptor is used to unbinds the HttpSession object.
9. ***debugging:*** Supports debugging by providing different debugging screens.
10. ***externalRef:*** The externalRef interceptor resolves external references using the ExternalReferenceResolver which is configured on the package.

11. ***execAndWait:*** Used to send an intermediate waiting page for the result.
12. ***exception:*** Maps exception to a result.
13. ***fileUpload:*** This interceptor provides support to file upload in struts 2.
14. ***i18n:*** Remembers the locale selected for a user's session and It provides support to internationalization and localization.
15. ***jsonValidation:*** Used for supporting asynchronous validation.
16. ***logger:*** It outputs the name of the action.
17. ***store:*** It Store and retrieve action messages / errors / field errors for action that implements ValidationAware interface.
18. ***modelDriven:*** Used to makes other model object, which pushes the getModel Result onto the Value Stack.
19. ***scopedModelDriven:*** However, it is very similar to ModelDriven but works for action that implements ScopedModelDriven.
20. ***params:*** It is used to set the request parameters into the Action.
21. ***actionMappingParams:*** This interceptor performs an important work of seting all parameters from the action mapping on the value stack for a request.
22. ***prepare:*** If the Action implements Preparable, then it performs preparation logic.
23. ***profiling:*** Supports action profiling.
24. ***roles:*** It supports action based on role.
25. ***scope:*** Used for storing Action state in the session or application scope.
26. ***servletConfig:*** Provide access to Maps representing HttpServletRequest and HttpServletResponse.
27. ***sessionAutowiring:*** It is already configured as default interceptor.
28. ***staticParams:*** Used to map the static properties to action properties.
29. ***timer:*** Outputs the time required to execute an action.
30. ***token:*** Prevents duplicate request form submission
31. ***tokenSession:*** It prevents duplication submission of request, but stores the submitted data in session when handed an invalid token.
32. ***validation:*** Provides support to input validation.
33. ***workflow:*** Invokes the validate method of action class and if Action errors are created then it returns the INPUT view.
34. ***annotationWorkflow:*** This interceptor is responsible for invoking any annotated methods on the action class which is being executed. It supports specially @Before, @BeforeResult and @After annotations.
35. ***multiselect:*** The org.apache.struts2.interceptor.MultiselectInterceptor is configured as default interceptor. The responsibility of this interceptor is to check each form parameter submitted to the action. If it finds one with a prefix of __multiselect_ it inserts a value for a parameter whose name is derived from the suffix to __multiselect_ if it does not exist. The default value inserted is an empty String array.