Struts2 Notes

topics:

---

1. Intro,architecure of Struts2

2. Hello world

3. Action Interface, ActionSupport

4.Aware Interfaces

5.Namespace,Multiple mapping files, Dynamic Method Invocation

6. OGNL, valueStack

7. Control tags

8. UI tags

9. Interceptors

10. validation framework

11. Struts 2 Type Conversion

12. Internationalization (i18n) support

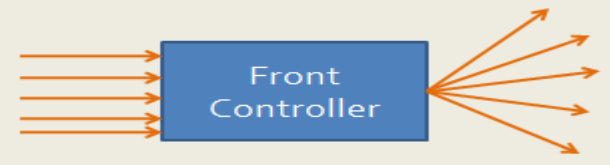
1. Intro,architecure of Struts2

What is Struts2?

Elegant, extensible MVC based framework for creating enterprise-ready Java web applications.

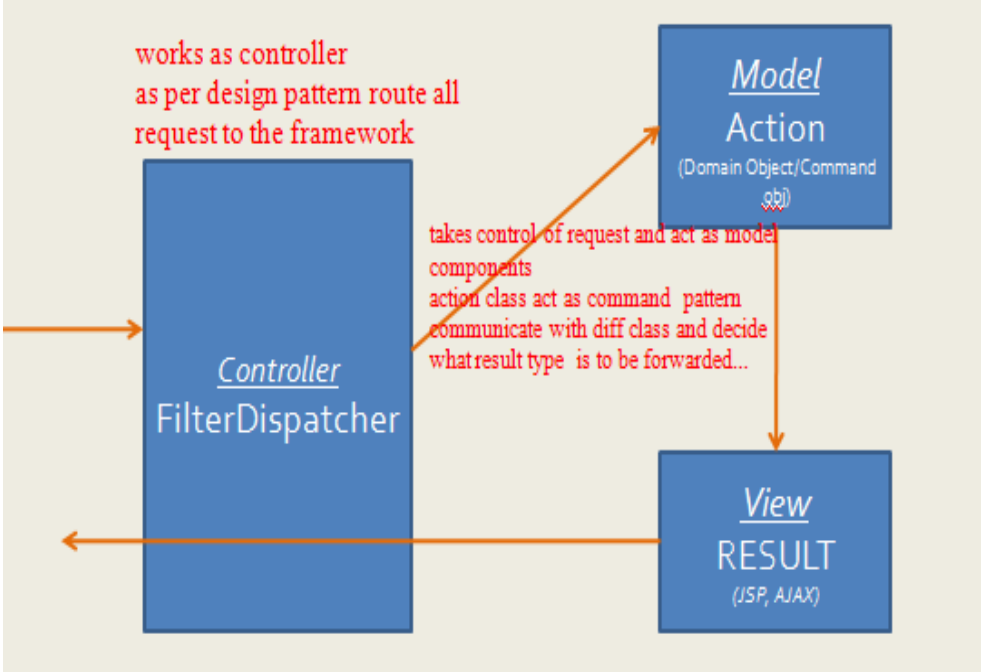
Design pattern used by Struts2 ?

- Front Controller pattern
  - is a component looks for all the request for specific url pattern and routes them into the framework for further processing...

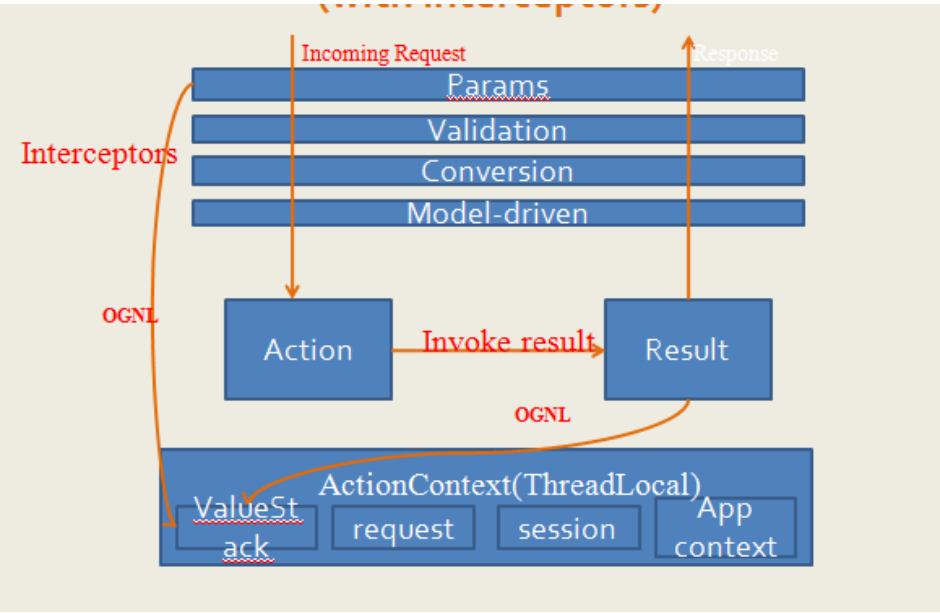


- Composite Pattern
  - struts tiles
- Command Pattern
  - comm. with diff components
  - Ex Action classes
- Decorator Pattern
  - view solution like freemarker etc

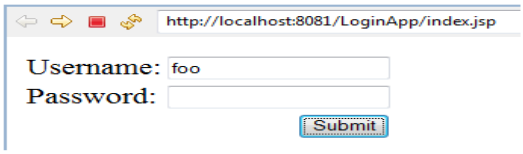
STRUTS 2 BASIC ARCHITECTURE

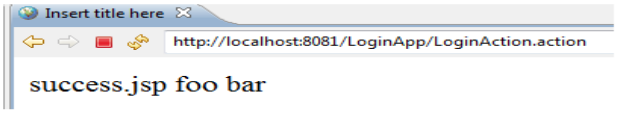


Struts 2 Architecture(with Interceptors)



2. Hello world struts2





steps:

1. create an dynamic web project in eclipse and put jar in lib and set classpath

2. set filter in web.xml

org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter

3. create a hello world Action LoginAction

```
public class LoginAction
 {

        private String name;
        private String passwords;
        ...
        ...


        public String execute() {

                return SUCCESS;

        }
  }
```

4. create an struts.xml in src and map action to it

```
<?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>
</struts>
<constant name="struts.enable.DynamicMethodInvocation" value="false" />
   <constant name="struts.devMode" value="false" />
   <package name="default" namespace="/" extends="struts-default">

                <action name="LoginAction" class="com.actions.LoginAction" method="execute">
                        <result name="success">pages/success.jsp</result>
                        <result name="error">pages/failure.jsp</result>
                </action>
   </package>
```

5. create suitable views

index.jsp

```
<%@ taglib prefix="s" uri="/struts-tags"%>
<s:form action=" LoginAction.action">
                <s:textfield name="name" label="Username" />
                <s:password name="password" label="Password" />
                <s:submit />
</s:form>
```

success.jsp

```
        <%@ taglib prefix="s" uri="/struts-tags"%>
        <s:property value="name"/>
```

3. Action Interface, ActionSupport

Action interface
Action interface define some useful constants

What it contain.....

```
public interface Action {
   public static final String SUCCESS = "success";
   public static final String NONE = "none";
   public static final String ERROR = "error";
   public static final String INPUT = "input";
   public static final String LOGIN = "login";
   public String execute() throws Exception;
}
```

static final String SUCCESS

Indicates successful execution and that
means the result view is shown to the end user.

static final String ERROR

Indicates that there was a failure.
Show an error view, possibly asking the user to retry entering data

static final String INPUT

This is used for a form action
indicating that inputs are. The form associated
with the handler should be shown to the end user.

This result is also used if the given input params are invalid,
meaning the user should try providing input again.

static final String LOGIN

Indicates that the user was not logged in.
The login view should be shown.

static final String NONE

Indicates successful execution but no action is taken.
Useful for actions which wants to redirect etc.

ActionSupport

ActionSupport class provides default implementaion for various services required
by common actions classes...
class ActionSupport implements Validateable,
ValidationAware,LocaleProvider,TextProvider,ValidationAware,Action,Serilizable{

}

<<Validateable>>:

provide validate() method
that allows our action is to be validate
validate() called before execute() method

<<LocaleProvider>>

getLocale() method to provide locate to
be used for localized methods

<<ValidationAware>>

provides methods for saving/retrieving errors messages
eX:
void addActionError(String message);
void addFieldError(String fieldName, String message);

<<TextProvider>>

provides methods to access to resoure bundles
Ex:
String getText(String key, String val);

<<Serializable>>

marker interface.....

Ex:

Use of ActionSupport class for <<Validateable>>and <<TextProvider>>
and property file

Write login application with validation and ApplicationResources.properties file

Login.jsp

Note that key= "….." will pick values form .property file….

```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>

<body>

<s:actionerror />
<s:form action="login.action" method="post">
    <s:textfield name="username" key="label.username" size="20" />
    <s:password name="password" key="label.password" size="20" />
    <s:submit method="execute" key="label.login" align="center" />
</s:form>
</body>
</html>
```

Have validate() in action class……..

```
@Override
public void validate(){
    if((username==null)||(username.length()==0))
    {
        addFieldError("username", getText("username.blank"));
    }

    if((password==null)||(password.length()==0))
    {
        addFieldError("username", getText("password.blank"));
    }
}
```

Have property file

```
label.username=Username
label.password=Password
label.login=Login
error.login=Invalid Username/Password. Please try again.
username.blank=Enter user name
password.blank=Enter user password
```

Mapping for "input" in struts.xml

```
<package name="default" extends="struts-default" namespace="/">
    <action name="login" method="execute"
        class="com.actions.LoginAction">
        <result name="success">Welcome.jsp</result>
        <result name="error">Login.jsp</result>
        <result name="input">Login.jsp</result>
    </action>
</package>
```

Order of execution of action is as follows:

1. if action implements validateable interface , action validate() method is going to execute before execute() method
2. it return "input" if validation fail.

ActionContext

ActionContext can be define as container, which contain objects that require Action for its execution

We can use ActionContext to get object like request, response,session,parameter etc

```
public String execute() {

    ActionContext ctx=ActionContext.getContext();
    HttpServletRequest req=(HttpServletRequest) ctx.get(ServletActionContext.HTTP_REQUEST);
    req.setAttribute("name", username);

    HttpSession session=(HttpSession) ctx.get(ServletActionContext.SESSION);
    session.setAttribute("name", username);
```

Although we have better technique to get session, reqest etc that we are going to discuss next topic.

4. Aware Interfaces

AKA Dependancy Injection in Struts2

When we want HTTP specific object in action, we can use aware interface to inject dependancies….

<<ApplicationAware>>

public void setApplication(Map app);

<<SessionAware>>

public void setSession(Map session);

<<ParameterAware>>

public void setParameter(Map param);

<<ServletResponseAware>>

public void setervletResponseAware(HttpServletResponse res);

<<ServletRequestAware>>

public void setervletResponseAware(HttpServletRequest res)

Ex:

Setting something in session scope:

```
public class InjectSession extends ActionSupport implements SessionAware{

    Map<String,Object> session;
    @Override
    public void setSession(Map<String,Object> session) {
        this.session=session;
    }
    public String execute(){
        User user=new User();
        user.setName("raj");
        user.setPassword("pass");
        session.put("user",user);
        return SUCCESS;
    }

}
```

Now getting in jsp:

```
<s:property value="#session.user.name"/>
```

Simlirly.....

<s:property value="#session.user"/>
<s:property value="#session['user']"/>

<s:property value="#application.user"/>
<s:property value="#parameters.user"/>

More about Struts 2 Actions classes..

primary job of actions
---

1. action act as a data carrier (DTO)

2. action also working as controller
(mini controller)

u should not write bussiness logic in action we should call
treate action class as an mini controller

How action pojo works
First, the action plays an important role in the transfer of
data from the request through to the view, whether its a JSP or
other type of result.

Second, the action assist the framework in determining
which result should render the view that will be returned in the
response to the request.

Condition to be an action

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.

ActionSupport

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

```
class ActionSupport  implemts Action ....
{
}
```

5.Namespace,Multiple mapping files, Dynamic Method Invocation

Namespace

Note that package tag(struts.xml) has the following attributes:

Attribute	Description
name (required)	The unique identifier for the package
extends	Which package does this package extend from? By default, we use struts-default as the base package.
abstract	If marked true, the package is not available for end user consumption.
namesapce	Unique namespace for the actions

Why namespace

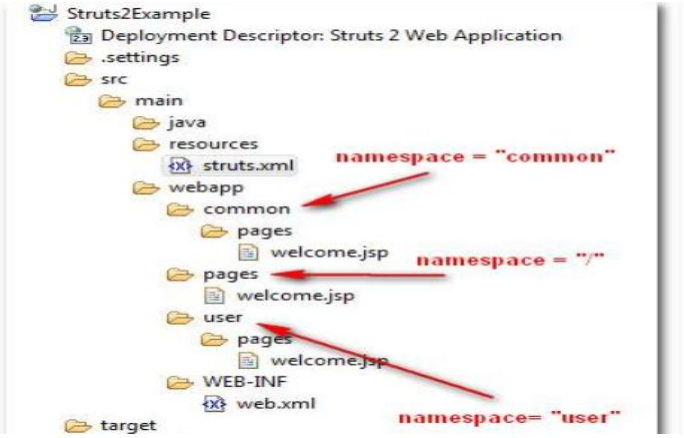
Namespace is a concept to handle the multiple modules by given a namespace to each module.

In addition, it can used to avoid conflicts between
same action names located at different modules

The package "name" will not affect the result, just give a meaningful name.

```
<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.



Mapping how it works?

Example 1

URL : *http://localhost:8080/Struts2Example/SayWelcome.action*

Will match the root namespace.

```
<package name="default" namespace="/" extends="struts-default">
	<action name="SayWelcome">
		<result>pages/welcome.jsp</result>
	</action>
</package>
```

And display the content of **webapp/pages/welcome.jsp**.

Example 2

URL : *http://localhost:8080/Struts2Example/common/SayWelcome.action*

Will match the common namespace.

```
<package name="common" namespace="/common" extends="struts-default">
	<action name="SayWelcome">
		<result>pages/welcome.jsp</result>
	</action>
</package>
```

Dynamic Method Invocation

- It help us to avoid configuring a separate action mapping for each method in the Action class by using the wildcard method
- AKA short cut can create problems

Ex:

The word that matches for the first asterisk will be substituted for the method attribute. So when the request URL is "*addUser*" the *add()* method in the UserAction class will be invoked.

```
<struts>
    <package name="default" extends="struts-default">
        <action name="*User" method="{1}" class="com.actions.UserAction">
            <result name="success">/success.jsp</result>
        </action>
    </package>
</struts>
```

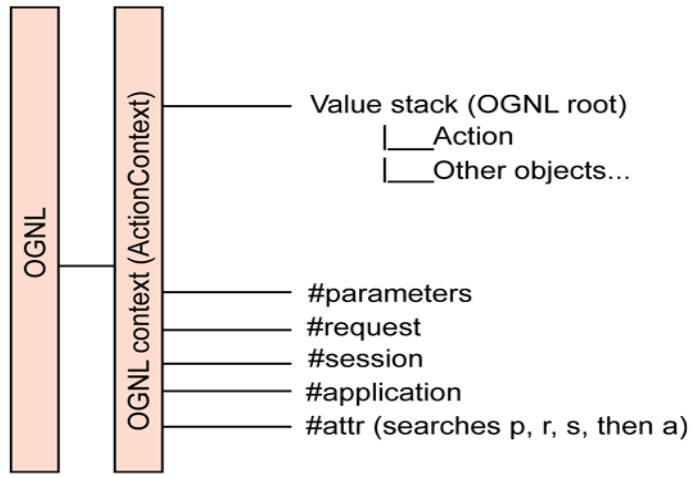
```
<s:form action="User" >
<s:submit />
<s:submit action="addUser" value="Add" />
<s:submit action="updateUser" value="Update" />
<s:submit action="deleteUser" value="Delete" />
</s:form>
```

```
public class UserAction extends ActionSupport

    private String message;

    public String execute()
    {
        message = "Inside execute method";
        return SUCCESS;
    }
    public String add()
    {
        message = "Inside add method";
        return SUCCESS;
    }
    public String update()
    {
        message = "Inside update method";
        return SUCCESS;
    }
    public String delete()
    {
        message = "Inside delete method";
        return SUCCESS;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
```

6. OGNL, valueStack

- The automation of **data transfer and type conversion** is one of the most powerful features of Struts 2. With the help of OGNL, the Struts 2 framework allows transfer of data onto more complex Java-side types like List, Map, etc.
- OGNL is the interface between the Struts 2 framework string-based HTTP Input and Output and the Java-based internal processing
- OGNL is a powerful expression language that is used to reference and manipulate data on the ValueStack.
- OGNL also helps in data transfer and type conversion.
- The OGNL is very similar to the JSP Expression Language.
- OGNL is based on the idea of having a root or default object within the context.

Figure 2. OGNL stack



Ex:

7. Generic tags

Struts2 tags are divided into

1. Generic tags
   Used for controlling flow of data
   And for data extraction from the value stack.

   There are two type of generic tags
   - Control tags
   - Data tags
2. UI tags
   Convern about form creation etc.

Control tags

if

Ex:

```
<s:if test="%{true}">
    this line will be displayed.
</s:if>

<s:if test="%{false}">
    this line will be displayed.
</s:if>
```

else

```
<s:if test="type=="manager">
    your are an manager
</s:if>

<s:else>
    not an manager
</s:if>
```

iterator

aka for loop to iterate for collection array etc

Ex:

```
<s:iterator status="stat" value="{11,22,33,44,55,66}">
    <s:property value="#stat.index"/>
    <s:property value="top"/>
    <s:if test="#stat.last==false">,</s:if>
</s:iterator>
```

append

used to append collection objects to an single collection

```
<s:append id="myAppender">
        <s:param value="%{fruits}"/>
        <s:param value="%{books}"/>
        <s:param value="%{colors}"/>
</s:append>
```

Now:

```
<s:iterator value="%{#myAppender}">

</s:iterator>
```

merge
sort
subset
generator
elseIf

Example:

Setting values in an action

```
public String execute(){
    fruits=new ArrayList<String>();
    cities=new ArrayList<String>();
    colors=new ArrayList<String>();
    fruits.add("Apple");
    fruits.add("Mango");
    fruits.add("Orange");

    cities.add("Delhi");
    cities.add("Mumbai");
    cities.add("Pune");

    colors.add("Red");
    colors.add("Green");
    colors.add("Blue");

    return SUCCESS;
}
```

How to display in an view:

```
<s:append id="appendedItr">
    <s:param   value="%{fruits}"/>
    <s:param   value="%{cities}"/>
    <s:param   value="%{colors}"/>
</s:append>
<s:iterator value="%{#appendedItr}">
    <s:property />,
</s:iterator>
<br><br><b>Merging Iterators:</b><br>
<s:merge id="mergedItr">
    <s:param   value="%{fruits}"/>
    <s:param   value="%{cities}"/>
    <s:param   value="%{colors}"/>
</s:merge>
<s:iterator value="%{#mergedItr}">
    <s:property />,
</s:iterator>
```

Data tags

data tags used for creating and manipulating data
helps to access data from value stack or help to put data to value stack

a

simpiler to <a href..../>

Ex:
<s:url id="url" action="addAction"></s:url>

<s:a href="%{url}">adding</s:a>

action

Used to call actions direcly from jsp

Ex: consider following in struts.xml

<action name="regForm" class="com.RegistrationAction">
<result name="success">reg.jsp</result>
</action>

Now:
in an jsp....

<s:action name="regForm" executeResult="true"/>

by default it is false

date

<s:date name="new java.util.Date()" format="dd/mmm/yyyy"/>
<s:date name="new java.util.Date()" format="%{getText('app.date.format')}"/>

include

<s:include value="header.jsp"/>

param

push

used to push the value on value stack

id : used for referencing element

value: specify value to be pushed to value stack

make accessing data simple...use if you have to
use that data object extensively....

Example :

Consider below example , how use of push make easy to access session scoped varaibles….

```
<s:set name="user" value="#session['user']"/>

<s:push value="#user"/>
        <s:property value="userName'/>

        ....
        ....
        <s:property value="address"/>
</s:push>
```

bean

set

text

url

property

debug

i18n

calling an action from href

```
<p><a href="<s:url action='hello'/>">Hello World</a></p>
```

mapping of that action

```
<action name="hello" class="org.apache.struts.helloworld.action.HelloWorldAction" method="execute">
  <result name="success">/HelloWorld.jsp</result>
</action>
```

url tag with param

```
<s:url action="hello" var="helloLink">
  <s:param name="userName">Bruce Phillips</s:param>
</s:url>

<p><a href="${helloLink}">Hello Bruce Phillips</a></p>
```

8. UI tags

Form tags
form
checkboxlist
file
token
password
textarea
checkbox
select
radio
head
optiontransferselect
reset
updownselect
label
hidden
doubleselect
combobox
submit
datetimepicker
optgroup
textfield

list is long…lets us do some form processing example and try to cover most.

9. Interceptors

Interceptors are conceptually
the same as servlet filters.

Interceptors allow for crosscutting functionality to be
implemented separately from the action as well as the framework.

AOP ie Aspect oriented programming

is not the replacement of OOP but it is support concept to oops

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.

Many of the features provided in the Struts2 framework are implemented using interceptors

examples include

exception handling,
file uploading,
lifecycle callbacks and
validation etc.

In fact, as Struts2 bases much of its functionality on interceptors,
it is not unlikely to have 7 or 8 interceptors assigned per action.

some imp interceptor in struts2

alias

Allows parameters to have different name aliases across requests.

checkbox

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

conversionError

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

createSession

Automatically creates an HTTP session if one does not already exist

debugging

Provides several different debugging screens to the developer.

execAndWait

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

exception

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

fileUpload

Facilitates easy file uploading.

i18n

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

logger

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

params

Sets the request parameters on the action.

prepare

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

profile

Allows simple profiling information to be logged for
actions.

scope

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

ServletConfig

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

timer

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

token

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

validation

Provides validation support for actions

How to use Interceptors?

```
<interceptor-ref name="params"/>
<interceptor-ref name="timer" />
```

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;
}
```

Steps:Hello World User define interceptor

1. create an intercepter (eg: MyInterceptor) and implement <<Interceptor >>
2. and overrid

```
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;
}
```

3. metion maaping in struts.xml

```
<struts>
  <constant name="struts.devMode" value="true" />
  <package name="helloworld" extends="struts-default">
    <interceptors>
      <interceptor name="myinterceptor"  class="com.interceptors.MyInterceptor" />
    </interceptors>

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

```
  </package>
</struts>
```

```
<struts>

 <constant name="struts.enable.DynamicMethodInvocation" value="false" />
     <constant name="struts.devMode" value="false" />
     <package name="default" namespace="/" extends="struts-default">

        <interceptors>
          <interceptor name="myinterceptor1"  class="com.interceptors.MyInterceptor1" />
           <interceptor name="myinterceptor2"  class="com.interceptors.MyInterceptor2" />

           <interceptor-stack name="custom_stack">
             <interceptor-ref name="myinterceptor1"></interceptor-ref>
             <interceptor-ref name="myinterceptor2"></interceptor-ref>
           </interceptor-stack>
       </interceptors>

         <action name="LoginAction" class="com.actions.LoginAction" method="execute">
                 <interceptor-ref name="basicStack"></interceptor-ref>
                  <interceptor-ref name="custom_stack" />

             <result name="success">pages/success.jsp</result>
             <result name="error">pages/failure.jsp</result>
          </action>
     </package>

</struts>
```

basic stack

```
<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>
```

Applying basic stack

```
<action name="hello" class="com.MyAction">
  <interceptor-ref name="basicStack"/>
  <result>view.jsp</result>
</action>
```

The above registration of "basicStack" will register complete stake of all the six interceptors with hello action.

This should be noted that interceptors are executed in the order, in which they have been configured. For example, in above case, exception will be executed first, second would be servlet-config and so on.

Now let discuss some imp interceptor one by one……..

Chaining interceptor

Used to copy all the objects from the value stack of currently executing Action to next one aligned in action chaining

Action chaining

```
<action name="Action1" class="com.actions.Action1" method="execute">
                        <interceptor-ref name="basicStack"/>
                        <result name="success" type="chain">action2</result>
```

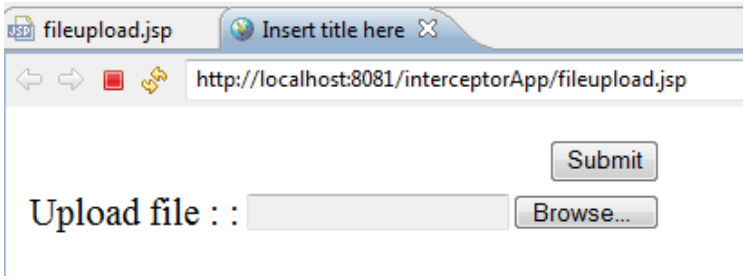
```
                <result name="error">failure.jsp</result>
            </action>
        <action name="action2" class="com.actions.Action2" method="execute">
                <interceptor-ref name="basicStack"/>
                <result name="success">success.jsp</result>
                <result name="error">failure.jsp</result>
            </action>
```

```
<action name="Action1" class="com.actions.Action1" method="execute">
    <interceptor-ref name="basicStack"/>
    <result name="success" type="chain">Action2</result>
    <result name="error">pages/failure.jsp</result>
</action>

<action name="Action2" class="com.actions.Action2" method="execute">
    <interceptor-ref name="basicStack"/>
    <result name="success">pages/resultchain.jsp</result>
    <result name="error">pages/failure.jsp</result>
</action>
```

FileUpload interceptor

When we upload a file using html form action class need to know some description about file to be uploaded

1. File object to handle file
2. Content type of file
3. Name of file



SendRedirect in Struts2

```
<action name="hello"  class="com..HelloWorldAction" method="execute">
      <result name="success" type="redirect">
            <param name="location">/NewWorld.jsp</param >
     </result>
    </action>
```

Logger interceptor

When added to intercetor stack, logs the start and end point fo execution of action or execution of whole whole stack define for that action including all interceptors and action itself

```
<interceptor-ref name="logger"></interceptor-ref>
```

Model driven interceptor

Responsible for looking for model driven actions and add model of the action into the actions value stack making it available in actions

We need to implement two things:

1. Action class must implements ModelDriven interface
2. Model-Driven interceptor must be applied to the action

10. validation framework

2 ways to do validation

1. with the help of ActionSupport
2. XML way: more flexible

with the help of ActionSupport

create a page

```
<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>
```

add following in action class

```
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");
    }
  }
```

dont forget to map for input in struts2

```
<package name="default" namespace="/" extends="struts-default">
                <action name="EmployeeReg" class="com.actions.EmployeeReg" method="execute">
                        <result name="success">success.jsp</result>
                        <result name="input">regform.jsp</result>
                </action>
  </package>
```

When the user presses the submit button, Struts 2 will automatically execute the validate method and if any of the if statements listed inside the method are true, Struts 2 will call its addFieldError method. If any errors have been added then Struts 2 will not proceed to call the execute method. Rather the Struts 2 framework will return input as the result of calling the action.
So when validation fails and Struts 2 returns input, the Struts 2 framework will redisplay the index.jsp file.

Struts - XML Based Validators

if it is the action **EmployeeReg** then name of validation file must be **EmployeeReg-validation.xml**

create validation xml file in '[action-class]'-validation.xml

creaate an reg form

```
<%@taglib prefix="s" uri="/struts-tags" %>
<s:form action="EmployeeReg.action" method="post" validate="true">
    <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>
```

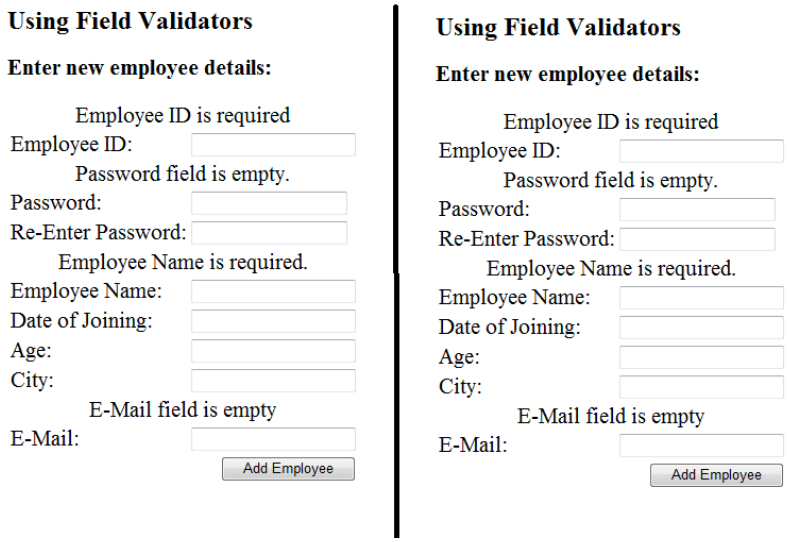
EmployeeReg-validation.xml

```
<!DOCTYPE validators PUBLIC
"-//OpenSymphony Group//XWork Validator 1.0.2//EN"
"http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>
  <field name="name">
    <field-validator type="required">
      <message>
        The name is required.
      </message>
    </field-validator>
  </field>

  <field name="age">
   <field-validator type="int">
      <param name="min">29</param>
      <param name="max">64</param>
      <message>
        Age must be in between 28 and 65
      </message>
    </field-validator>
  </field>
</validators>
```

Client side validation

validate="true"
this option let java script produce at client side……

More Example:



11. Struts 2 Type Conversion

Struts2 privide automatical type conversion for basic data types

such as ....

- Integer, Float, Double, Decimal
- Date and Datetime
- Arrays and Collections
- Enumerations
- Boolean
- BigDecimal

what if we have user define object?

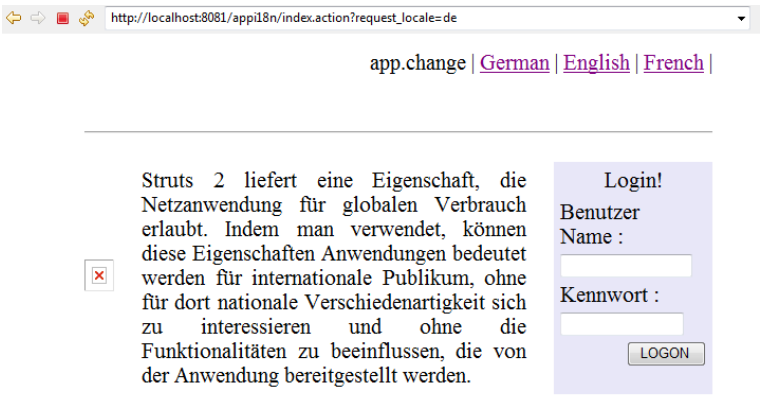
in that cases Struts 2 Type Conversion is very handy.....struts will print object identification number…

Ex:

12. Internationalization (i18n) support

1. resource bundles
2. interceptors and
3. tag libraries

Hello world example



You don't need to worry about writing pages in different languages.
All you have to do is to create a resource bundle for each language that you want.

The resource bundles will contain titles, messages, and other text in the language of your user.

Resource bundles are the file that contains the key/value pairs for the default language of your application.
To develop your application in multiple languages, you would have to maintain multiple property files corresponding to those languages/locale and define all the content in terms of key/value pairs.

For example if you are going to develop your application for US English (Default), Spanish, and Franch the you would have to create three properties files.

global.properties

global.properties: By default English (United States) will be applied
global_fr.properties: This will be used for Franch locale.
global_es.properties: This will be used for Spanish locale.