Struts 2

Rajeev Gupta
M. Tech. CS

Workshop topics

#Introduction & Architecture of Struts2

#Hello world

# Action Interface, ActionSupport

#Aware Interfaces

# Namespace, Multiple mapping files, Dynamic Method Invocation

#OGNL, Value Stack

# Control tags

# Date tags

# UI tags

# Interceptors

# validation framework

# Struts 2 Type Conversion

#Internationalization (i18n) support

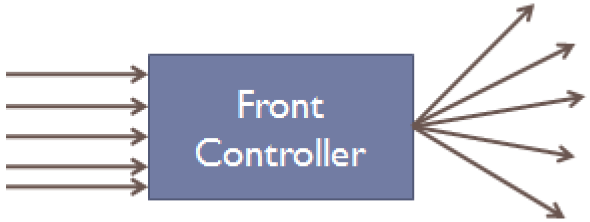
#Introduction & Architecture of Struts2

Struts2

- Struts2 is Elegant, extensible MVC based framework for creating enterprise-ready Java web applications.
  - Pull-MVC framework. i.e. the data that is to be displayed to user has to be pulled from the Action.
  - Supports annotation based configurations
  - Action class in Struts 2 act as the model in the web application
  - Comes with power APIs to configure Interceptors
  - The view part is highly configurable and it supports different result-types such as Velocity, FreeMarker, JSP, etc

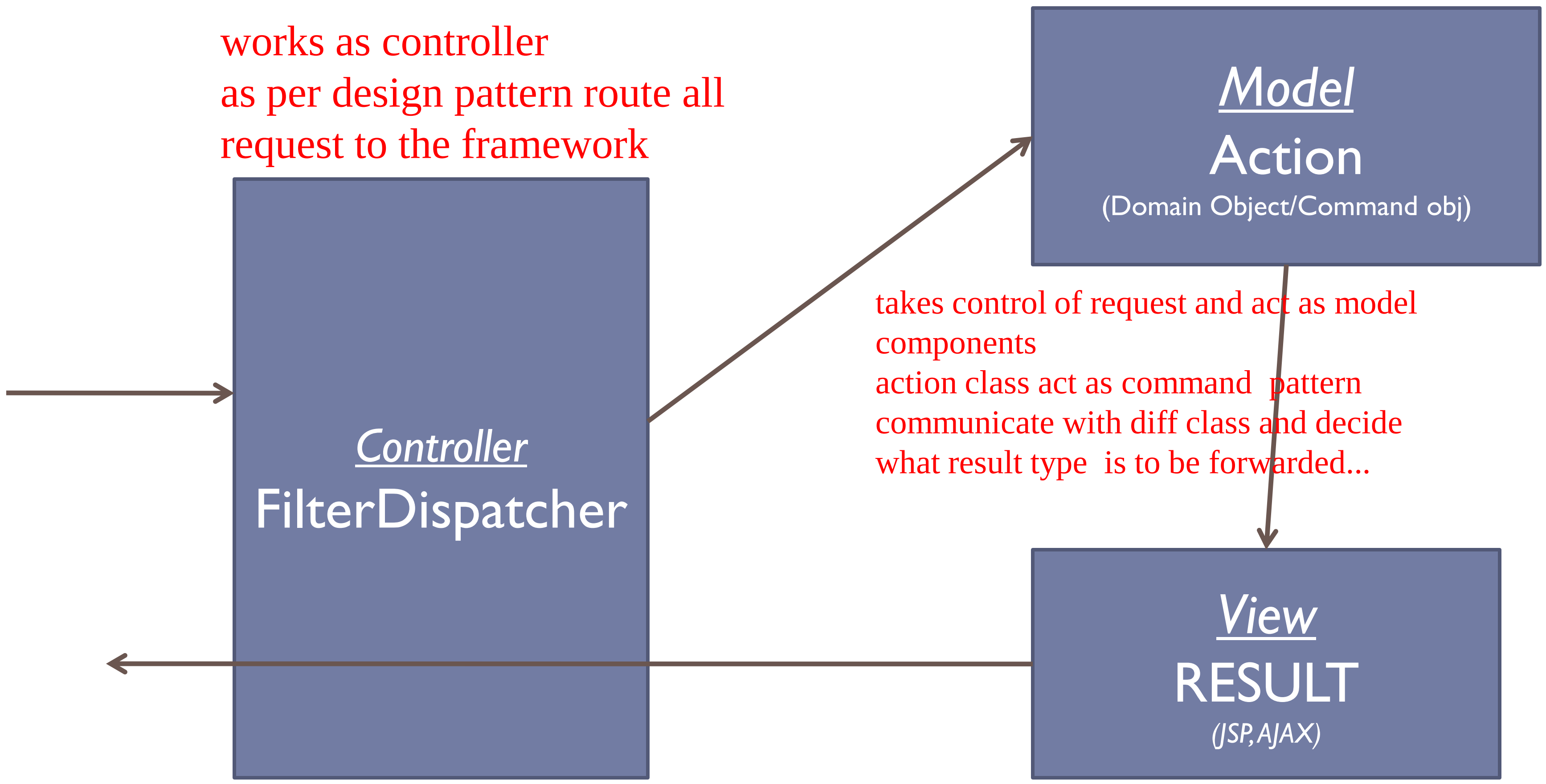
Design Patterns used by struts

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



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

STRUTS 2 BASIC ARCHITECTURE


works as controller
as per design pattern route all
request to the framework
Controller
FilterDispatcher
Model
Action
(Domain Object/Command obj)
takes control of request and act as model
components
action class act as command pattern
communicate with diff class and decide
what result type is to be forwarded...
View
RESULT
(JSP,AJAX)

Struts 1 vs Struts 2 in netshell

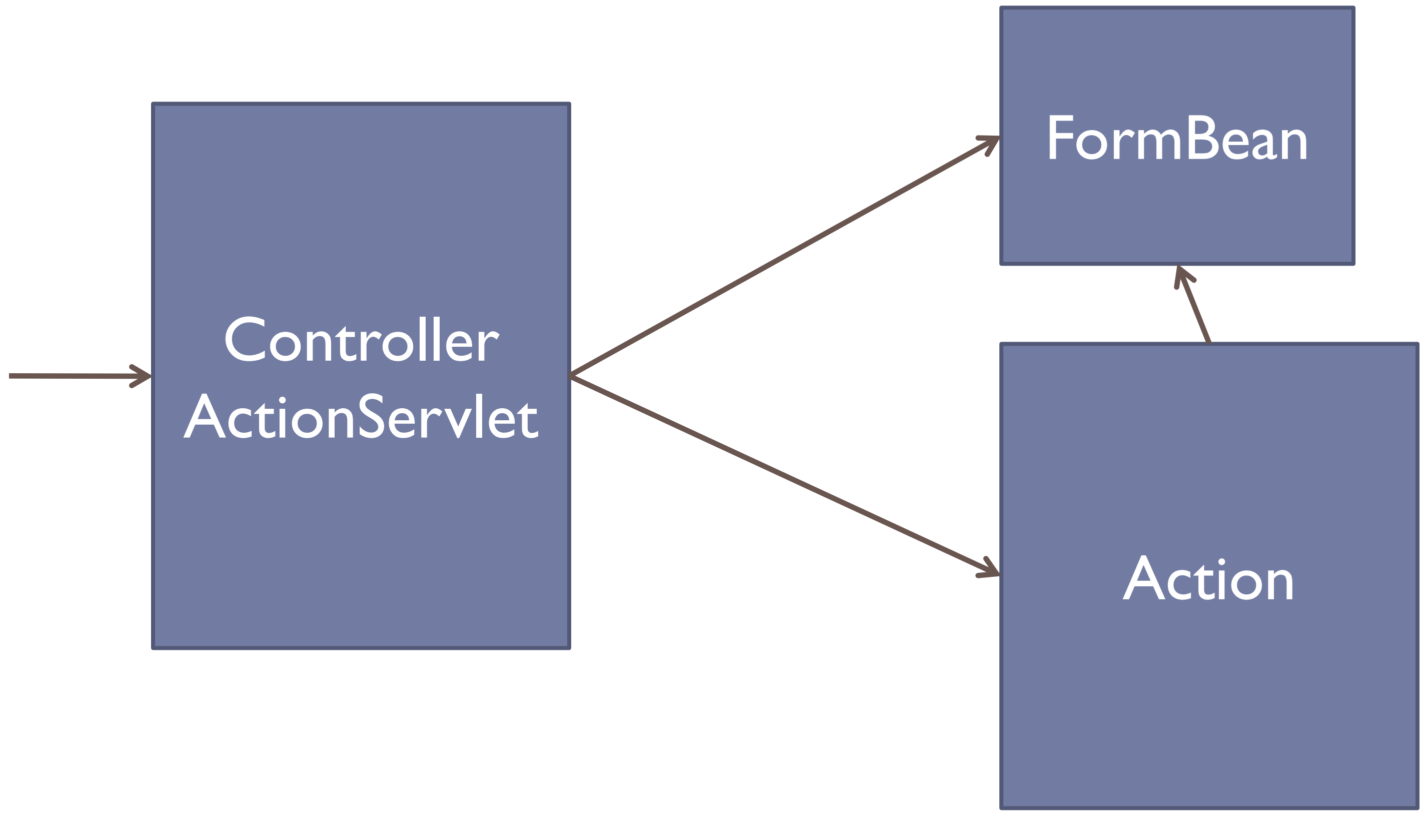
- Struts 1.x
  - In Struts1.x ActionServlet act as a front controller and **for each request new form bean instance is created** to hold parameter that access an thread of Action class
  - only one instance of Action is created

- Struts2.x
  - In Struts2.x FilterDispacher that act as front controller, as request received a new instance of action class is created and param interceptor load all the parameter from the request to the field of action instance

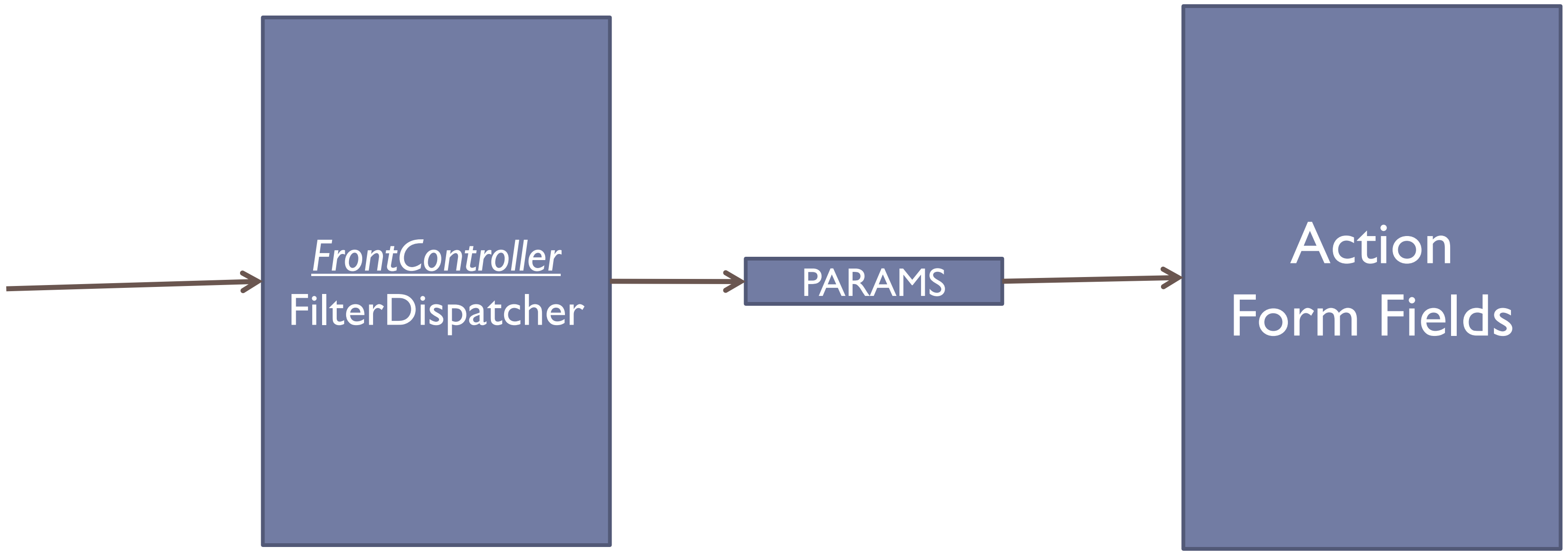
  - No separate form bean to hold the request parameter

  - Create new instance for each and every request ; the action can now hold all the request parameter
  - can be directly mapped to Action (Hence can work as Domain logic...)

Struts 1.x Action



Struts 2 Action

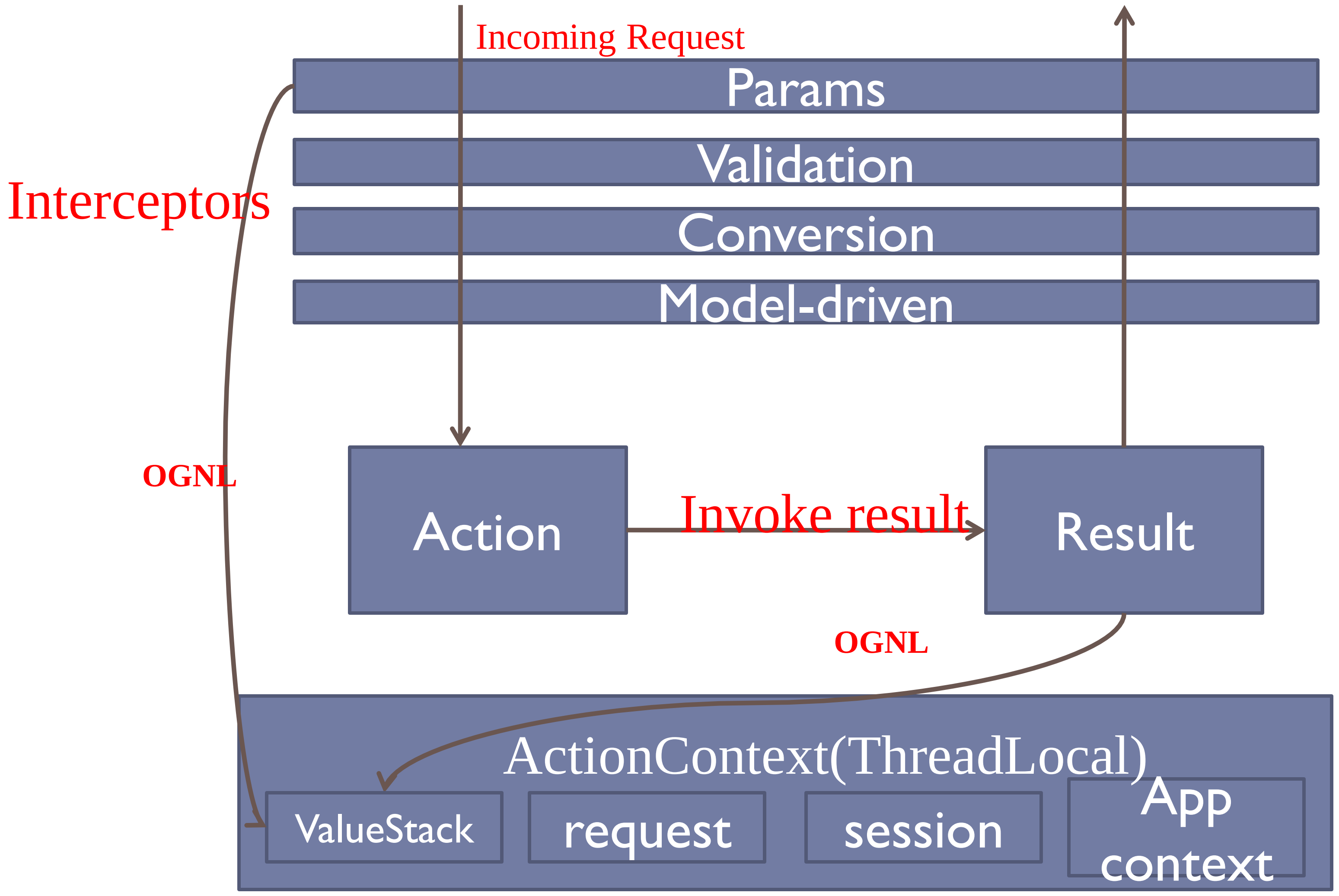


Struts 2 Arch with interceptor

- Request received by struts framework before it can given to an action class. Before action can fire configured interceptor does some pre processing
- Some of interceptor are (detail latter)
  - Param: responsible for transfer all request parameter to the action instances and maintain a copy in values stack...
  - validation: for validation
  - Conversion: type conversion
  - Model driven: for handling form bean
  - etc
- after all interceptor execute then ....

- Action takes responsibilities to execute some business and logic and return the result. and decide which Result to be displayed, then this display page access value stack with OGNL then response is send through interceptor (post processing)

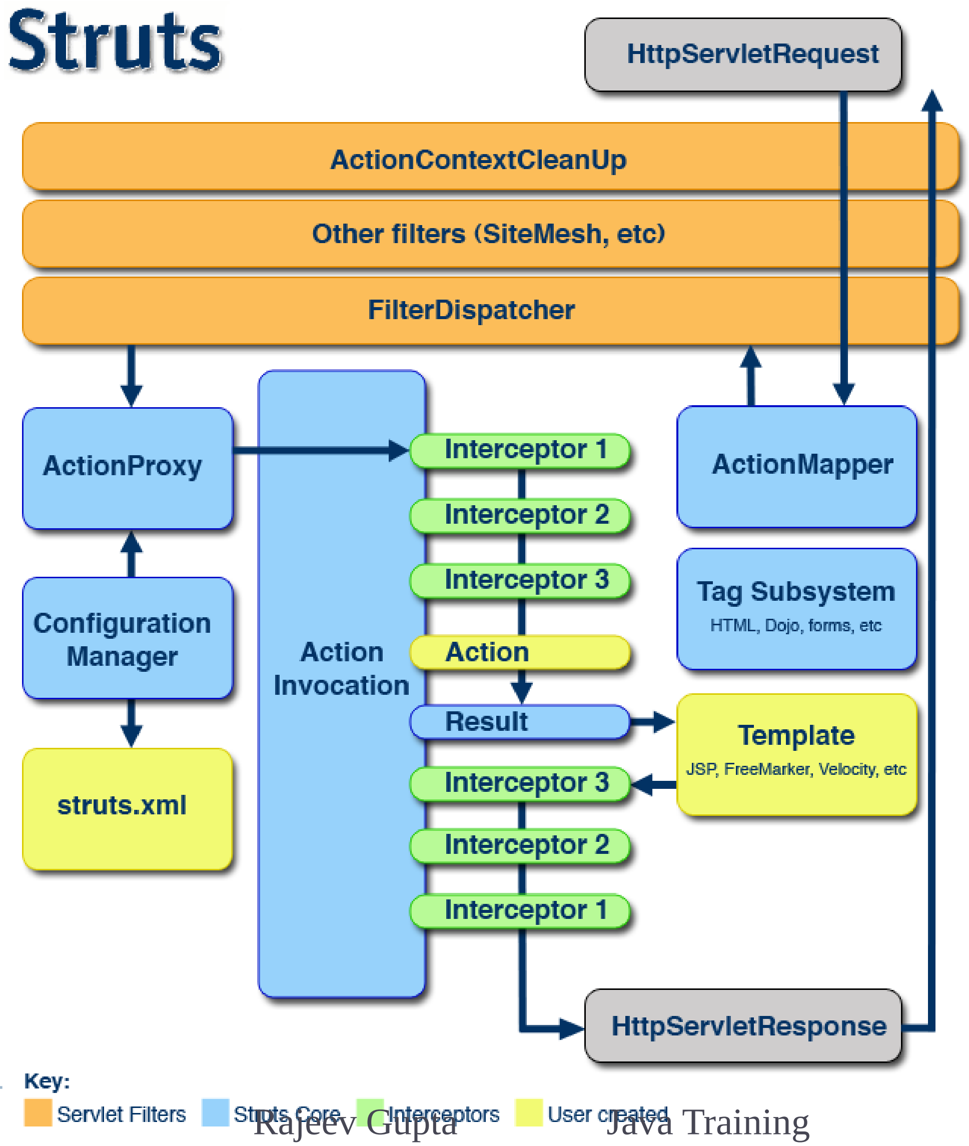
Struts 2 Architecture (with Interceptors)



Advantages of Struts 2

- An Advanced framework with lot of features.
- Based on the MVC Architecture.
- Simple configuration
- Interceptors to reduce the cross cutting functionality
- OGNL
- Pluggable with different Result types like Ajax, JSP, Free Marker, Velocity etc.,

Struts 2 – Behind the scenes



Architecture in details

1. The normal lifecycle of struts begins when the request is sent from client. This request is passed to filter dispatcher by web container.

2. The Filter Dispatcher filter is called which consults the **ActionMapper** to determine whether an **Action** should be invoked.

3. If ActionMapper finds an Action to be invoked, the Filter Dispatcher delegates control to **ActionProxy**.

4. ActionProxy reads the configuration file such as struts.xml. ActionProxy creates an instance of **ActionInvocation** class and delegates the control.

Architecture in details…

5. ActionInvocation is responsible for command pattern implementation. It invokes the Interceptors one by one (if required) and then invoke the Action.

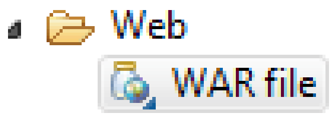
6. Once the Action returns, the **ActionInvocation** is responsible for looking up the proper result associated with the Action result code mapped in struts.xml.

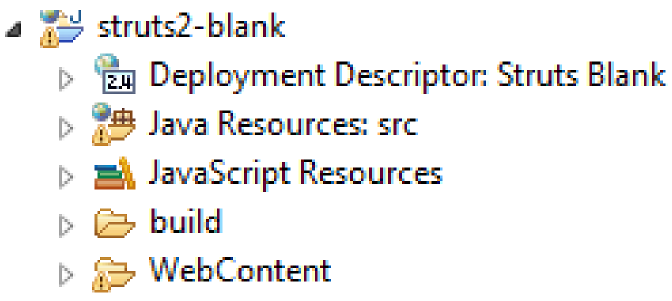
7. The Interceptors are executed again in reverse order and the response is returned to the Filter (In most cases to FilterDispatcher). And the result is then sent to the servlet container which in turns send it back to client.

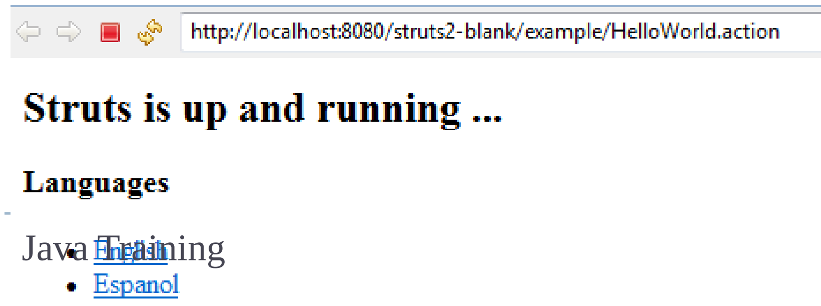
#Hello world

Struts2 Quick

- **Start Downloading and run**
- http://struts.apache.org/download.cgi
- Extract struts 2 download
- Struts come with a blank application called struts2-blank.war.
- In eclipse import this application
  - File→ Import
  - Select Web→War file
  - Browse and select the war file
  - And Finish leaving the rest default entries.
  - Now run this application on the server





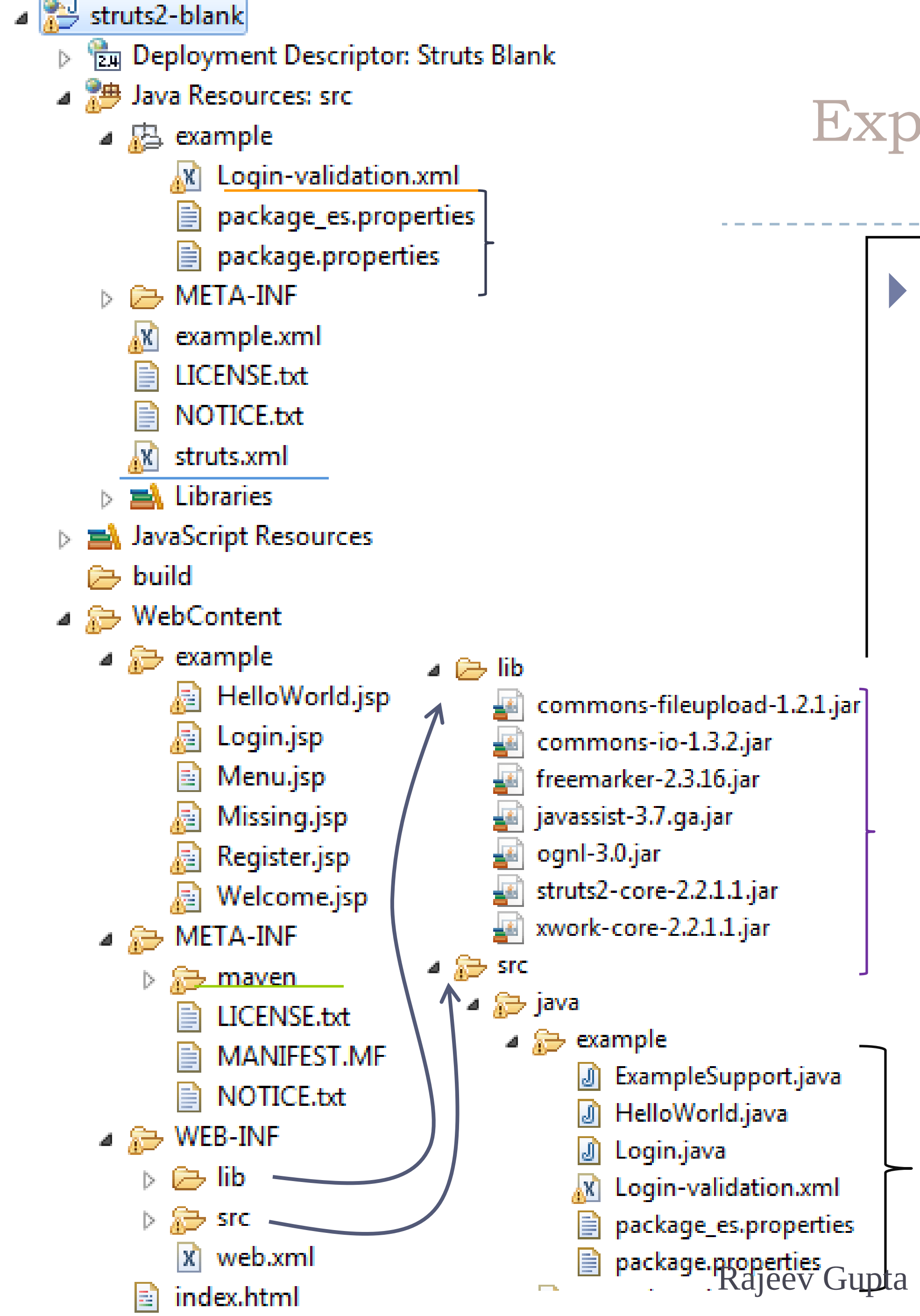

http://localhost:8080/struts2-blank/example/HelloWorld.action

Struts is up and running ...

Languages

- English
- Espanol

Exploring struts2-blank.war



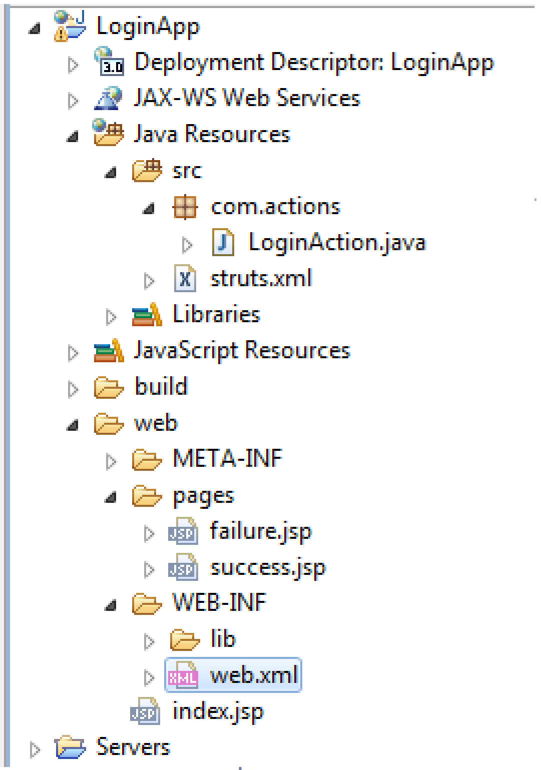
```
◢ struts2-blank
  ▷ Deployment Descriptor: Struts Blank
  ◢ Java Resources: src
    ◢ example
        Login-validation.xml
        package_es.properties
        package.properties
    ▷ META-INF
      example.xml
      LICENSE.txt
      NOTICE.txt
      struts.xml
    ▷ Libraries
  ▷ JavaScript Resources
    build
  ◢ WebContent
    ◢ example
        HelloWorld.jsp
        Login.jsp
        Menu.jsp
        Missing.jsp
        Register.jsp
        Welcome.jsp
    ◢ META-INF
      ▷ maven
        LICENSE.txt
        MANIFEST.MF
        NOTICE.txt
    ◢ WEB-INF
      ▷ lib
      ▷ src
        web.xml
      index.html
```

```
◢ lib
    commons-fileupload-1.2.1.jar
    commons-io-1.3.2.jar
    freemarker-2.3.16.jar
    javassist-3.7.ga.jar
    ognl-3.0.jar
    struts2-core-2.2.1.1.jar
    xwork-core-2.2.1.1.jar
◢ src
  ◢ java
    ◢ example
        ExampleSupport.java
        HelloWorld.java
        Login.java
        Login-validation.xml
        package_es.properties
        package.properties
```

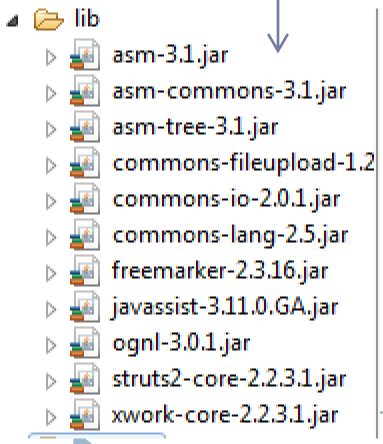
- Some of the important files to notice here are
  - struts.xml
  - Some validation xml file
  - Maven
  - lib files
  - some property files
  - java class files
  - web.xml, jsp

Creating hello world application

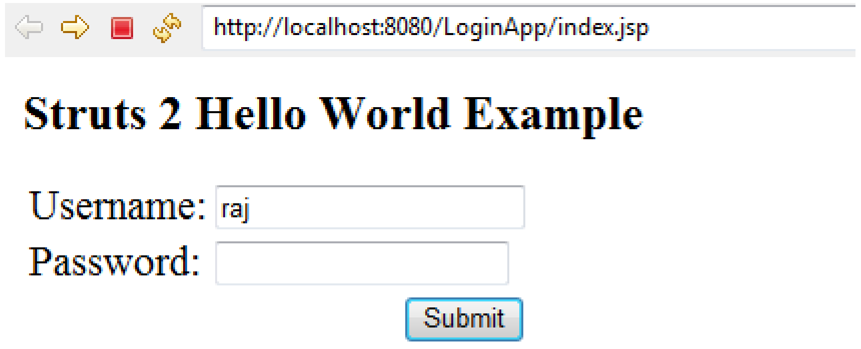
- Steps
  - Create an dynamic web project in eclipse and put jar in lib(from previous project) and set classpath
  - set filter in web.xml org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
  - Create a hello world Action LoginAction
  - Create an struts.xml in src and map action to it
  - Create suitable views for displaying result



```
LoginApp
  Deployment Descriptor: LoginApp
  JAX-WS Web Services
  Java Resources
    src
      com.actions
        LoginAction.java
      struts.xml
    Libraries
  JavaScript Resources
  build
  web
    META-INF
    pages
      failure.jsp
      success.jsp
    WEB-INF
      lib
      web.xml
    index.jsp
Servers
```



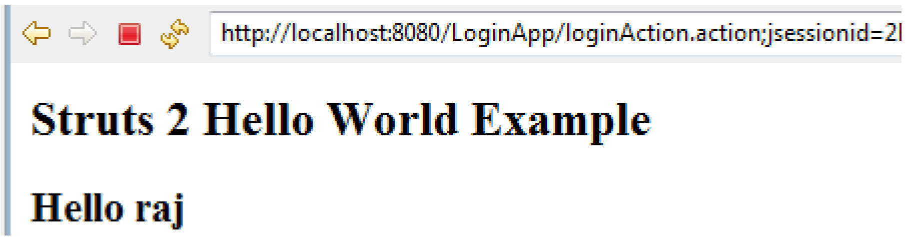
```
lib
  asm-3.1.jar
  asm-commons-3.1.jar
  asm-tree-3.1.jar
  commons-fileupload-1.2
  commons-io-2.0.1.jar
  commons-lang-2.5.jar
  freemarker-2.3.16.jar
  javassist-3.11.0.GA.jar
  ognl-3.0.1.jar
  struts2-core-2.2.3.1.jar
  xwork-core-2.2.3.1.jar
```



http://localhost:8080/LoginApp/index.jsp

Struts 2 Hello World Example

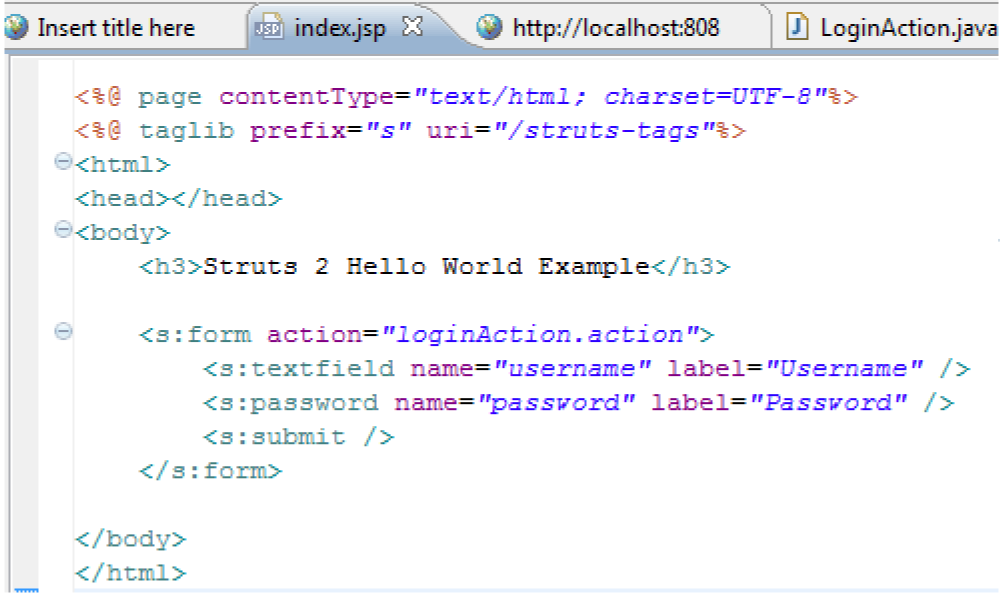
Username:	raj
Password:	



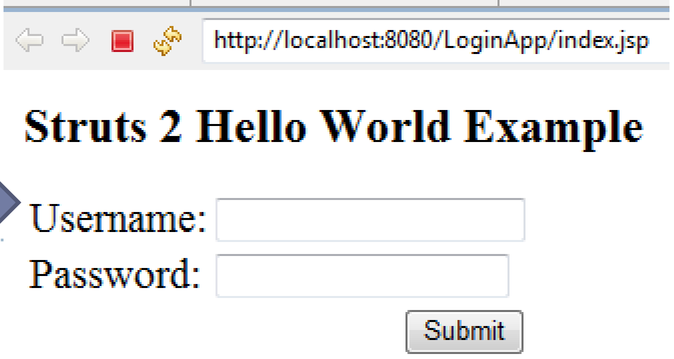
http://localhost:8080/LoginApp/loginAction.action;jsessionid=2I

Struts 2 Hello World Example

Hello raj



```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head></head>
<body>
    <h3>Struts 2 Hello World Example</h3>

    <s:form action="loginAction.action">
        <s:textfield name="username" label="Username" />
        <s:password name="password" label="Password" />
        <s:submit />
    </s:form>

</body>
</html>
```

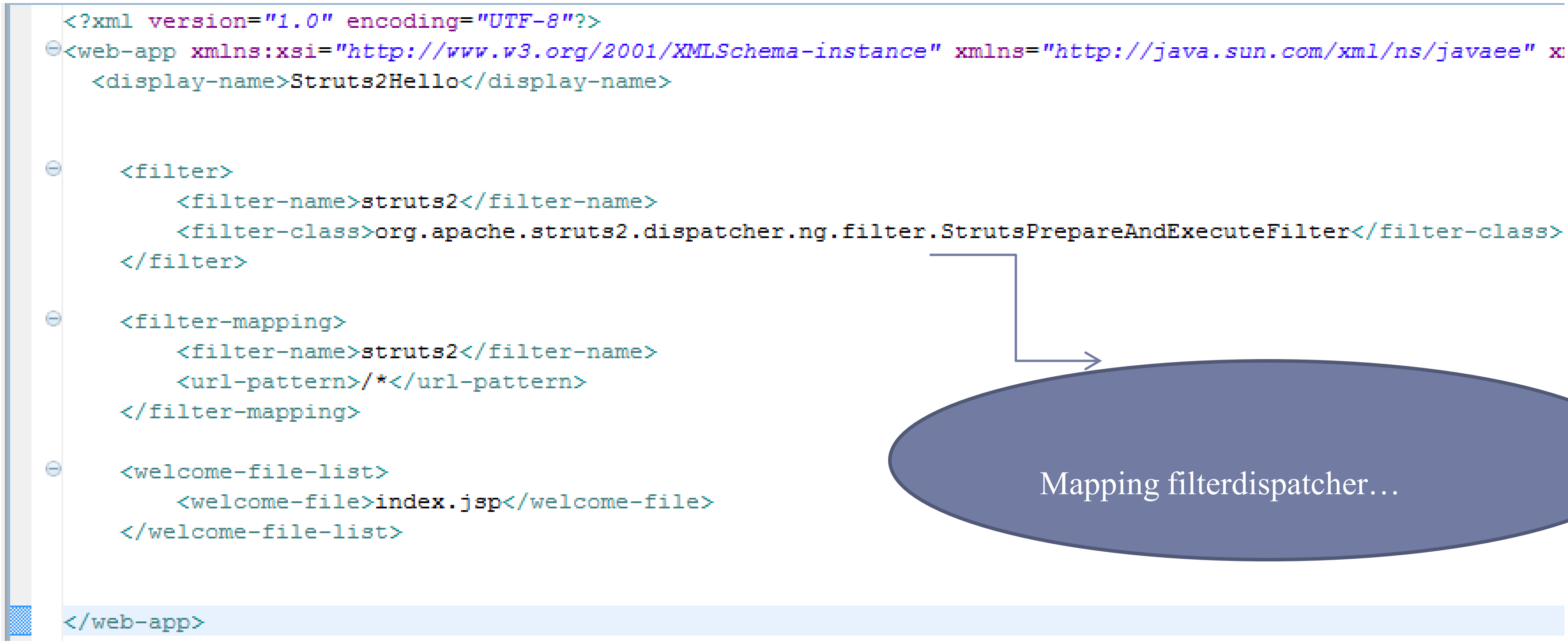


Struts 2 Hello World Example

at run time ……

```
<html>
<head></head>
<body>
      <h3>Struts 2 Hello World Example</h3>

      <form id="loginAction" name="loginAction" action="loginAction.action;jsessionid=2F89A8AACC
method="post">
<table class="wwFormTable">
            <tr>
    <td class="tdLabel"><label for="loginAction_username" class="label">Username:</label></td>
    <td
><input type="text" name="username" value="" id="loginAction_username"/>
</td>
</tr>

            <tr>
    <td class="tdLabel"><label for="loginAction_password" class="label">Password:</label></td>
    <td
><input type="password" name="password" id="loginAction_password"/></td>
</tr>

            <tr>
    <td colspan="2"><div align="right"><input type="submit" id="loginAction_0" value="Submit"/>
</div></td>
</tr>
```

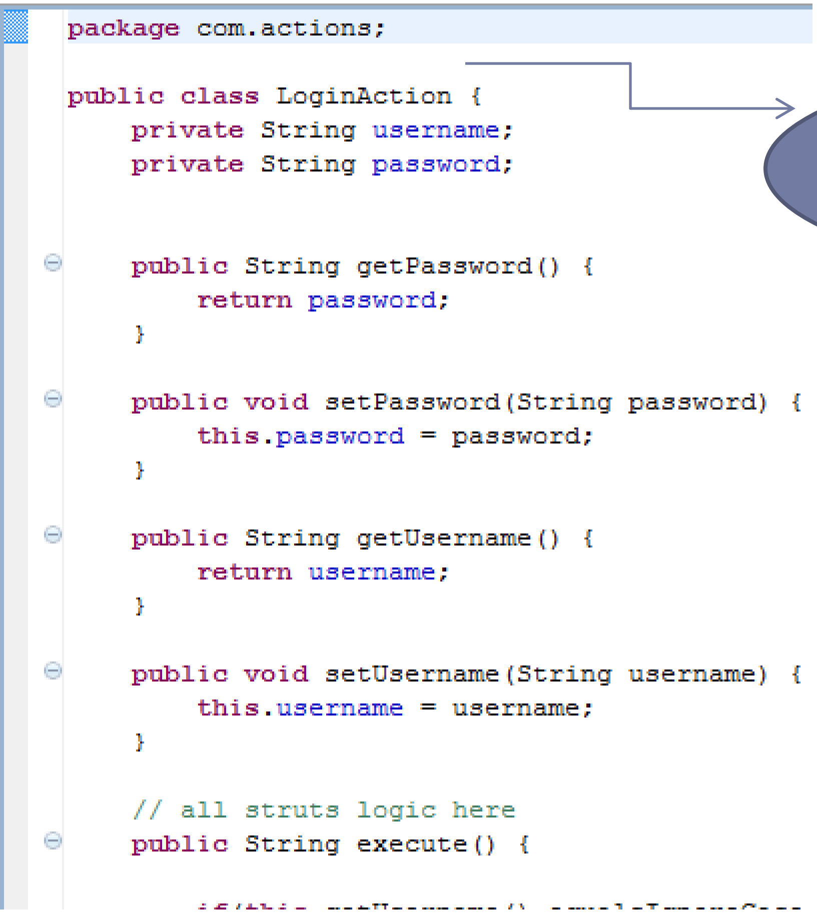
Mapping front controller in web.xml



```
<?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" x
  <display-name>Struts2Hello</display-name>


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

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

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>


</web-app>
```

Mapping filterdispatcher…

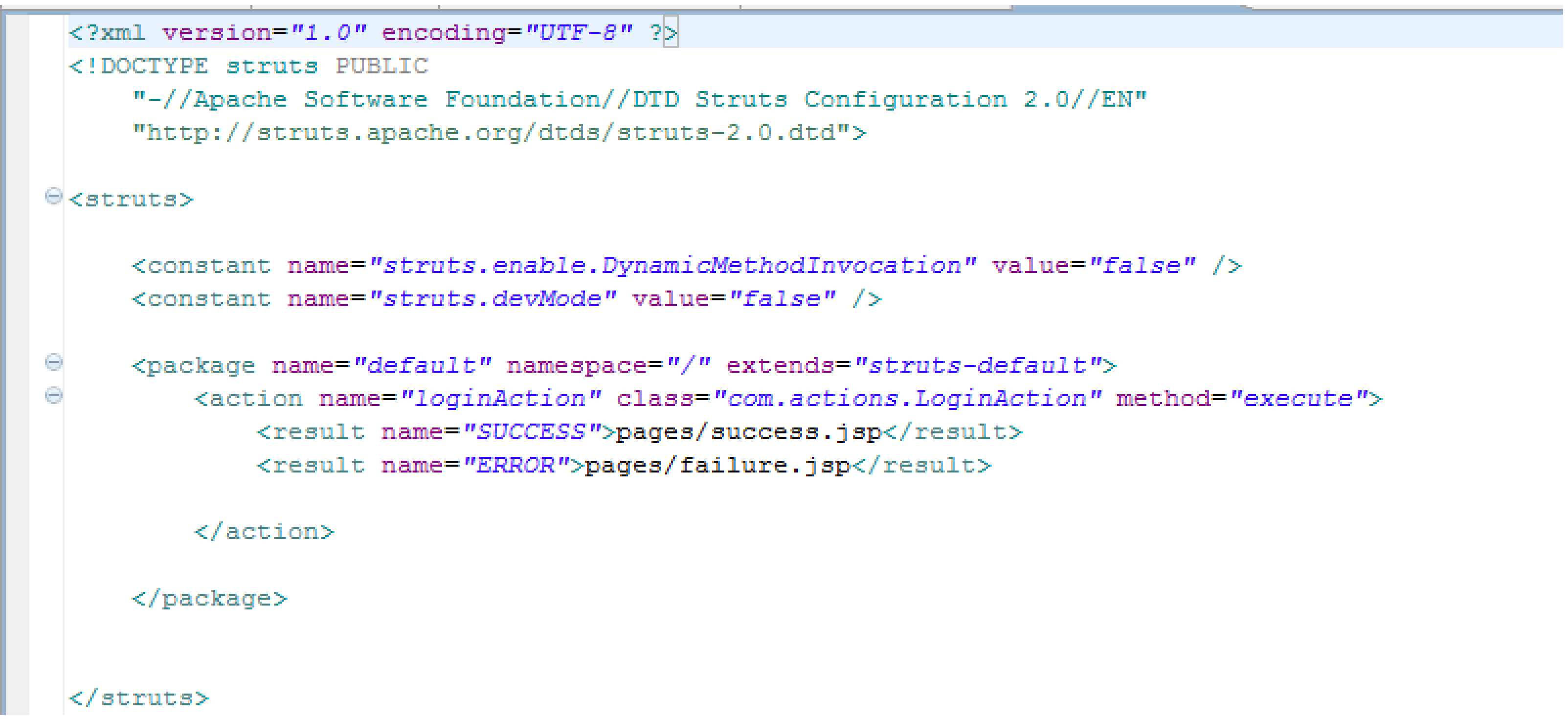
POJO class



```
package com.actions;

public class LoginAction {
    private String username;
    private String password;


    public String getPassword() {
        return password;
    }

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

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    // all struts logic here
    public String execute() {
```

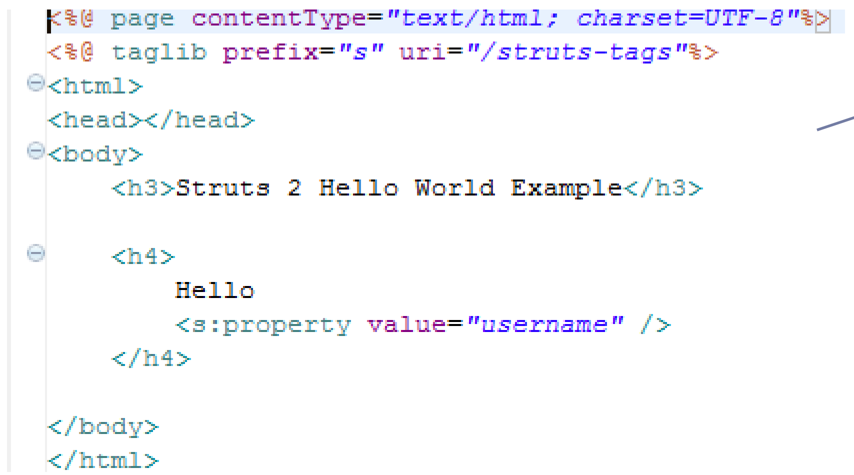
Action analogous to mini servlet

Mapping struts.xml

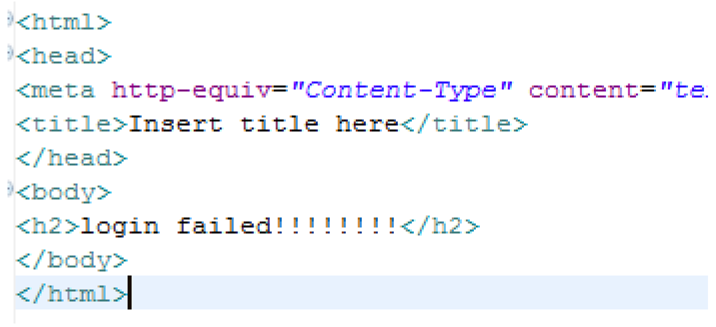


```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

    <constant name="struts.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>


</struts>
```

JSPs to display processing result



```
<%@ page contentType="text/html; charset=UTF-8"%>
<%@ taglib prefix="s" uri="/struts-tags"%>
<html>
<head></head>
<body>
    <h3>Struts 2 Hello World Example</h3>

    <h4>
        Hello
        <s:property value="username" />
    </h4>

</body>
</html>
```



```
<html>
<head>
<meta http-equiv="Content-Type" content="te
<title>Insert title here</title>
</head>
<body>
<h2>login failed!!!!!!!!</h2>
</body>
</html>
```

Success.jsp

Failure.jsp

# Action Interface, ActionSupport

Action Interface

- Action interface define some useful constants, we can used these constant as return from action methods.

```
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;
}
```

Action Interface

- **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{
}
```

ActionSupport

- **<<Validateable>>:**
  - provide validate() method that allows our action is to be validate validate() called before execute() method

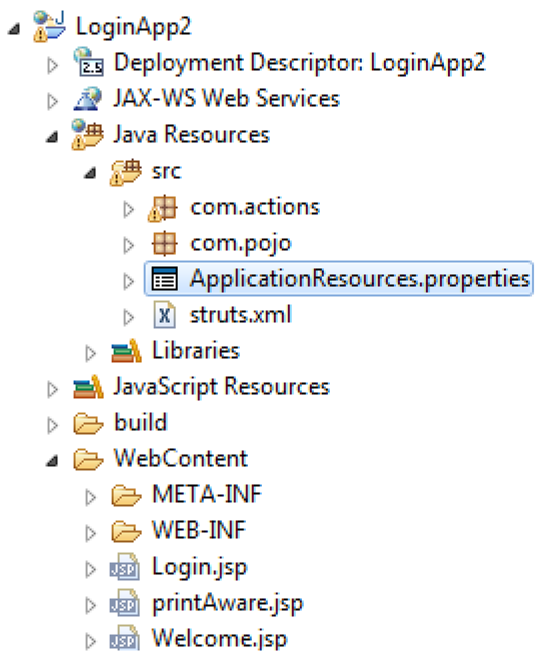
- **<<LocaleProvider>>**
  - getLocale() method to provide locate to be used for localized methods

- **<<ValidationAware>>**
  - provides methods for saving/retrieving errors messages
    - void addActionError(String message);
    - void addFieldError(String fieldName, String message);

ActionSupport

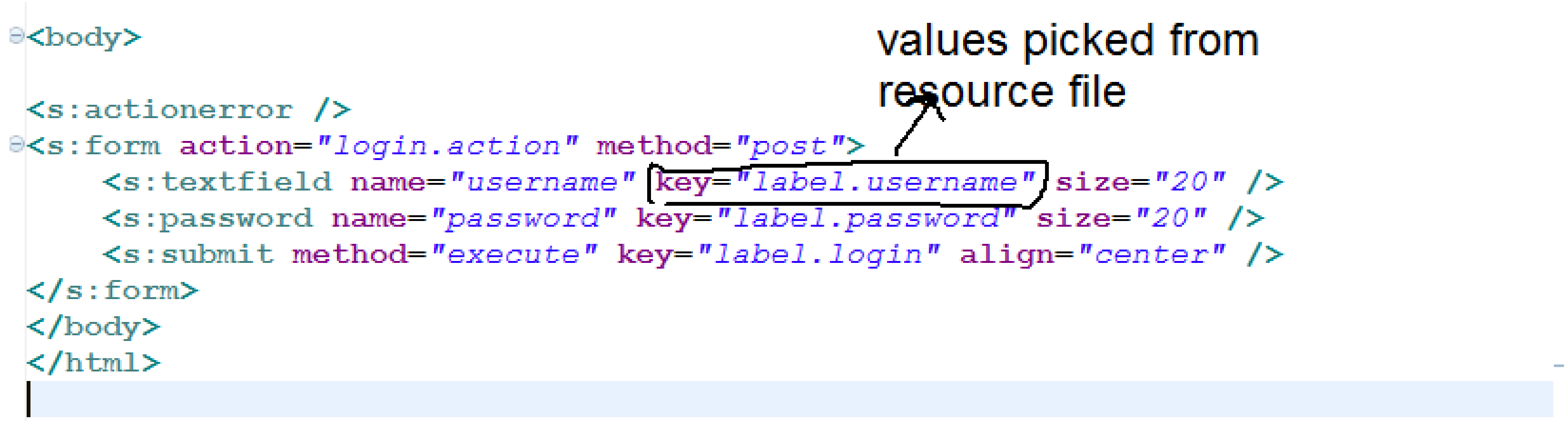
- **<<TextProvider>>**
  - provides methods to access to resoure bundles
    - Ex:
    - String getText(String key, String val);
- **<<Serializable>>**
  - marker interface…….
- Use of ActionSupport class for <<Validateable>>and <<TextProvider>>and property file
- Next we write login application with validation and ApplicationResources.properties file

Modified Login Application


```
LoginApp2
  Deployment Descriptor: LoginApp2
  JAX-WS Web Services
  Java Resources
    src
      com.actions
      com.pojo
      ApplicationResources.properties
      struts.xml
    Libraries
  JavaScript Resources
  build
  WebContent
    META-INF
    WEB-INF
    Login.jsp
    printAware.jsp
    Welcome.jsp
```

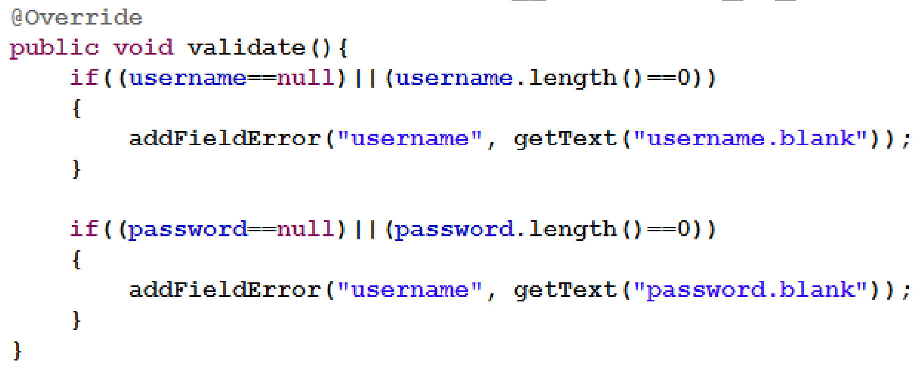
- We write key-value pair in ApplicationResources.properties file
- Note that key= "….." will pick values

  form .property file….
- Loose coupling and internationalization can be provide in this way
-

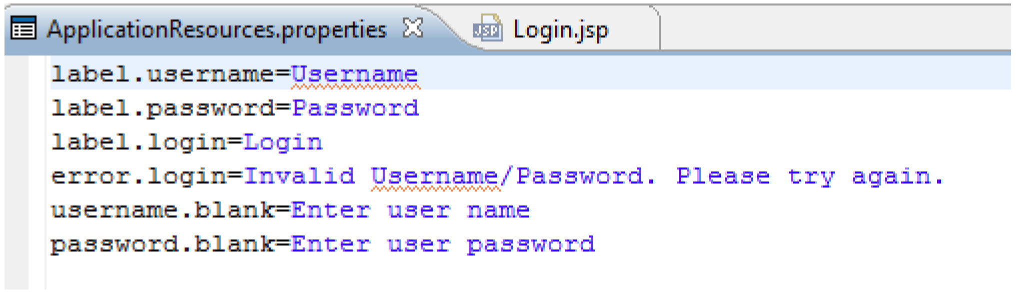

```
<body>

<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>
```

values picked from resource file

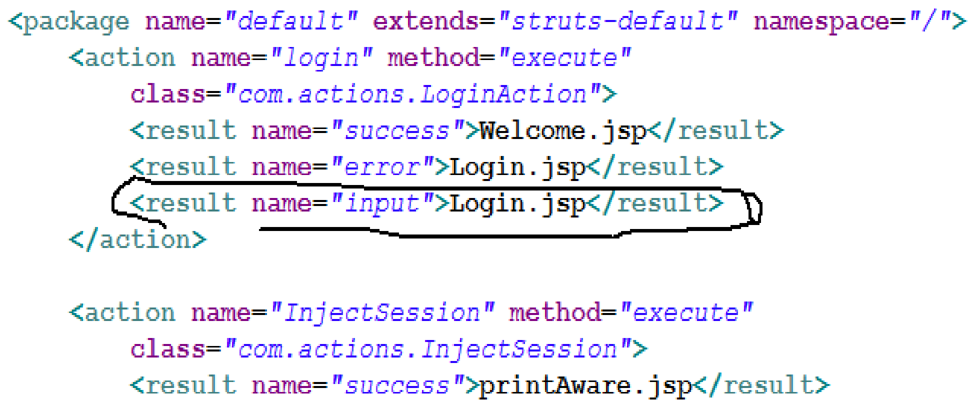
Modified Login Application



```java
@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"));
    }
}
```



```properties
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
```



```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>

    <action name="InjectSession" method="execute"
        class="com.actions.InjectSession">
        <result name="success">printAware.jsp</result>
```

- **Have validate() in action class........**
- **Have property file**
- **Mapping for "input" in struts.xml**

Modified Login Application

- **Order of execution of action is as follows:**
  - If action implements validateable interface , action validate() method is going to execute before execute() method 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);
```

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

#Aware Interfaces

Aware Interface

- Aware interface provides aka. Dependency 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);

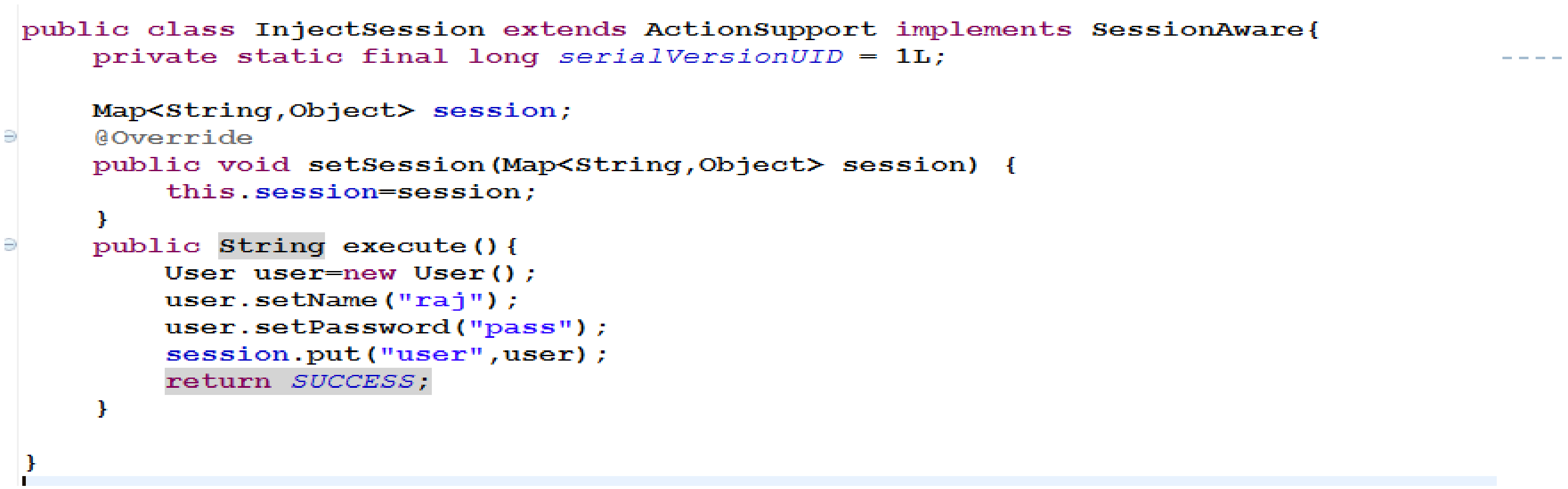
<<ServletResponseAware>>

    public void setervletResponseAware(HttpServletResponse res);

<<ServletRequestAware>>

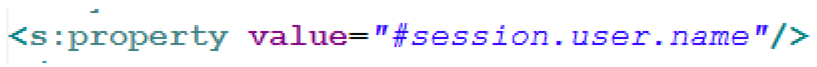
    public void setervletResponseAware(HttpServletRequest res)

Ex: Setting something in session scope



```
public class InjectSession extends ActionSupport implements SessionAware{
    private static final long serialVersionUID = 1L;

    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 accessing session scoped variable 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
  - Action act as a data carrier (DTO)
  - Action also working as controller (As in simple Servlet application)
  - We should not write business logic in action rather we should call DAO form it.
- **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.

More about Struts 2 Actions classes

- **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 behaviour is to use the execute() method.

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

# Namespace, Multiple mapping files, Dynamic Method Invocation

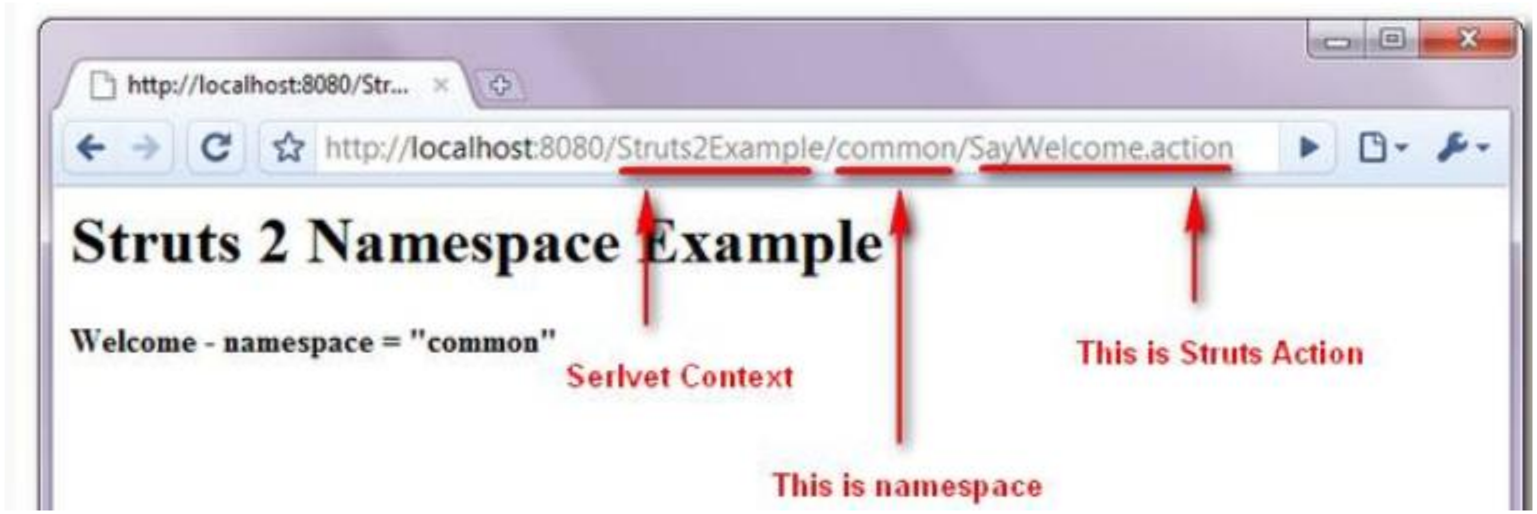
Namespace

- 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

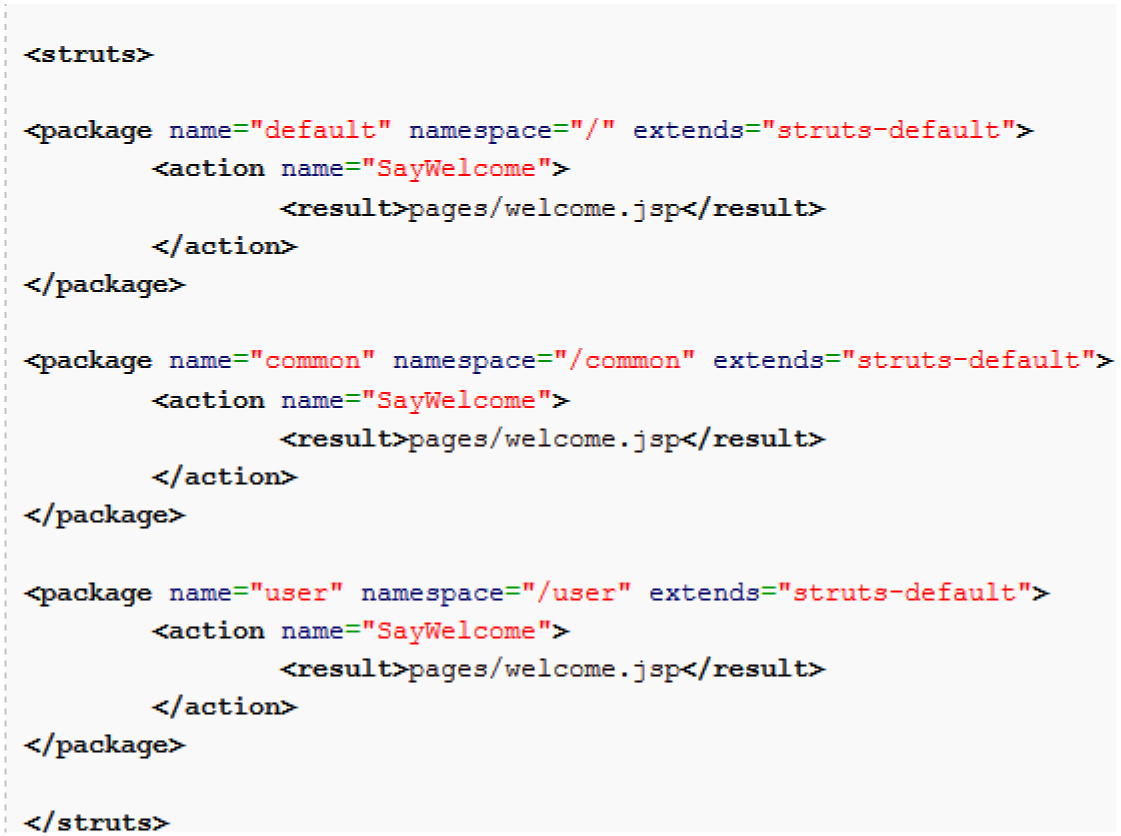
Struts 2 Namespace

- 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


http://localhost:8080/Str...
http://localhost:8080/Struts2Example/common/SayWelcome.action
Serlvet Context
This is namespace
This is Struts Action

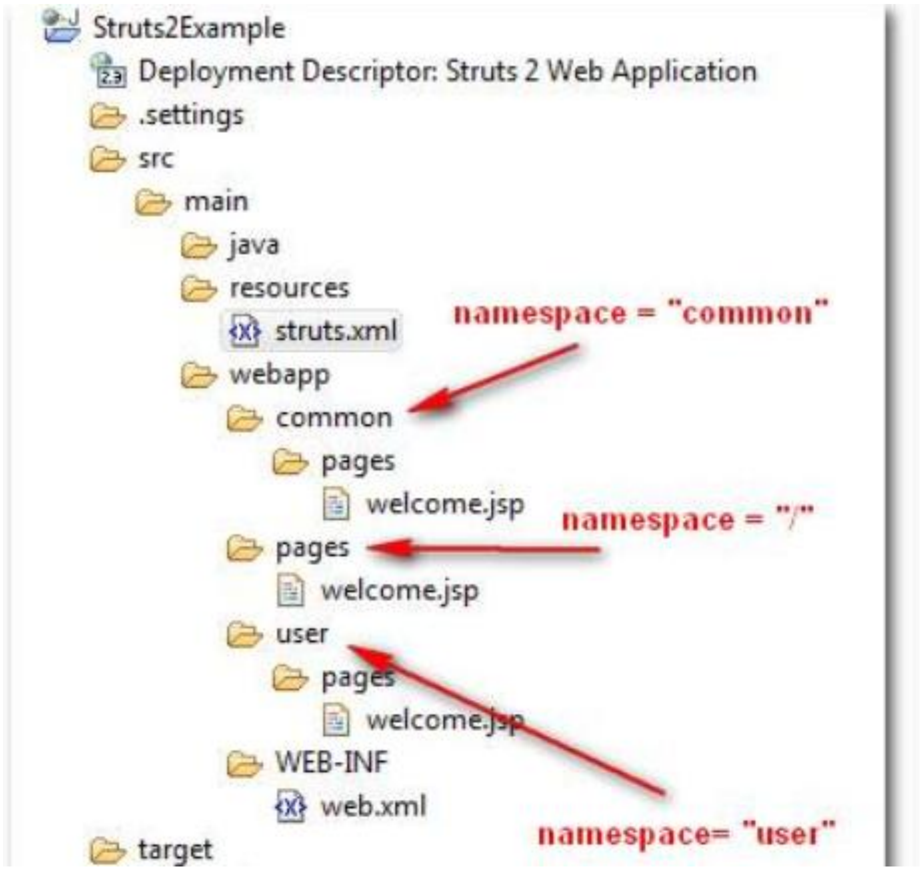
Struts 2 Namespace Example

Welcome - namespace = "common"



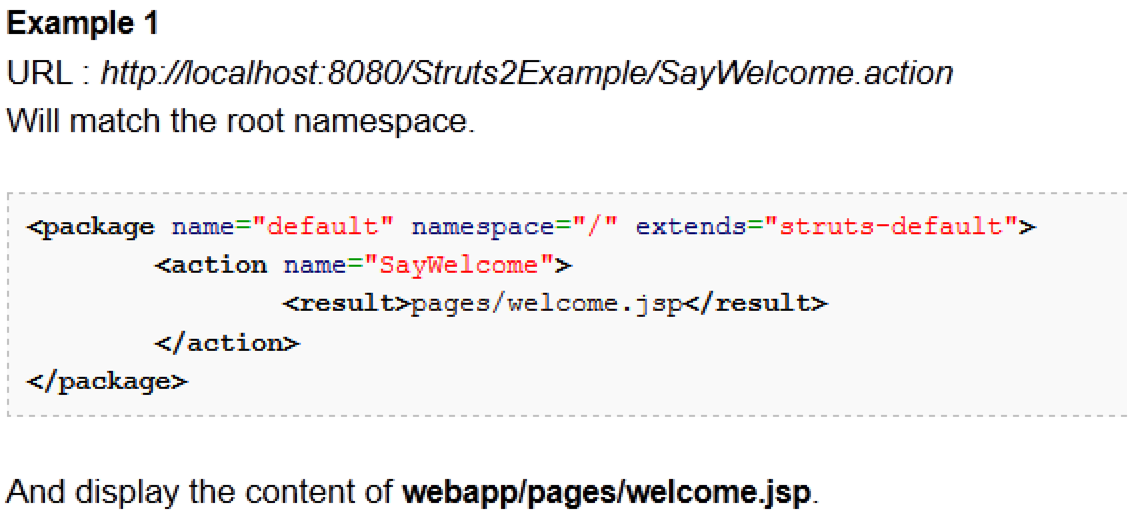
```
<struts>

<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>
```

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



Struts 2 action namespace map to folder structure.

Mapping how it works?



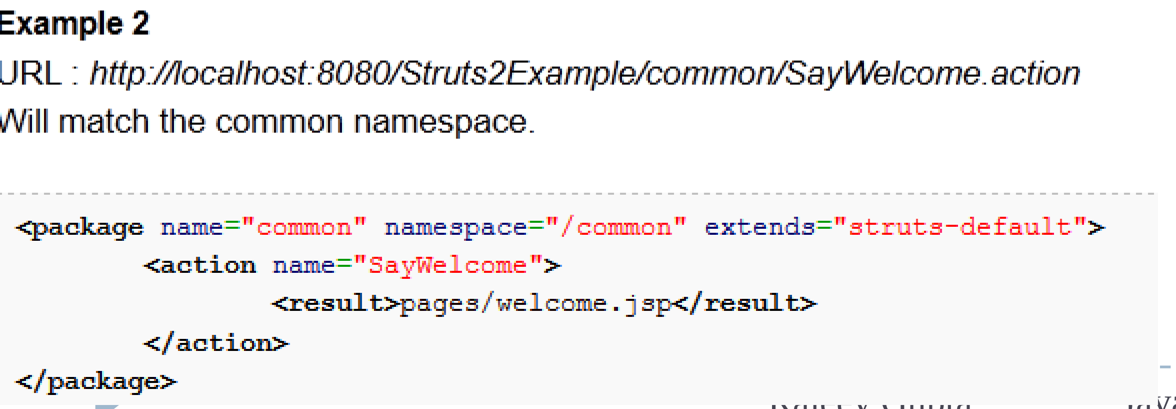
Example 1

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>
```

Mapping how it works?

Example 3

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

Will match the user namespace.

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

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

Multiple Struts configuration files

- In Struts 2, we should always assign each module a struts configuration file.
- Lets assume that we have two application modules user and audit, In this case, we can create three files :
  - struts-audit.xml – Put all audit module settings here.
  - struts-user.xml – Put all user modules settings here.
  - struts.xml – Put default settings and include the struts-audit.xml and struts-user.xml.

struts-audit.xml

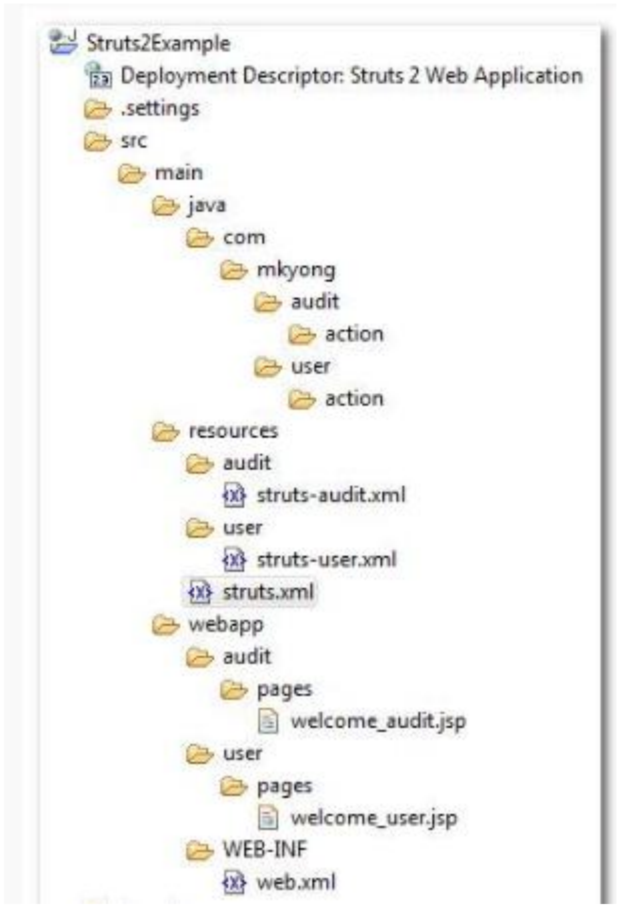
```
<package name="audit" namespace="/audit" extends="struts-default">
        <action name="WelcomeAudit">
                <result>pages/welcome_audit.jsp</result>
        </action>
</package>
```

struts-user.xml

```
<package name="user" namespace="/user" extends="struts-default">
        <action name="WelcomeUser">
                <result>pages/welcome_user.jsp</result>
        </action>
</package>
```

struts.xml

```
<struts>

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

<include file="user/struts-user.xml"></include>
<include file="audit/struts-audit.xml"></include>

</struts>
```

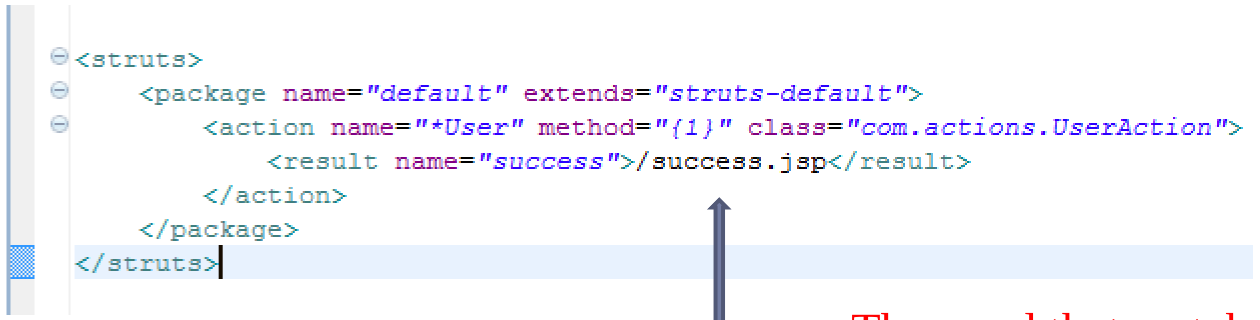


```
Struts2Example
  Deployment Descriptor: Struts 2 Web Application
  .settings
  src
    main
      java
        com
          mkyong
            audit
              action
            user
              action
      resources
        audit
          struts-audit.xml
        user
          struts-user.xml
        struts.xml
      webapp
        audit
          pages
            welcome_audit.jsp
        user
          pages
            welcome_user.jsp
        WEB-INF
          web.xml
```

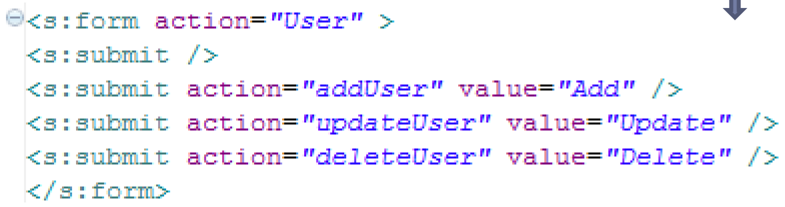
Dynamic Method Invocation

- 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



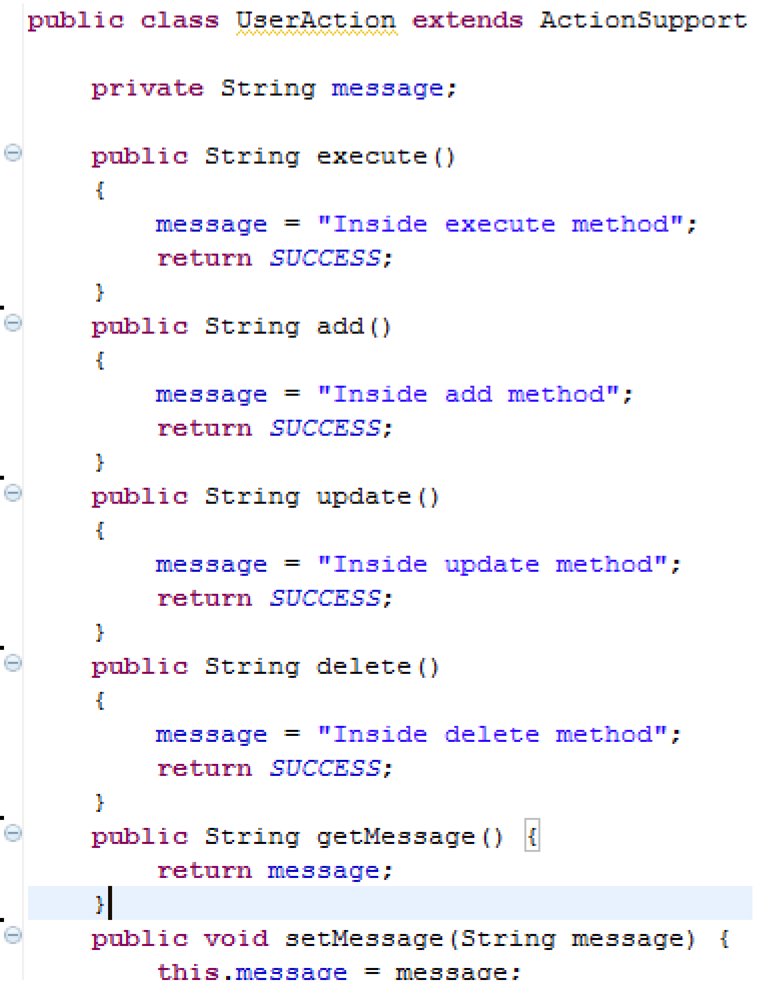
```
<struts>
    <package name="default" extends="struts-default">
        <action name="*User" method="{1}" class="com.actions.UserAction">
            <result name="success">/success.jsp</result>
        </action>
    </package>
</struts>
```



```
<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>
```

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.

Dynamic Method Invocation: Action class



```
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;
```

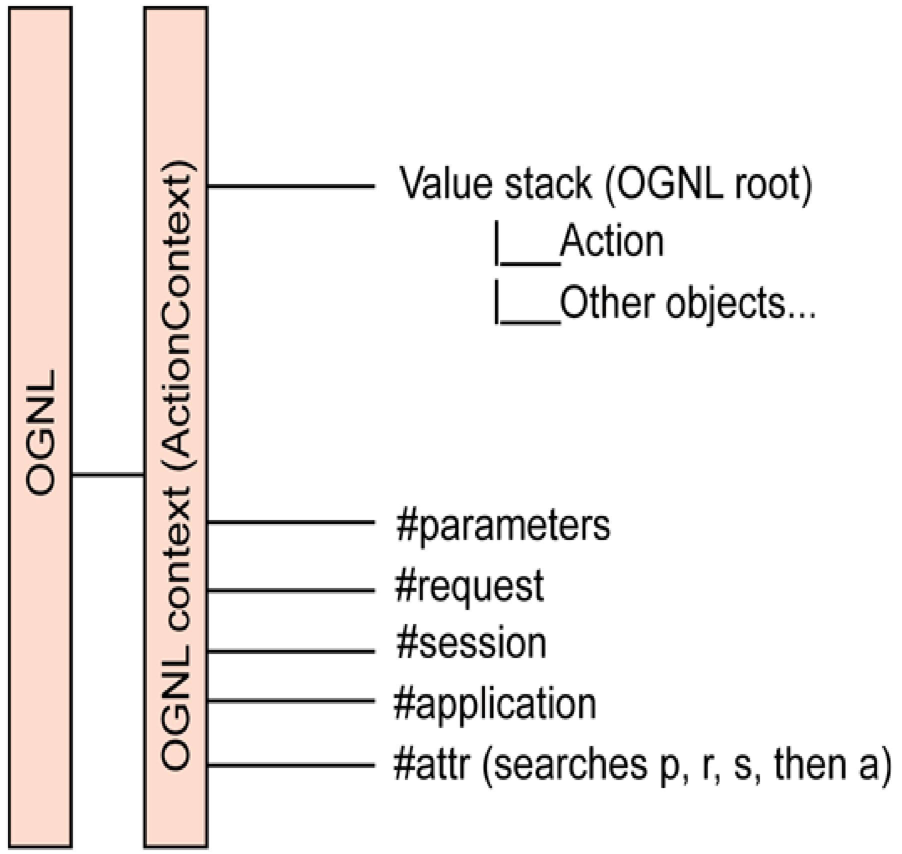
#OGNL, Value Stack

OGNL Object Graph Navigation Language

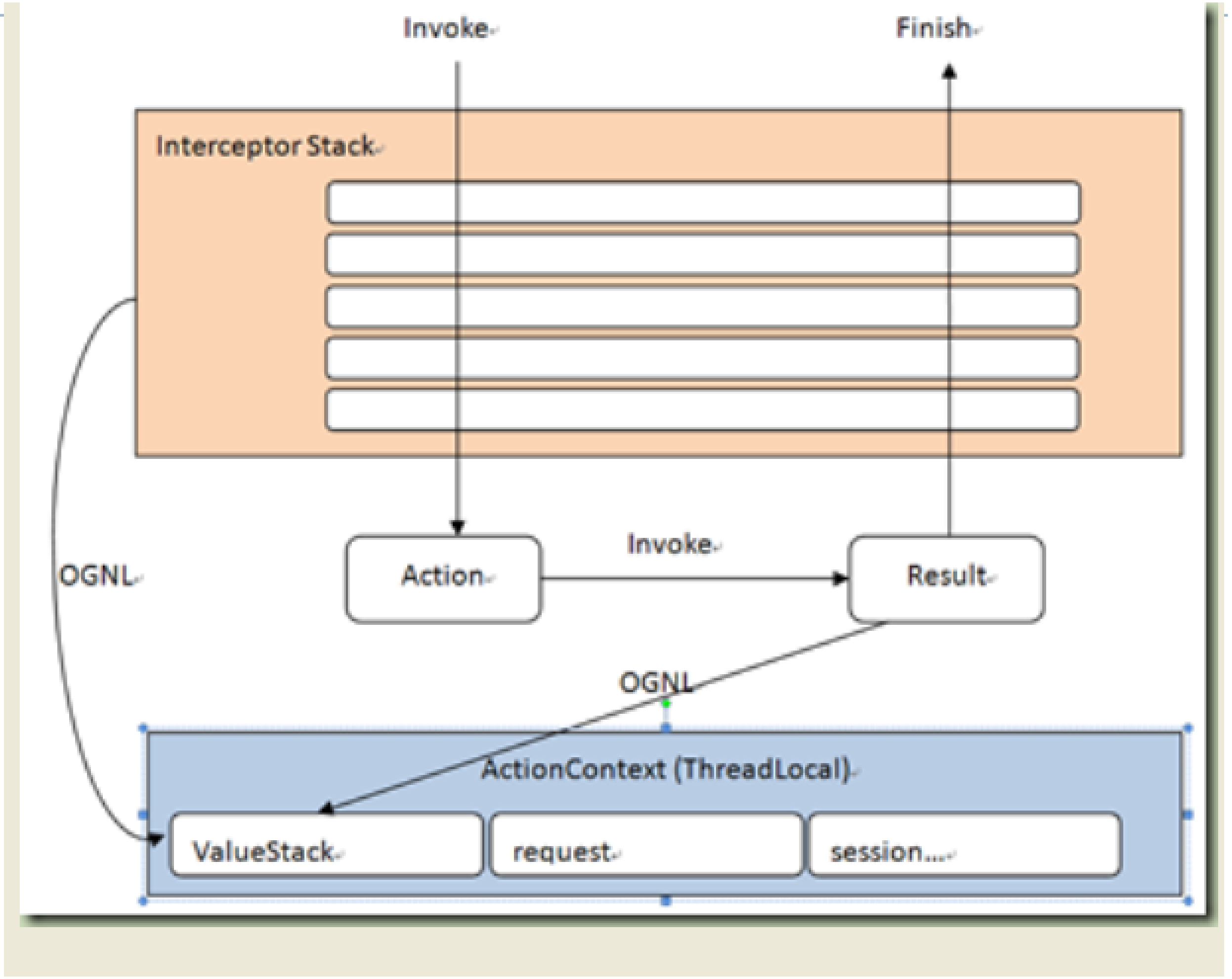
- OGNL is a powerful expression language that is used to reference and manipulate data on the ValueStack.
- OGNL also helps in data transfer and type conversion.
- The OGNL is very similar to the JSP Expression Language.
- OGNL is based on the idea of having a root or default object within the context

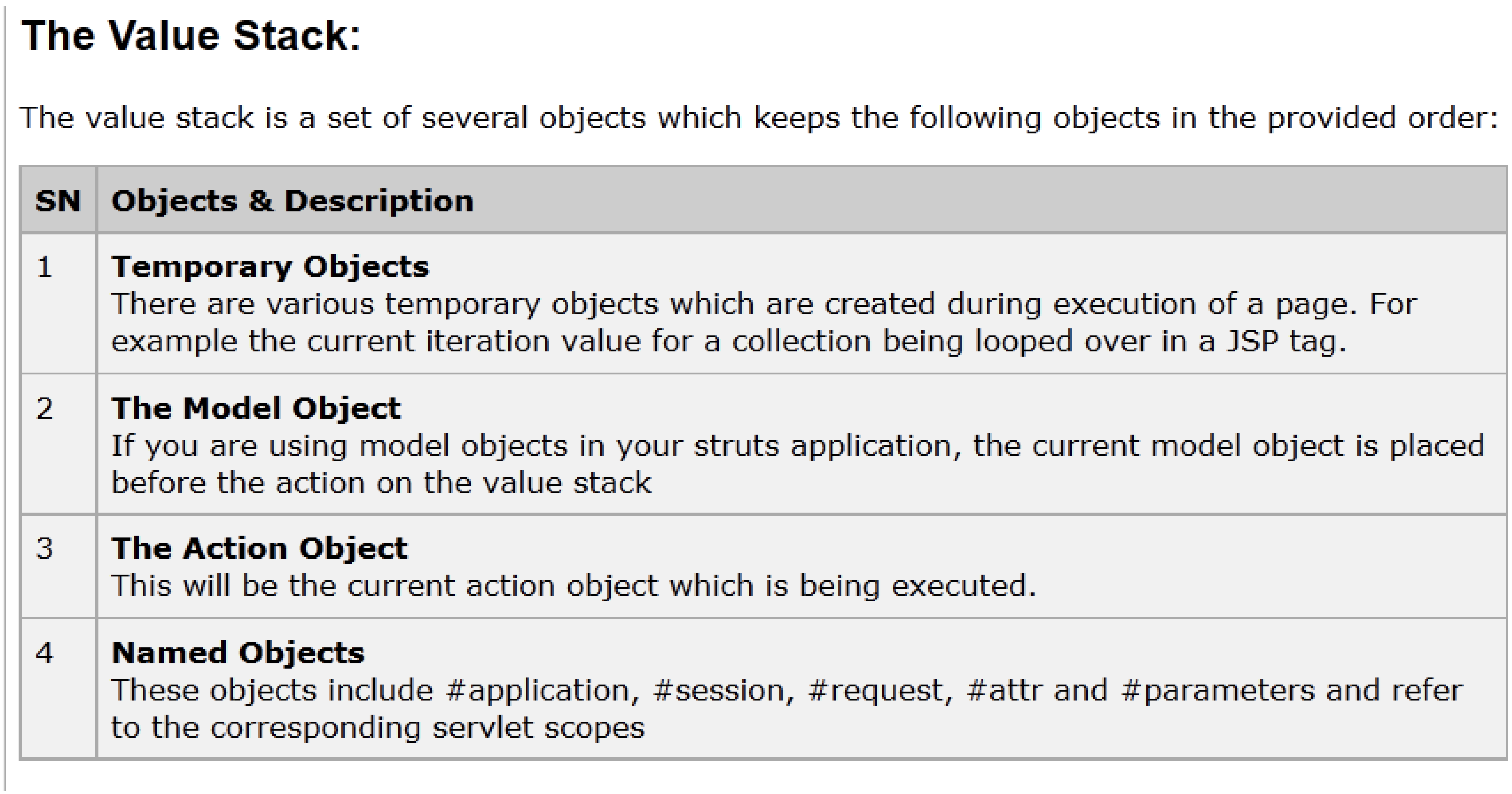
Struts 2 and OGNL

Figure 2. OGNL stack



- 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.


Invoke
Finish
Interceptor Stack
OGNL
Action
Invoke
Result
OGNL
ActionContext (ThreadLocal)
ValueStack
request
session....



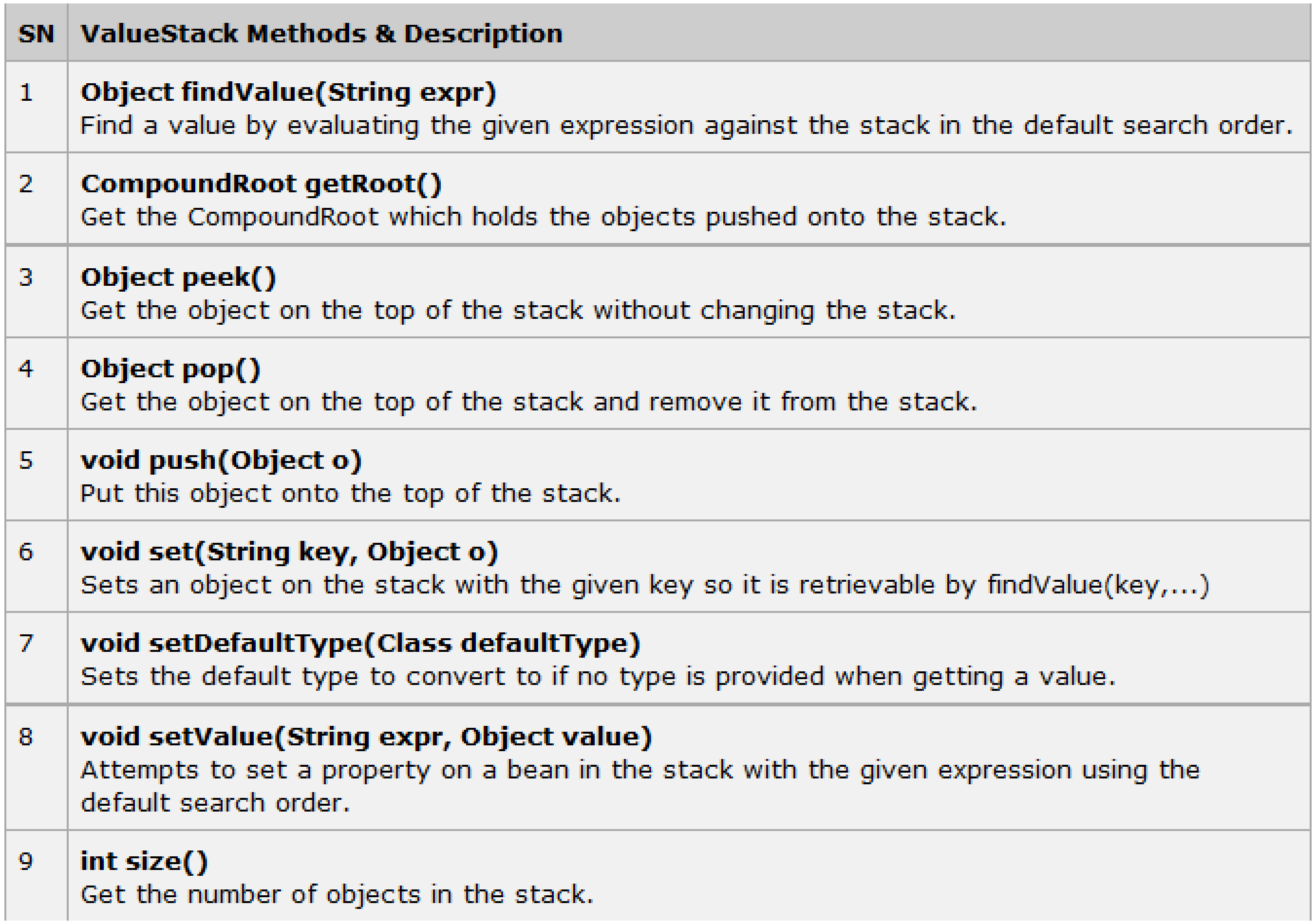
The Value Stack:

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

SN	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

can get valueStack object inside your action as follows

```
ActionContext.getContext().getValueStack()
```



SN	ValueStack Methods & Description
1	**Object findValue(String expr)** Find a value by evaluating the given expression against the stack in the default search order.
2	**CompoundRoot getRoot()** Get the CompoundRoot which holds the objects pushed onto the stack.
3	**Object peek()** Get the object on the top of the stack without changing the stack.
4	**Object pop()** Get the object on the top of the stack and remove it from the stack.
5	**void push(Object o)** Put this object onto the top of the stack.
6	**void set(String key, Object o)** Sets an object on the stack with the given key so it is retrievable by findValue(key,...)
7	**void setDefaultType(Class defaultType)** Sets the default type to convert to if no type is provided when getting a value.
8	**void setValue(String expr, Object value)** Attempts to set a property on a bean in the stack with the given expression using the default search order.
9	**int size()** Get the number of objects in the stack.

Once you have a ValueStack object, you can use following methods to manipulate that object

As mentioned earlier, OGNL is based on a context and Struts builds an ActionContext map for use with OGNL. The ActionContext map consists of the following:

1. **application** - application scoped variables
2. **session** - session scoped variables
3. **root / value stack** - all your action variables are stored here
4. **request** - request scoped variables
5. **parameters** - request parameters
6. **atributes** - the attributes stored in page, request, session and application scope

It is important to understand that the Action object is always available in the value stack. So, therefore if your Action object has properties x and y there are readily available for you to use.

Objects in the ActionContext are referred using the pound symbol, however, the objects in the value stack can be directly referenced, for example if **employee** is a property of an action class then it can ge referenced as follows:

```
<s:property value="name"/>
```

If you have an attribute in session called "login" you can retrieve it as follows:

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

OGNL also supports dealing with collections - namely Map, List and Set. For example to display a dropdown list of colors, you could do:

```
<s:select name="color" list="{'red','yellow','green'}" />
```

The OGNL expression is clever to interpret the "red","yellow","green" as colours and build a list based on that.

# Control tags

Generic tags

- Struts2 tags are divided into generic and UI tags

  - 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
  - UI tags
    - Concern about form creation

Control tags

if

```
<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

A kind of 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>
```

Control tags

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>
```

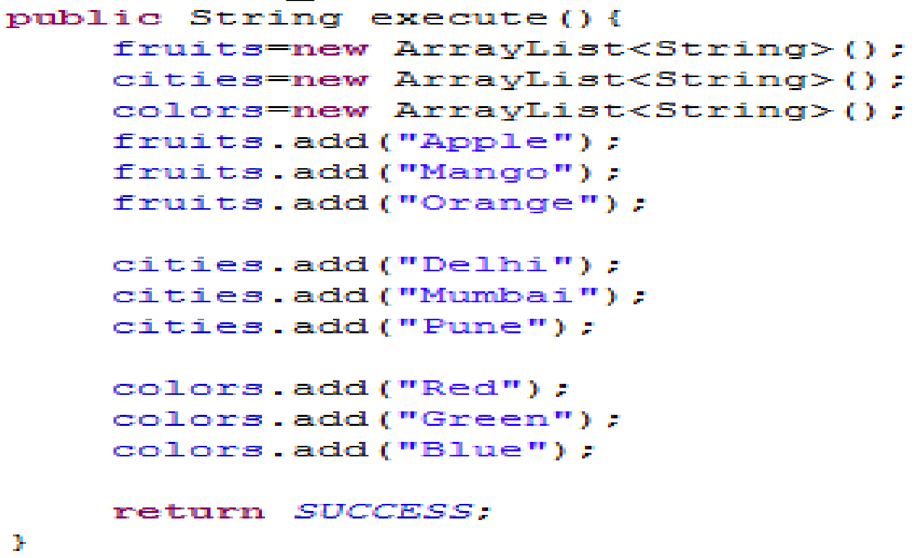
Now:

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

- So many tags, explore yourself
- merge
- sort
- subset
- generator
- elseIf

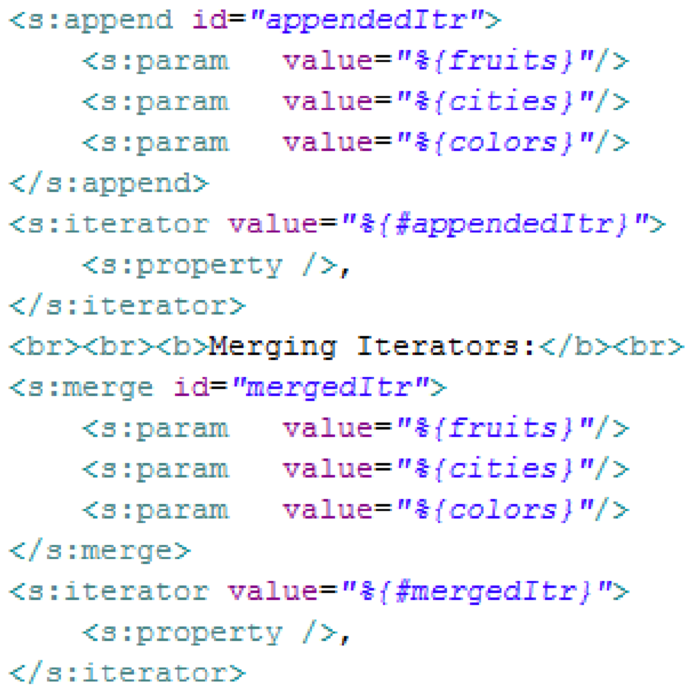
Control tags

- **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

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

- Examples

a

Similer to <a href..../>

Ex:

```
<s:url id="url" action="addAction"></s:url>
<s:a href="%{url}">adding</s:a>
```

Data tags

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

Data tags

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>
```

Data tags

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>
```

Data tags

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"/>
```

Lots of tags: please explore ☺

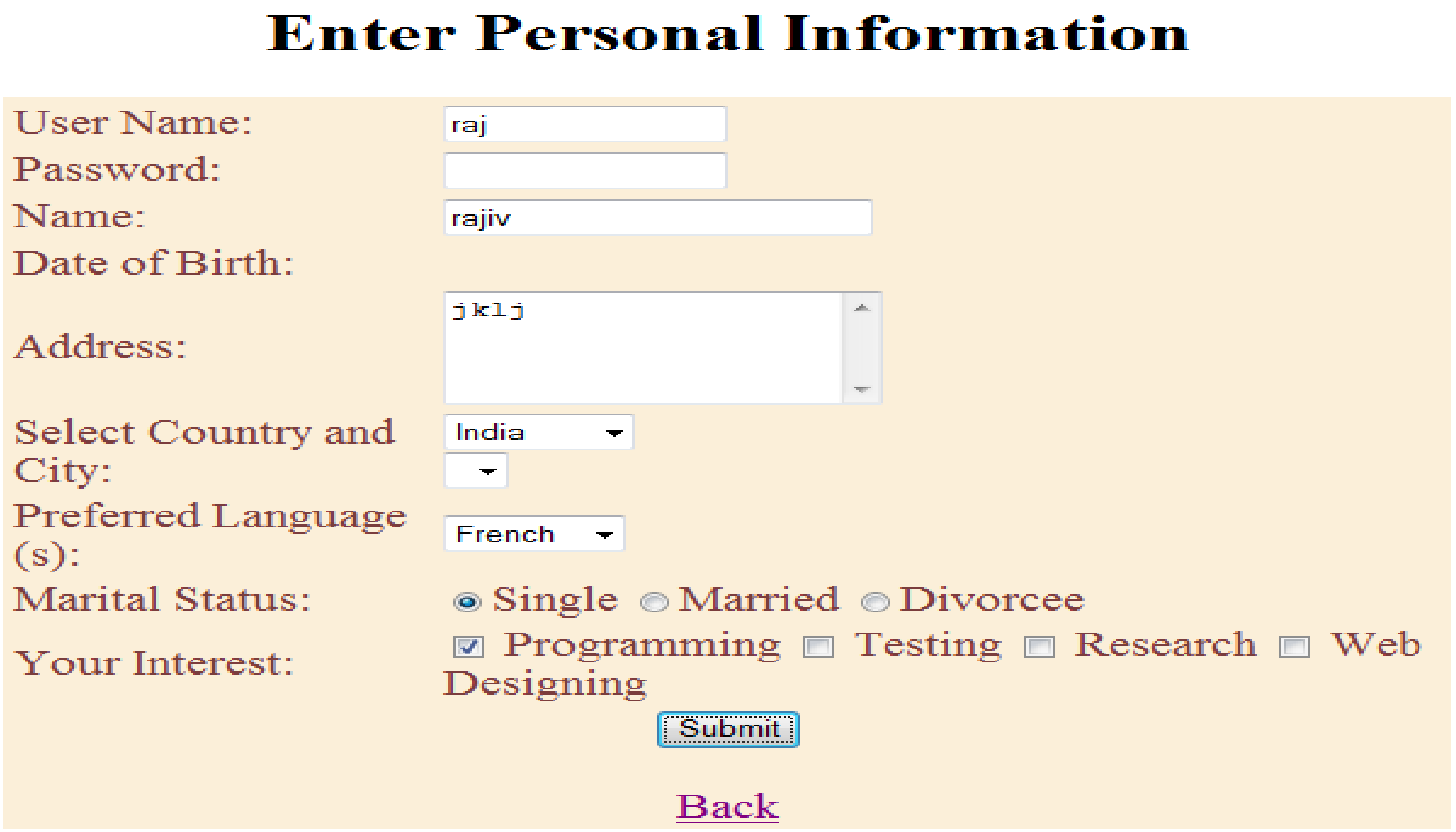
Bean	set	text
url	property	debug
I8n etc…		

# UI tags

UI tags

- form
- checkboxlist
- file
- token
- password
- textarea
- checkbox
- select
- radio
- head
- optiontransferselect
- reset
- updownselect
- Label
- Hidden
- Doubleselect
- Combobox
- Submit
- Datetimepicker
- Optgroup
- textfield

UI tags



Enter Personal Information

User Name:	raj
Password:	
Name:	rajiv
Date of Birth:	
Address:	jklj
Select Country and City:	India
Preferred Language (s):	French
Marital Status:	[✓ Single] Married Divorcee
Your Interest:	[✓ Programming] Testing Research Web Designing

Submit

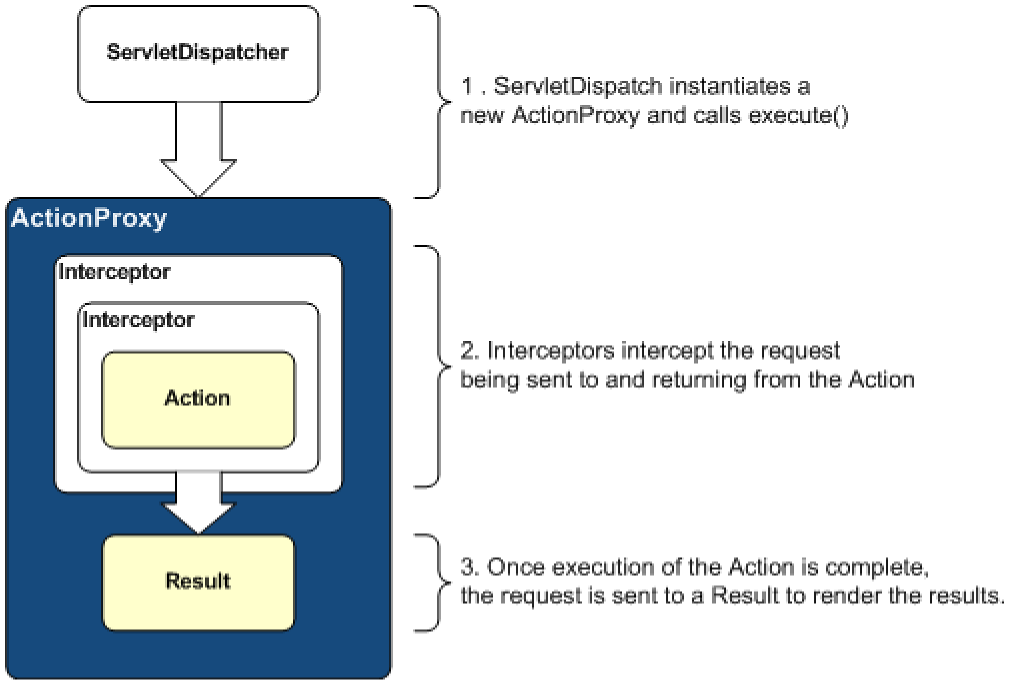
Back

- Please refer examples shared…

# Interceptors

Interceptors

- Can execute code before and after execution
- Are thread-safe
- Can be used for
  - Validation
  - Pre populating fields
  - Double-submit prevention
  - Session control
  - Authentication
  - Type conversion


ServletDispatcher
1 . ServletDispatch instantiates a new ActionProxy and calls execute()
ActionProxy
Interceptor
Interceptor
Action
2. Interceptors intercept the request being sent to and returning from the Action
Result
3. Once execution of the Action is complete, the request is sent to a Result to render the results.

Interceptors

- 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
- Interceptors are conceptually the same as servlet filters.

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

Interceptors

- Struts2 comes with default list of Interceptors already configured in the application in struts-default.xml file. We can create our own custom Interceptors and plugin into a Struts2 based web application.

- Framework creates an object of **ActionInvocation** that encapsulates the action and all the interceptors configured for that action.

- Each interceptors are called before the action gets called. Once the action is called and result is generated, each interceptors are again called in reverse order to perform post processing work.

- Interceptors can alter the workflow of action. It may prevent the execution of action.

Interceptors

- **Interceptor 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 Interceptors in Struts 2(I)

- **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

Some Interceptors in Struts 2 (II)

- **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.

Some Interceptors in Struts 2 (III)

- **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.

Some Interceptors in Struts 2 (IV)

- **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 form submission.
- **Validation**
  - Provides validation support for actions

Interceptors

- **How to use Interceptors?**
  - **Interceptor need to be configure in struts.xml file as**

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

Struts 2 developers are used to declare the actions belong to a package that extend the "**struts-default**", which contains the default set of interceptors.

```
<package name="default" namespace="/" extends="struts-default">
        <action name="testingAction"
                class="com.mkyong.common.action.TestingAction" >
                <result name="success">pages/result.jsp</result>
        </action>
</package>
```

The default set of interceptors are grouped as "**defaultStack**" in **struts-default.xml** file, which is located in the **struts2-core.jar** file. The "**defaultStack**" provides all the core Struts 2 functionality, which is suit the need of most application.

Try study the **struts-default.xml** file, it's always the best interceptors reference.

Mapping interceptor to actions

To map other interceptors to action, use the “**interceptor-ref**” element.

```
<package name="default" namespace="/" extends="struts-default">
        <action name="testingAction"
                class="com.mkyong.common.action.TestingAction" >
                <interceptor-ref name="timer"/>
                <interceptor-ref name="logger"/>
                <result name="success">pages/result.jsp</result>
        </action>
</package>
```

In above snippet code, it map the “**timer**” and “**logger**” interceptors to the “**TestingAction**” action class via “**interceptor-ref**” element.

Important to note…

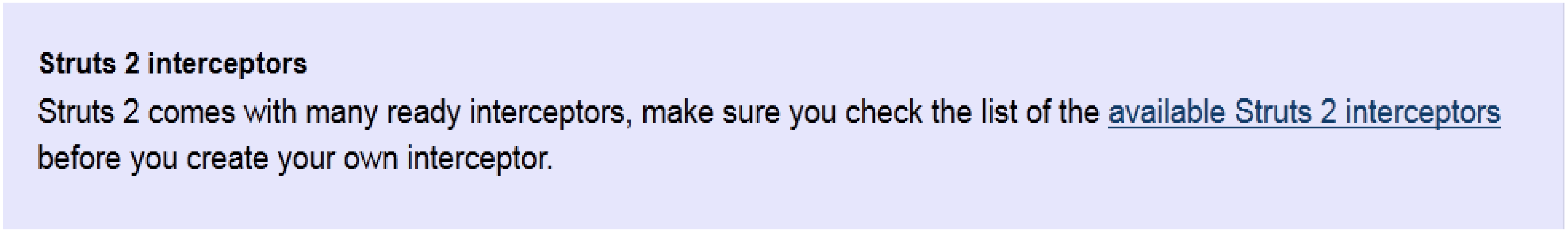
Since the “**TestingAction**” is declared it’s own interceptors, **it’s immediate loses all the inherit default set of interceptors**, you must explicitly declare the “**defaultStack**” in order to use it, see below example.

```
<package name="default" namespace="/" extends="struts-default">
        <action name="testingAction"
                class="com.mkyong.common.action.TestingAction" >
                <interceptor-ref name="timer"/>
                <interceptor-ref name="logger"/>
                <interceptor-ref name="defaultStack"/>
                <result name="success">pages/result.jsp</result>
        </action>
</package>
```

Custom interceptor

- Steps

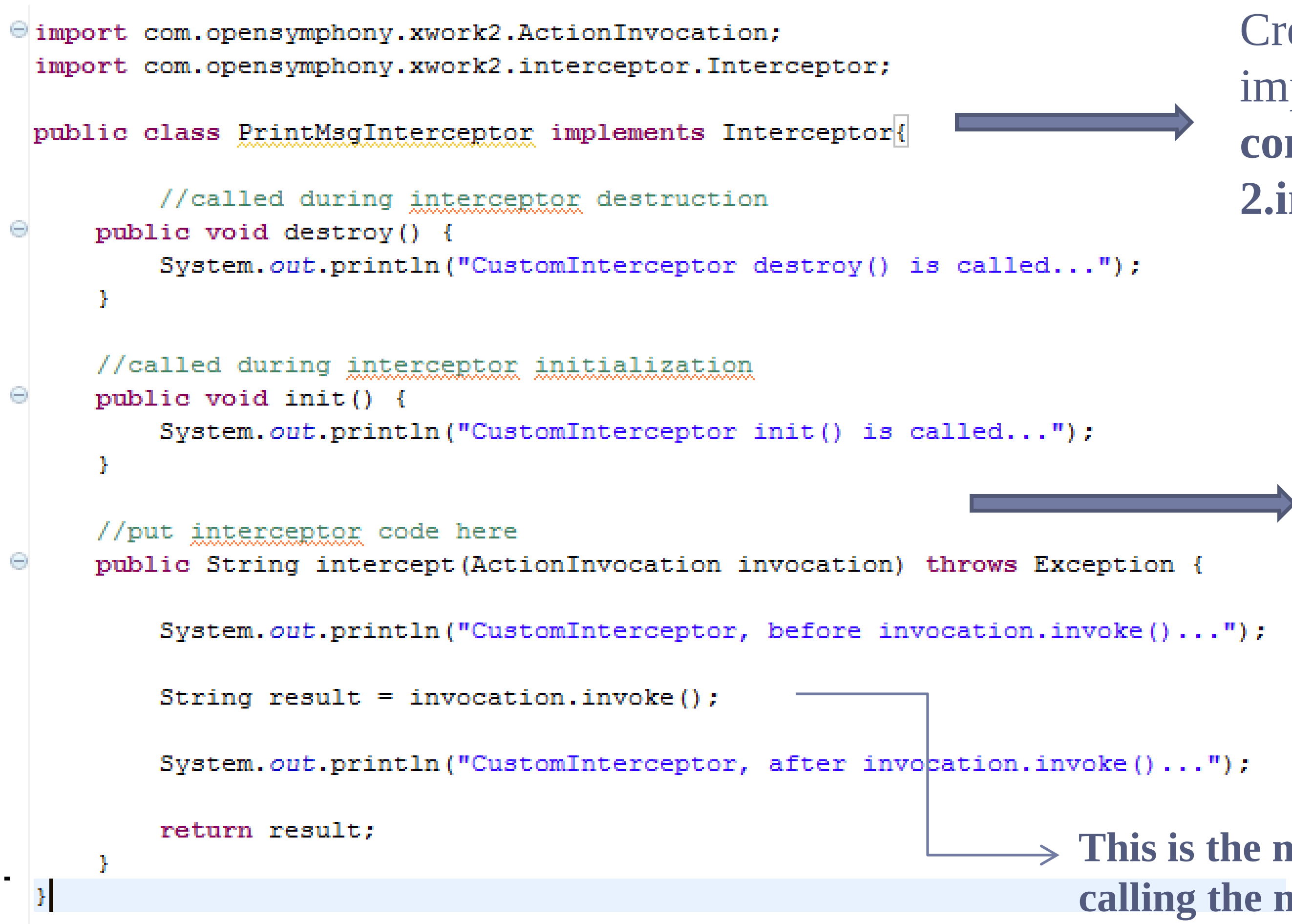
1. Create a class that implements **Interceptor**.
   and Implement the **intercept(ActionInvocation invocation)** method.
2. Configure the interceptor in the **struts.xml and** Link it to action.



Struts 2 interceptors

Struts 2 comes with many ready interceptors, make sure you check the list of the available Struts 2 interceptors before you create your own interceptor.

Step 1:



```
import com.opensymphony.xwork2.ActionInvocation;
import com.opensymphony.xwork2.interceptor.Interceptor;

public class PrintMsgInterceptor implements Interceptor{

        //called during interceptor destruction
    public void destroy() {
        System.out.println("CustomInterceptor destroy() is called...");
    }

    //called during interceptor initialization
    public void init() {
        System.out.println("CustomInterceptor init() is called...");
    }

    //put interceptor code here
    public String intercept(ActionInvocation invocation) throws Exception {

        System.out.println("CustomInterceptor, before invocation.invoke()...");

        String result = invocation.invoke();

        System.out.println("CustomInterceptor, after invocation.invoke()...");

        return result;
    }
}
```

Create a class that implements **com.opensymphony.xwork2.interceptor.Interceptor**.

Implement the **intercept(ActionInvocation invocation)**

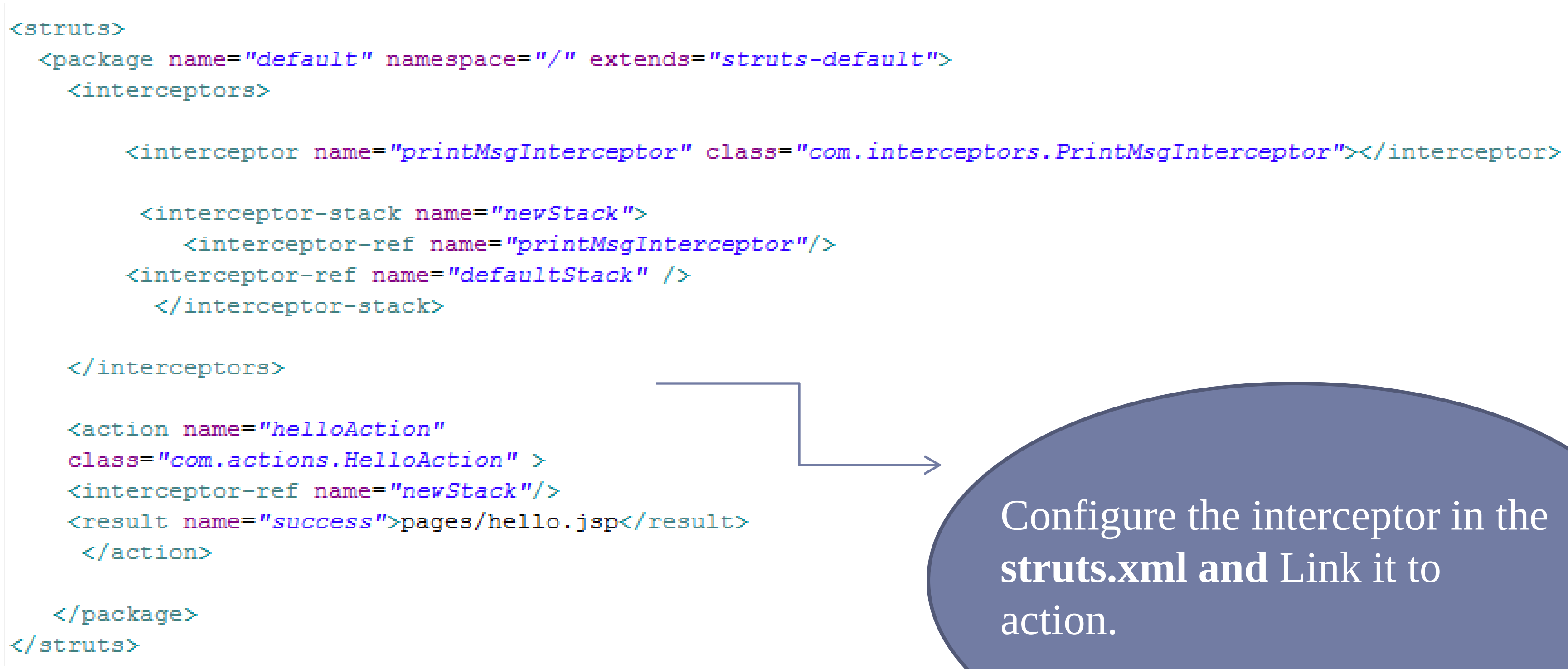
This is the method responsible for calling the next interceptor or the action

Explanation
The interceptor class must implements the **com.opensymphony.xwork2.interceptor.Interceptor interface**. During interceptor initialization, **init()** is called; interceptor destruction, **destroy()** is called. In last, put all interceptor code that does the work inside the **intercept(ActionInvocation invocation)** method.

invocation.invoke()
In the interceptor intercept() method, you **must called the invocation.invoke()** and return it's result. This is the method responsible for calling the next interceptor or the action. The action will failed to continue without calling the **invocation.invoke()** method.

destroy() is not reliable
It's not recommend to put any code inside the **destroy()**, because this method is not reliable. When your application server is force shutdown or be killed by command, the **destroy()** will not be called.

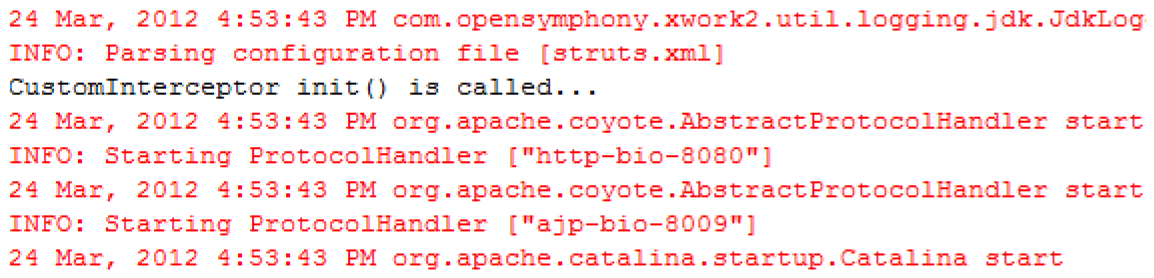
Step 2:



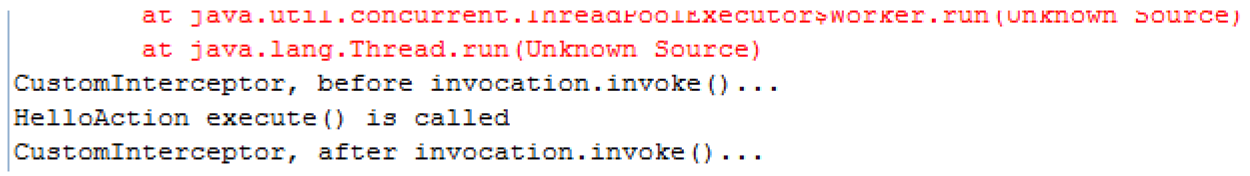
```
<struts>
  <package name="default" namespace="/" extends="struts-default">
    <interceptors>

        <interceptor name="printMsgInterceptor" class="com.interceptors.PrintMsgInterceptor"></interceptor>

         <interceptor-stack name="newStack">
            <interceptor-ref name="printMsgInterceptor"/>
        <interceptor-ref name="defaultStack" />
          </interceptor-stack>

    </interceptors>

    <action name="helloAction"
    class="com.actions.HelloAction" >
    <interceptor-ref name="newStack"/>
    <result name="success">pages/hello.jsp</result>
     </action>

   </package>
</struts>
```

Configure the interceptor in the **struts.xml and** Link it to action.

Interceptor will execute…



```
24 Mar, 2012 4:53:43 PM com.opensymphony.xwork2.util.logging.jdk.JdkLog
INFO: Parsing configuration file [struts.xml]
CustomInterceptor init() is called...
24 Mar, 2012 4:53:43 PM org.apache.coyote.AbstractProtocolHandler start
INFO: Starting ProtocolHandler ["http-bio-8080"]
24 Mar, 2012 4:53:43 PM org.apache.coyote.AbstractProtocolHandler start
INFO: Starting ProtocolHandler ["ajp-bio-8009"]
24 Mar, 2012 4:53:43 PM org.apache.catalina.startup.Catalina start
```



```
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
        at java.lang.Thread.run(Unknown Source)
CustomInterceptor, before invocation.invoke()...
HelloAction execute() is called
CustomInterceptor, after invocation.invoke()...
```

# validation framework

Validation

- **2 ways to do validation**
  1. with the help of ActionSupport
  2. XML way, more flexible

Validation with with the help of ActionSupport

- Steps

  Create an form

```
<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 validate() method to 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");
    }
  }
```

Validation with with the help of ActionSupport

- Don’t forget to map for “input” in strut.xml

```
<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>
```

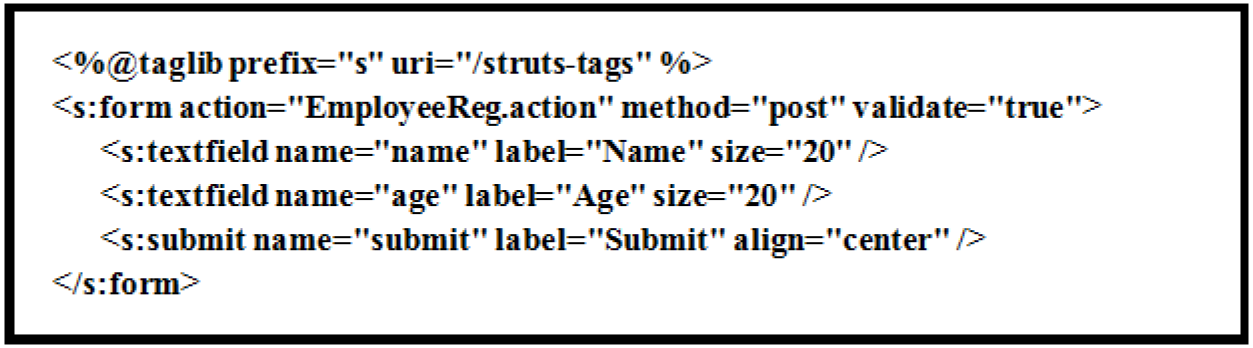
- When the user presses the submit button, Struts 2 will automatically execute the validate method and if any of the if statements listed inside the method are true, Struts 2 will call its addFieldError method. If any errors have been added then Struts 2 will not proceed to call the execute method. Rather 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 validation

- More flexible and powerful.
- Steps

1. Create registration 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>
```

2. Create Action class say EmployeeReg

Struts - XML Based validation

```
<!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>
```

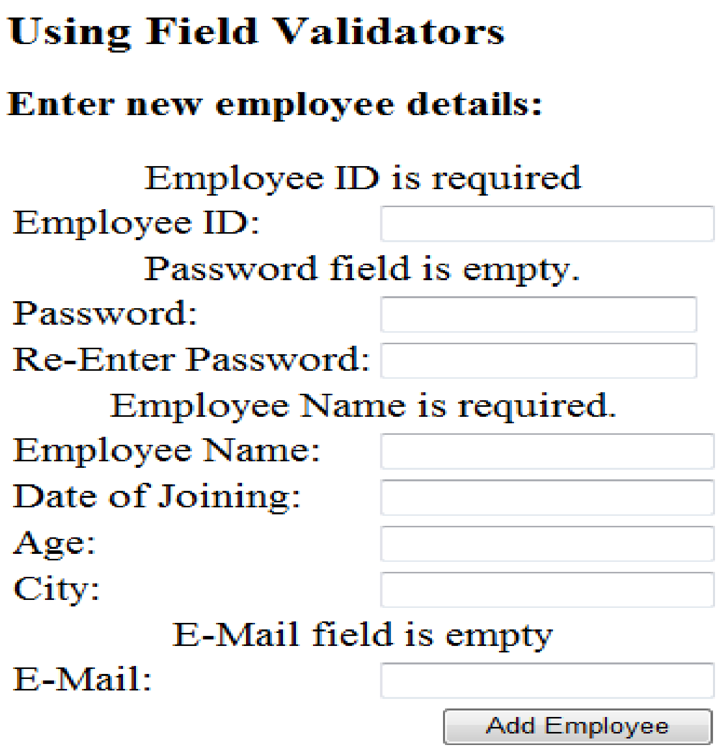
3. Now create EmployeeReg-validation.xml validation file in same package in which EmployeeReg is stored.

- If the action **EmployeeReg** then name of validation file must be **EmployeeReg-validation.xml**
  - create validation xml file in '[action-class]'-validation.xml

- **If we want Client side validation**
- Just add validate="true“ to <s:form…> this option let java script produce at client side……

Struts-XML based validation

- Refer given code…



Using Field Validators

Enter new employee details:

Employee ID is required

Employee ID:	

Password field is empty.

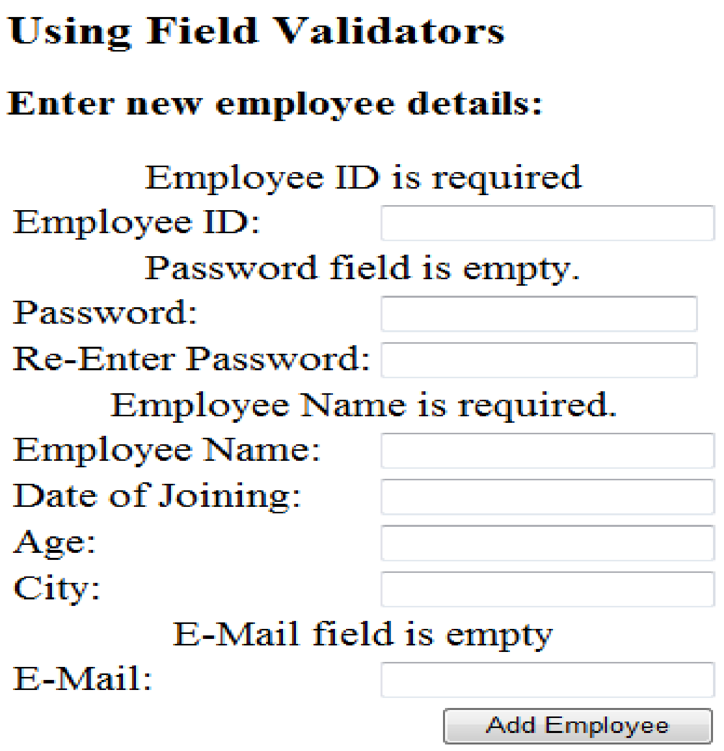
Password:	
Re-Enter Password:	

Employee Name is required.

Employee Name:	
Date of Joining:	
Age:	
City:	

E-Mail field is empty

E-Mail:	



Using Field Validators

Enter new employee details:

Employee ID is required

Employee ID:	

Password field is empty.

Password:	
Re-Enter Password:	

Employee Name is required.

Employee Name:	
Date of Joining:	
Age:	
City:	

E-Mail field is empty

E-Mail:	

# Struts 2 Type Conversion

Struts 2 Type Conversion

- Struts2 provide automatically 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.....

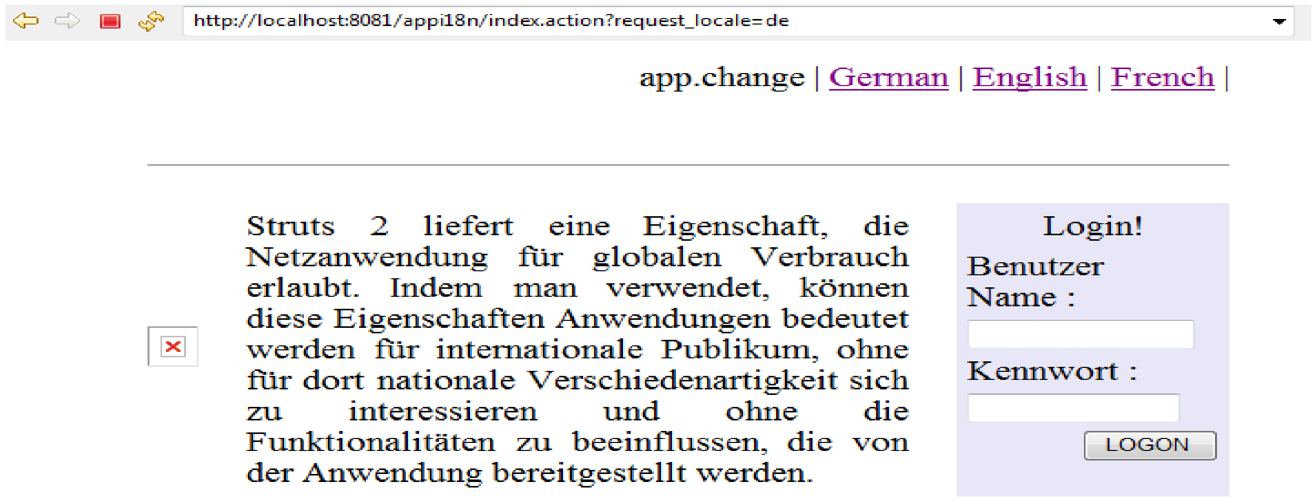
# Internationalization (i18n) support

Internationalization

- For implement i18n in we need

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

- **Hello world example (Refer code Provided)**



http://localhost:8081/appi18n/index.action?request_locale=de

app.change | German | English | French |

Struts 2 liefert eine Eigenschaft, die Netzanwendung für globalen Verbrauch erlaubt. Indem man verwendet, können diese Eigenschaften Anwendungen bedeutet werden für internationale Publikum, ohne für dort nationale Verschiedenartigkeit sich zu interessieren und ohne die Funktionalitäten zu beeinflussen, die von der Anwendung bereitgestellt werden.

Login!

Benutzer Name :

Kennwort :

LOGON

Internationalization

- We need not to worry about writing pages in different languages. All we 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.

Internationalization

- 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.

Thank
You