

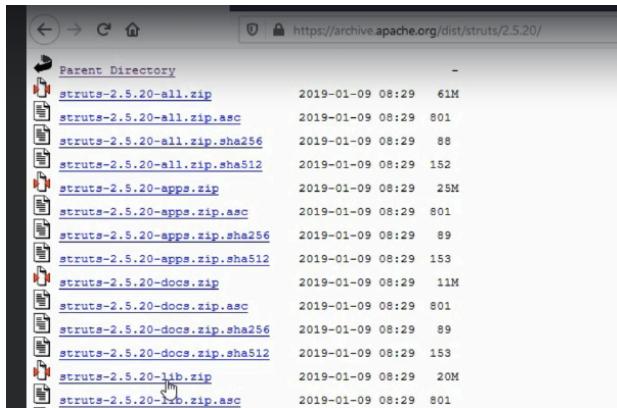
Struts 2 Annotations:

- Two ways of declaring Struts 2 architecture:-
1) XML-based configurations (struts.xml)
2) Annotations

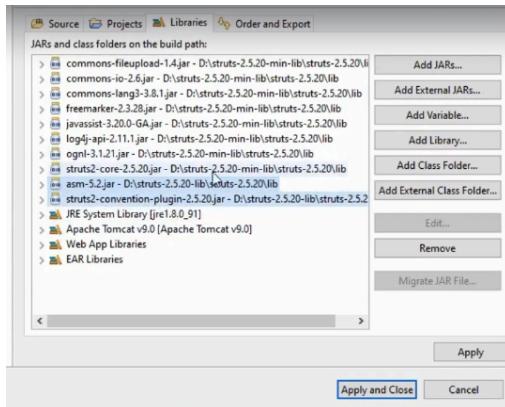
Benefits of annotations:-

- 1) Reduce our code to some extent
- 2) Provides support for some tools
- 3) Organizes code in a better way

We need additional jar files



We need these two additional jars



And also deployment assembly -> add -> select Java Build Path Entities -> select jars and finish.

Add success and error jsp pages

The screenshot shows the Struts2Annotations project structure in the left pane. It includes .settings, build, src, WebContent, META-INF, WEB-INF, lib, web.xml, error.jsp, success.jsp, .classpath, and .project files.

The right pane displays the content of success.jsp:

```
5<html>
6<head>
7 <meta http-equiv="Content-Type" content="text/html"/>
8 <title>Insert title here</title>
9 </head>
10<body>
11
12 <h1>Success page</h1>
13
14 </body>
15 </html>
```

Our action classes must be any one of this package. The below are the conventions we have to follow for annotation configuration.

**The following package names are allowed:-
action, actions, struts, struts2**



Annotations won't work for the last column in the above pic

Framework locates action classes with above package name and below classes convention and their implementation/extend.

Classes whose names end with 'Action'
Examples:- CalculateAction, AddProductAction

Classes which:-

- 1) Implement 'Action' interface
- 2) Extend 'ActionSupport' class

```
5
6<struts>
7@  <package name="test" extends="struts-default">
8@    <action name="testAction" class="TestAction">
9      <result name="success">/success.jsp</result>
10     <result name="error">/error.jsp</result>
11   </action>
12 </package>
13 </struts>
```

@Action

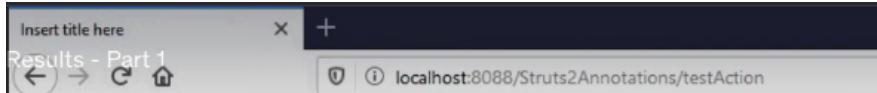
In line number 8 name refers to the endpoint and class refers to the action class which has an execute method. In that place we use this annotation. We can use this at either class level or at execute method as shown in below screenshot. The above one xml the below one java config.

The screenshot shows the Eclipse IDE interface. On the left is the Project Explorer view, displaying a Struts2Annotations project with various files like struts.xml, error.jsp, success.jsp, and TestAction.java. The central area shows the code editor with TestAction.java open. The code defines a class TestAction with an execute() method. An annotation @Action("/testAction") is placed above the class definition, and another annotation @Results is placed above the execute() method, mapping "success" to success.jsp and "error" to error.jsp.

```
1 package action;
2
3@import org.apache.struts2.convention.annotation.Action;
4 import org.apache.struts2.convention.annotation.Result;
5 import org.apache.struts2.convention.annotation.Results;
6
7 @Results({
8     @Result(name = "success", location = "/success.jsp"),
9     @Result(name = "error", location = "/error.jsp")
10 })
11 @Action("/testAction")
12 public class TestAction {
13
14@     public String execute()
15     {
16         System.out.println("execute() method called");
17         return "success";
18     }
19 }
```

This screenshot shows a closer look at the Java code in the TestAction.java file. It highlights the execute() method and its implementation. The method prints a message to the console and returns the string "success".

```
16     System.out.println("execute() method called");
17     return "success";
18 }
```



Combine Action and result annotations as follow (another approach)

```

1 package actions;
2
3@import org.apache.struts2.convention.annotation.Action;[]
4
5 @Action(value = "/testAction", results = { @Result(name = "success", location = "/success.jsp"),
6     @Result(name = "error", location = "/error.jsp") })
7
8 public class TestAction {
9
10    public String execute()
11    {
12        System.out.println("execute() method called");
13        return "error";
14    }
15
16 }
17

```

Finance Calculator Application

Calculate EMI Calculate Interest Rate Calculate Income Tax

<s:form /> <s:form /> <s:form />

Calculate Action

Markers Properties Servers Data Source Explorer Snippets Console Call Hierarchy

No consoles to display at this time.

Single action class is okay for multiple similar actions

```

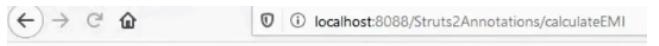
1 package actions;
2
3@import org.apache.struts2.convention.annotation.Action;[]
4
5
6 public class CalculateAction {
7
8    public String calculateEMI() {
9        System.out.println("calculateEMI() called");
10       return "success";
11    }
12
13    public String calculateInterestRate() {
14        System.out.println("calculateInterestRate() called");
15       return "success";
16    }
17
18    public String calculateIncomeTax() {
19        System.out.println("calculateIncomeTax() called");
20       return "success";
21    }
22
23 }
24

```

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE struts PUBLIC
3   "-//Apache Software Foundation//DTD Struts Configuration 2.5//EN"
4   "http://struts.apache.org/dtds/struts-2.5.dtd">
5
6<struts>
7   <package name="test" extends="struts-default">
8     <action name="calculateEMI" class="actions.CalculateAction" method="calculateEMI">
9       <result name="success"/>/success.jsp</result>
10    </action>
11    <action name="calculateInterestRate" class="actions.CalculateAction" method="calculateInterestRate">
12      <result name="success"/>/success.jsp</result>
13    </action>
14    <action name="calculateIncomeTax" class="actions.CalculateAction" method="calculateIncomeTax">
15      <result name="success"/>/success.jsp</result>
16    </action>
17  </package>
18 </struts>

```



Success page

```
Design Source
Markers Properties Servers Data Source Explorer Snippets Console Call Hierarchy
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (20-Jun-2020, 1:59:41 AM)
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
Jun 20, 2020 2:00:19 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in [33,595] milliseconds
calculateEMI() called
```

The above one is xml configuration and the below one is java configuration

```
ProductManagementApplication
RegistrationApplication
Servers
StrutsAnnotations
src
actions
CalculateAction.java
TestAction.java
struts.xml
WebContent
META-INF
WEB-INF
lib
web.xml
error.jsp
success.jsp
.classpath
.project
StrutsTags
StrutsTest
src
struts.xml
TestAction.java
WebContent
.classpath
.project
```

```
CalculateAction.java struts.xml
1 package actions;
2
3 import org.apache.struts2.convention.annotation.Action;
4
5 @Result(name = "success", location = "/success.jsp")
6 public class CalculateAction {
7
8     @Action("/calculateEMI")
9     public String calculateEMI() {
10         System.out.println("calculateEMI() called");
11         return "success";
12     }
13
14     @Action("/calculateInterestRate")
15     public String calculateInterestRate() {
16         System.out.println("calculateInterestRate() called");
17         return "success";
18     }
19
20     @Action("/calculateIncomeTax")
21     public String calculateIncomeTax() {
22         System.out.println("calculateIncomeTax() called");
23     }
24 }
```

```
Markers Properties Servers Data Source Explorer Snippets Console Call Hierarchy
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (20-Jun-2020, 2:05:43 AM)
INFO: Server startup in [14,264] milliseconds
calculateEMI() called
calculateInterestRate() called
calculateIncomeTax() called
```

Now let us map different jsp pages for different methods

```
CalculateAction.java struts.xml
1 package actions;
2
3 import org.apache.struts2.convention.annotation.Action;
4
5 public class CalculateAction {
6
7     @Action(value="/calculateEMI", results = {@Result(name = "success", location = "/success.jsp")})
8     public String calculateEMI() {
9         System.out.println("calculateEMI() called");
10        return "success";
11    }
12
13    @Action(value="/calculateInterestRate", results = {@Result(name = "success", location = "/error.jsp")})
14    public String calculateInterestRate() {
15        System.out.println("calculateInterestRate() called");
16        return "success";
17    }
18
19    @Action(value="/calculateIncomeTax")
20    public String calculateIncomeTax() {
21        System.out.println("calculateIncomeTax() called");
22    }
23 }
```

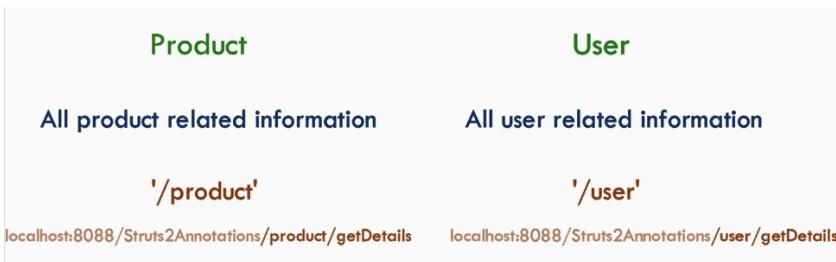
Naming space:



If we try like below. Imagine if we have 100 of such actions (difficult to remember)



The below one is called name space.



Packages are used to group actions into logical units as shown below where we have same action line number 19 and 24 .

The screenshot shows an IDE interface with several tabs open:

- `ProductDetailsAction.java`
- `UserDetailsAction.java`
- `namespace.jsp`
- `struts.xml`

The `struts.xml` file contains the following configuration:

```

8<!-- <action name="calculateEMI" class="actions.CalculateAction" method="calculateEMI">
9    <result name="success"/>success.jsp</result>
10</action>
11<action name="calculateInterestRate" class="actions.CalculateAction" method="calculateInterestRate">
12    <result name="success"/>success.jsp</result>
13</action>
14<action name="calculateIncomeTax" class="actions.CalculateAction" method="calculateIncomeTax">
15    <result name="success"/>success.jsp</result>
16</action> -->
17</package>
18<package name="product" extends="struts-default" namespace="/product">
19    <action name="getDetails" class="actions.ProductDetailsAction" >
20        <result name="success"/>namespace.jsp</result>
21    </action>
22</package>
23<package name="user" extends="struts-default" namespace="/user">
24    <action name="getDetails" class="actions.UserDetailsAction" >
25        <result name="attribute: name"/>namespace.jsp</result>
26    </action>
27</package>
28</struts>

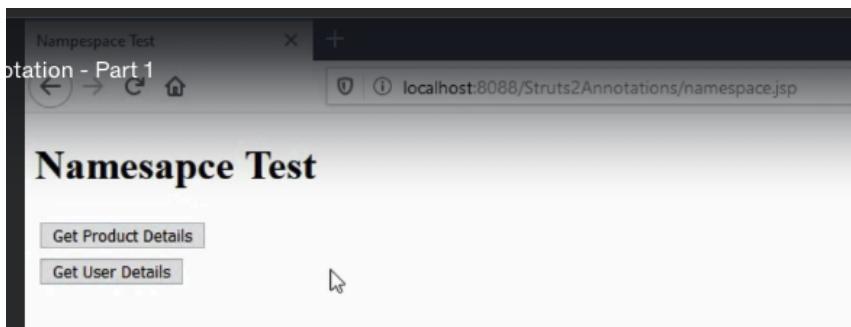
```

A tooltip is visible over the `name="getDetails"` action in the second package, showing the value `/user/getDetails`.

```

1 <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2 pageEncoding="ISO-8859-1"%>
3 <%@ taglib prefix="s" uri="/struts-tags" %>
4 <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loc
5<html>
6<head>
7 <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
8 <title>Namespace Test</title>
9 </head>
10<body>
11 <h1>Namespace Test</h1>
12 <s:form action = "/product/getDetails">
13   <s:submit value="Get Product Details"/>
14 </s:form>
15 <s:form action = "/user/getDetails">
16   <s:submit value="Get User Details"/>
17 </s:form>
18 </body>

```



Click the above link and not the messages in console.

```

1 package actions;
2
3 public class UserDetailsAction {
4
5     public String execute()
6     {
7         System.out.println("execute() method of UserDetailsAction called");
8         return "success";
9     }
10 }

```

Console output:

```

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_91\bin\java.exe (27-Jun-2020, 3:04:07 AM)
Jun 27, 2020 3:04:27 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in [17,040] milliseconds
execute() method of ProductDetailsAction called
execute() method of UserDetailsAction called

```

Lets us do the same thing with annotations. Because of the project structure namespaces will be created as shown below.

workspaceStruts2 - Struts2Annotations/src/com/testapp/actions/product/ProductDetailsAction.java - Eclipse IDE

```
1 package com.testapp.actions.product;
2
3 public class ProductDetailsAction {
4
5     public String execute()
6     {
7         System.out.println("execute() method of ProductDetailsAction called");
8         return "success";
9     }
10}
11
```

The screenshot shows the Eclipse IDE interface with the Project Explorer and Navigator tabs open. The code editor displays the `ProductDetailsAction.java` file. The code defines a single method `execute()` that prints a message to the console and returns the string "success". The file is located in the `com/testapp/actions/product` package.



workspaceStruts2 - Struts2Annotations/src/com/testapp/actions/product/ProductDetailsAction.java - Eclipse IDE

```
1 package com.testapp.actions.product;
2
3 import org.apache.struts2.convention.annotation.Action;
4 import org.apache.struts2.convention.annotation.Result;
5
6 @Action("/getDetails")
7 @Result(name="success",location="/namespace.jsp")
8 public class ProductDetailsAction {
9
10    public String execute()
11    {
12        System.out.println("execute() method of ProductDetailsAction called");
13        return "success";
14    }
15}
16
```

The screenshot shows the `ProductDetailsAction.java` code with annotations. The `@Action` annotation is used to map the action to the URL `/getDetails`. The `@Result` annotation specifies the success result name as "success" and the location as `/namespace.jsp`.

struts.xml

```
<!--
  actions
    <!-- product
      <!-- ProductDetailsAction.java
    -->
    <!-- user
      <!-- UserDetailsAction.java
        <!-- /user
    -->
      <!-- CalculateAction.java
      <!-- TestAction.java
    -->
-->
```

The screenshot shows the `UserDetailsAction.java` code with annotations. The `@Action` annotation is used to map the action to the URL `/getDetails`. The `@Result` annotation specifies the success result name as "success" and the location as `/namespace.jsp`.

The top part of the image shows a web browser window titled "Namespace Test". Inside the browser, there are two buttons: "Get Product Details" (highlighted in blue) and "Get User Details". The URL in the address bar is "localhost:8088/Struts2Annotations/namespace.jsp".

The bottom part of the image shows the Eclipse IDE's "Console" view. The logs indicate a successful server startup:

```

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (18-Jul-2020, 2:32:22 AM)
Jul 18, 2020 2:32:42 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in [17,535] milliseconds
execute() method of ProductDetailsAction called
execute() method of UserDetailsAction called

```

If namespace does not happen automatically follow the below folder structure.
Its developer interest to creating a namespace automatically or creating a custom name space.

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure including "ProductManagementApplication", "Servers", "Struts2Annotations", "struts.xml", "WebContent", "META-INF", "WEB-INF", "lib", "web.xml", "error.jsp", "namespace.jsp", "success.jsp", ".classpath", ".project", "Struts2Tags", and "Struts2Test".
- Code Editor:** Displays the code for `ProductDetailsAction.java`:

```

1 package com.testapp.product.actions;
2
3@import org.apache.struts2.convention.annotation.Action;
4
5 @Action("/getDetails")
6 @Result(name="success",location="/namespace.jsp")
7
8 public class ProductDetailsAction {
9
10    public String execute()
11    {
12        System.out.println("execute() method of ProductDetailsAction called");
13        return "success";
14    }
15 }
16

```
- Console:** Shows Tomcat startup logs:

```

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (18-Jul-2020, 2:36:13 AM)
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
Jul 18, 2020 2:36:31 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in [15,405] milliseconds
execute() method of UserDetailsAction called

```

With the above folder structure namespace wont work

5 }
6 **Using annotations, namespace can be implemented at:-**
package level
action level

Use namespace annotation as shown below.

The screenshot shows the Eclipse IDE interface with the Project Explorer and Navigator tabs open. The code editor displays a Java file named `UserDetailsAction.java` with the following content:

```

1 package com.testapp.user.actions;
2
3 import org.apache.struts2.convention.annotation.Action;
4 import org.apache.struts2.convention.annotation.Namespace;
5 import org.apache.struts2.convention.annotation.Result;
6
7 @Namespace("user")
8 @Action("getDetails")
9 @Result(name="success",location="/namespace.jsp")
10 public class UserDetailsAction {
11
12     public String execute()
13     {
14         System.out.println("execute() method of UserDetailsAction called");
15         return "success";
16     }
17 }

```

A green annotation `@Namespace("user")` is highlighted. A tooltip message is displayed below the code:

When you define a namespace using `@Namespace` annotation, it will override any existing namespace created by the framework due to the package structure

Create user and product metadata classes as shown below

The screenshot shows the Eclipse IDE interface with the Project Explorer and Navigator tabs open. The code editor displays a Java file named `UserMetadataAction.java` with the following content:

```

1 package com.testapp.user.actions;
2
3 import org.apache.struts2.convention.annotation.Action;
4
5 @Namespace("/user")
6 @Action("/getMetadata")
7 @Result(name="success",location="/namespace.jsp")
8 public class UserMetadataAction {
9
10     public String execute()
11     {
12         System.out.println("execute() method of UserMetadataAction called");
13         return "success";
14     }
15 }

```

Include something in the namespace

The screenshot shows the Eclipse IDE interface with the Project Explorer and Navigator tabs open. The code editor displays a JSP file named `namespace.jsp` with the following content:

```

10<body>
11
12 <h1>Namespace Test</h1>
13
14<s:form action = "/product/getDetails">
15     <s:submit value="Get Product Details"/>
16 </s:form>
17
18<s:form action = "/user/getDetails">
19     <s:submit value="Get User Details"/>
20 </s:form>
21
22<s:form action = "/product/getMetadata">
23     <s:submit value="Get Product Metadata"/>
24 </s:form>
25
26<s:form action = "/user/getMetadata">
27     <s:submit value="Get User Metadata"/>
28 </s:form>
29
30 </body>
31 </html>

```

The console tab at the bottom shows the output of the Tomcat server startup.

Using Action annotation we can override namespaces as shown below

The screenshot shows the Eclipse IDE interface with the following details:

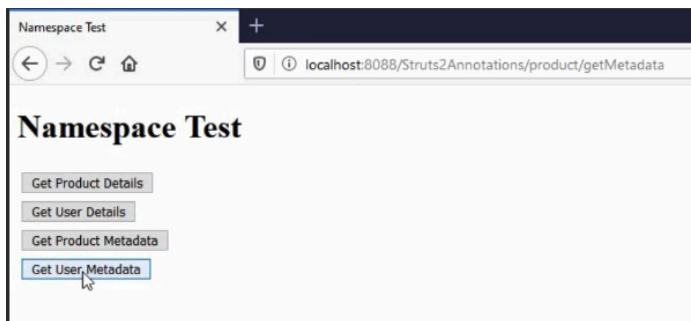
- Project Explorer:** Shows the project structure for "workspaceStruts2 - Struts2Annotations/src/com/testapp/product/actions/ProductMetadataAction.java".
- Code Editor:** Displays the Java code for `ProductMetadataAction.java`:


```

1 package com.testapp.product.actions;
2
3 import org.apache.struts2.convention.annotation.Action;
4
5 import org.apache.struts2.convention.annotation.Result;
6
7 @Action("/getMetadata")
8 @Result(name="success",location="/namespace.jsp")
9 public class ProductMetadataAction {
10
11     public String execute()
12     {
13         System.out.println("execute() method of ProductMetadataAction called");
14         return "success";
15     }
16 }
      
```
- Console:** Shows the output from Tomcat 9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (18-Jul-2020, 3:01:06 AM):


```

execute() method of ProductDetailsAction called
execute() method of UserDetailsAction called
execute() method of ProductMetadataAction called
execute() method of ProductMetadataAction called
      
```



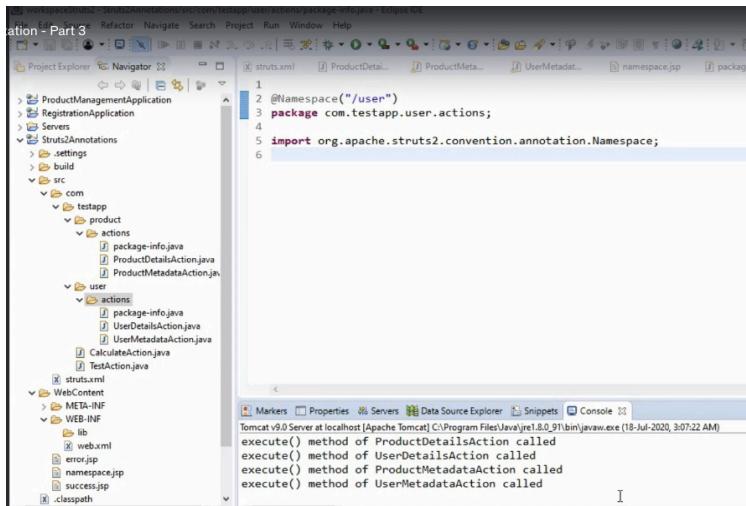
The screenshot shows the Eclipse IDE interface with the following details:

- Code Editor:** Displays the Java code for `ProductDetailsAction.java`:


```

3 import org.apache.struts2.convention.annotation.Action;
4 import org.apache.struts2.convention.annotation.Namespace;
5 import org.apache.struts2.convention.annotation.Result;
6
7 @Namespace("/product")
8 @Action("/getDetails")
9 @Result(name="success",location="/namespace.jsp")
10 public class ProductDetailsAction {
11
12     public String execute()
13     {
14         System.out.println("execute() method of ProductDetailsAction called");
15         return "success";
16     }
17 }
      
```
- Note:** A green note overlay states: "You can remove the `@Namespace` annotations from `ProductDetailsAction` and `UserDetailsAction`".

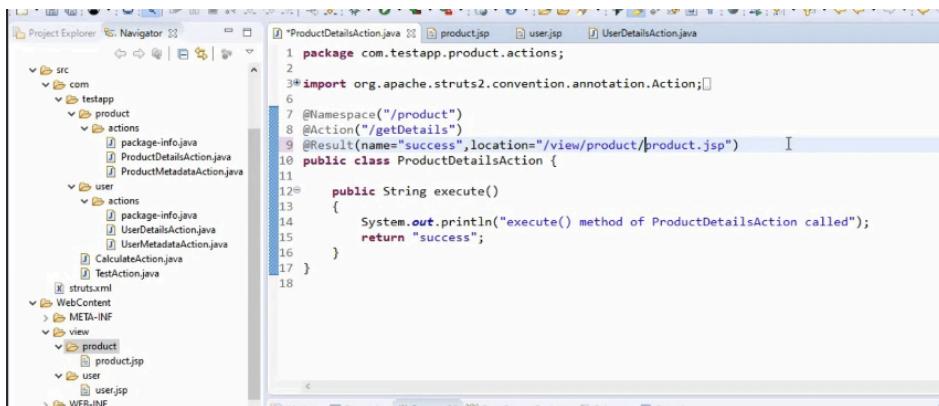
Add it in the info package-info.java classes as shown below. This is applicable at package level for action classes



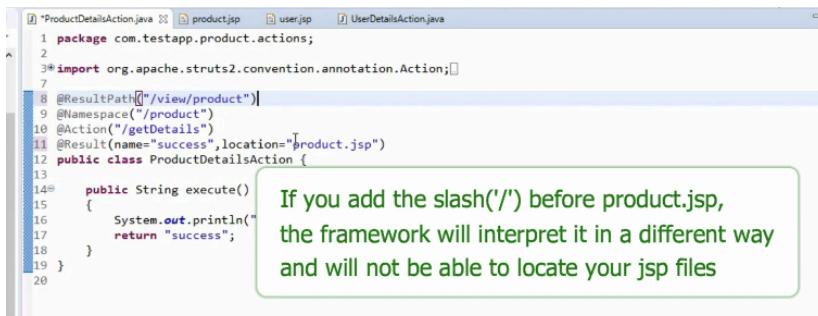
Result path annotation:

@ResultPath provides the path of the location where we will find our result pages/JSPs

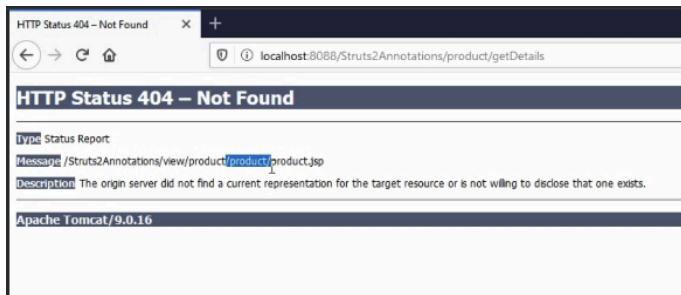
Create view folder and subfolder and place corresponding jsp pages and specify complete path at result annotation.



Another approach is use **@ResultPath**



If you up and running the application we get 404 errors because of the path it forms. Remove '/product' either from @ResultPath or @Namespace annotation. Refer below screenshot.



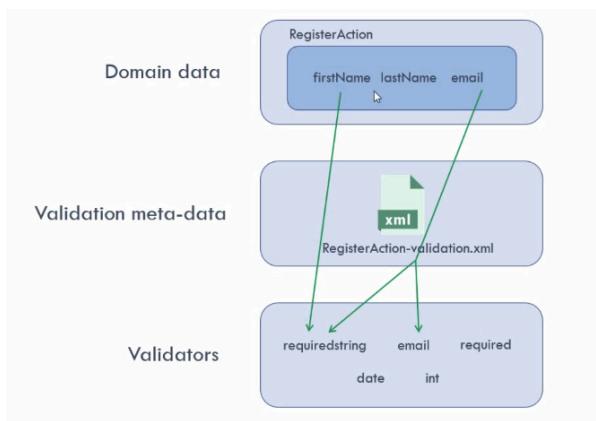
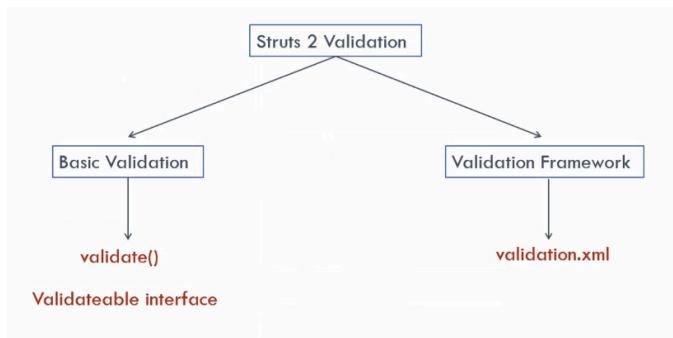
So follow the below approach. It will work for us.

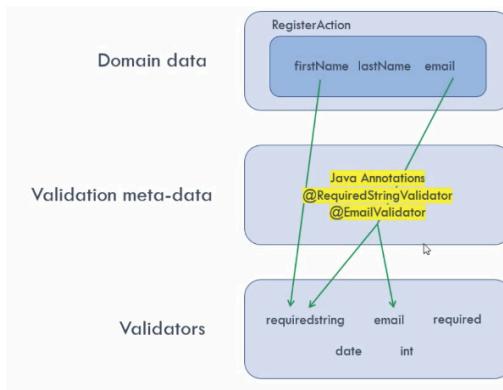
```

ProductDetailsAction.java    product.jsp    user.jsp    UserDetailsAction.java    package-info.java    package
1 package com.testapp.user.actions;
2
3 import org.apache.struts2.convention.annotation.Action;
4
5 @ResultPath("/view")
6 @Action("getDetails")
7 @Result(name="success",location="user.jsp")
8 public class UserDetailsAction {
9
10     public String execute()
11     {
12         System.out.println("execute() method of UserDetailsAction called");
13         return "success";
14     }
15 }
16
17 }
18
19

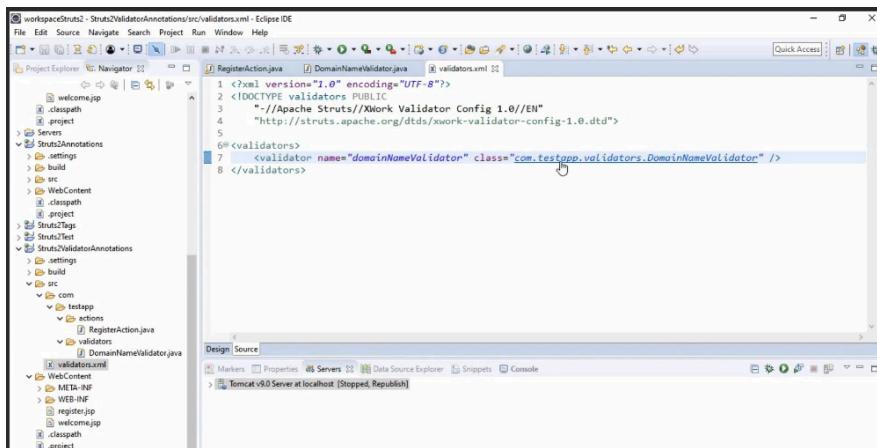
```

Validation annotations:

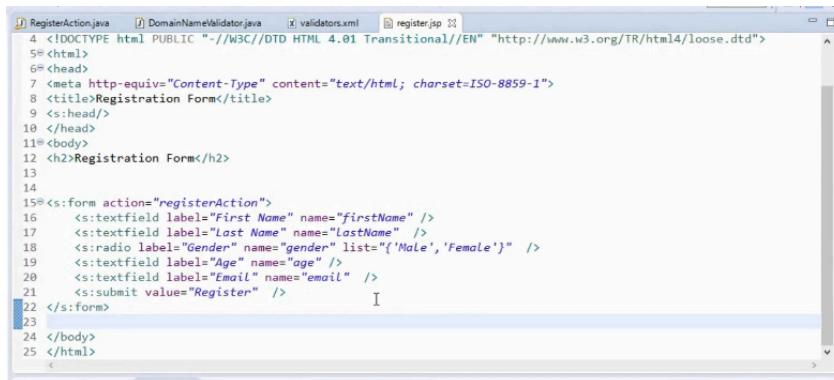




Create new project and maintain folder structure like below, copy required path and update validators.xml



And register.jsp page.



Let us implement below validations using annotations

The screenshot shows the Eclipse IDE interface with several tabs open. The active tab is 'validators.xml'. The XML code defines validation rules for fields like firstName, lastName, gender, and age.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC "-//Apache Struts//XWork Validator 1.0.3//EN"
  "http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">
<validators>
  <validator type="requiredstring">
    <param name="fieldName">firstName</param>
    <message key="error.firstName.required" />
  </validator>
  <validator type="requiredstring">
    <param name="fieldName">lastName</param>
    <message key="error.lastName.required" />
  </validator>
  <validator type="required">
    <param name="fieldName">gender</param>
    <message key="error.gender.required" />
  </validator>
  <validator type="required">
    <param name="fieldName">age</param>
    <message key="error.age.required" />
  </validator>
</validators>
```

The screenshot shows the Eclipse IDE interface with several tabs open. The active tab is 'RegisterAction-validation.xml'. This XML file contains validation rules for the 'email' field, including required string validation and email domain validation.

```
<param name="fieldName">age</param>
<message key="error.age.required" />
<validator type="int">
  <param name="fieldName">age</param>
  <param name="min">18</param>
  <message key="error.age.range" />
</validator>
<field name="email">
  <field-validator type="requiredstring">
    <message key="error.email.required" />
  </field-validator>
  <field-validator type="email">
    <message key="error.email.valid" />
  </field-validator>
  <field-validator type="domainNameValidator">
    <param name="domainName">gmail.com</param>
    <message key="error.email.validdomain" />
  </field-validator>
</field>
</validators>
```

The screenshot shows the Eclipse IDE interface with several tabs open. The active tab is 'RegisterAction.java'. The code defines getters and setters for firstName, lastName, and gender, each annotated with a required string validator.

```
32@RequiredStringValidator(type=ValidatorType.SIMPLE, fieldName="firstName", message="First name is required")
33 public String getFirstName() {
34     return firstName;
35 }
36
37@RequiredStringValidator(type=ValidatorType.SIMPLE, fieldName="lastName", message="Last name is required")
38 public String getLastname() {
39     return lastName;
40 }
41@RequiredStringValidator(type=ValidatorType.SIMPLE, fieldName="gender", message="Gender is required")
42 public String getGender() {
43     return gender;
44 }
```

The screenshot shows a Java code editor with the following code:

```
RegisterAction.java
public void setLastName(String lastName) {
    this.lastName = lastName;
}
@RequiredFieldValidator(type=ValidatorType.SIMPLE, fieldName="gender", message="Gender is required")
public String getGender() {
    return gender;
}
public void setGender(String gender) {
    this.gender = gender;
}
@RequiredFieldValidator(type=ValidatorType.SIMPLE, fieldName="age", message="Age is required")
@IntRangeFieldValidator(type=ValidatorType.SIMPLE, fieldName="age", min="18", message="Age should be above 18")
public Integer getAge() {
    return age;
}
public void setAge(Integer age) {
    this.age = age;
}
```

```
71
72= @RequiredStringValidator(type=ValidatorType.SIMPLE, fieldName="email", message="E-mail is required")
73= @EmailValidator(type=ValidatorType.SIMPLE, fieldName="email", message="Must provide a valid email")
74= @CustomValidator(type="domainNameValidator", fieldName="email", parameters = { @ValidationParameter(name=
75  public String getEmail() {
76      return email;
77  }
```

```
72=ge="E-mail is required")
73 provide a valid email")
74 = { @ValidationParameter(name="domainName",value="gmail.com")}, message="Email must have a valid domain name")
75
```

Registration Form

First name is required

First Name:

Last name is required

Last Name:

Gender is required

Gender: Male Female

Age is required

Age:

E-mail is required

Email must have a valid domain name

Email:

Try with valid data you get below page.

Welcome

First Name: Abhay
Last Name: Redkar
Gender: Male
Age: 21
Email: asd@gmail.com

Another way of doing validators is moving them above class name as shown below.

```

18 @Results({
19     @Result(name="success",location="/welcome.jsp"),
20     @Result(name="input",location="/register.jsp")
21 })
22 @Validations(
23     requiredStrings = {
24         @RequiredStringValidator(type=ValidatorType.SIMPLE, fieldName="firstName", message="First name is required"),
25         @RequiredStringValidator(type=ValidatorType.SIMPLE, fieldName="lastName", message="Last name is required"),
26         @RequiredStringValidator(type=ValidatorType.SIMPLE, fieldName="email", message="E-mail is required")
27     },
28     requiredFields = {
29         @RequiredFieldValidator(type=ValidatorType.SIMPLE, fieldName="gender", message="Gender is required"),
30         @RequiredFieldValidator(type=ValidatorType.SIMPLE, fieldName="age", message="Age is required")
31     },
32     emails = {
33         @EmailValidator(type=ValidatorType.SIMPLE, fieldName="email", message="Must provide a valid email")
34     },
35     intRangeFields = {
36         @IntRangeFieldValidator(type=ValidatorType.SIMPLE, fieldName="age", min="18", message="Age should be at least 18")
37     },
38     customValidators = {
39         @CustomValidator(type="domainNameValidator", fieldName="email", parameters = { @ValidationParameter(name="domain", value="example.com") })
40     }
41 )

```

Actions Annotation:

For both urls we get success page

```

1 package com.testapp.actions;
2
3 import org.apache.struts2.convention.annotation.Action;
4 import org.apache.struts2.convention.annotation.Actions;
5 import org.apache.struts2.convention.annotation.Result;
6
7 @Actions({
8     @Action("/testAction1"),
9     @Action("/testAction2")
10 })
11 @Result(name="success",location="/success.jsp")
12 public class MultipleURLTestAction {
13
14     public String execute() {
15         System.out.println("execute() method of MultipleURLTestAction called");
16         return "success";
17     }
18 }

```

Markers Properties Servers Data Source Explorer Snippets Console

Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_91\bin\java.exe (02-Aug-2020, 2:22:35 AM)
INFO: Server startup in [18,01] milliseconds
execute() method of MultipleURLTestAction called
execute() method of MultipleURLTestAction called

Same action executed above and below. Where above we returned same page below we returned two pages

The screenshot shows the Eclipse IDE interface with the project 'MultipleURLTest' selected in the Project Explorer. The code editor displays the file 'MultipleURLTestAction.java' containing the following Java code:

```
1 package com.testapp.actions;
2
3 import org.apache.struts2.convention.annotation.Action;
4 import org.apache.struts2.convention.annotation.Actions;
5 import org.apache.struts2.convention.annotation.Result;
6
7 @Actions({
8     @Action(value="/testAction1", results = {@Result(name="success",location="/result1.jsp")}),
9     @Action(value="/testAction2", results = {@Result(name="success",location="/result2.jsp")})
10 })
11 public class MultipleURLTestAction {
12
13     public String execute() {
14         System.out.println("execute() method of MultipleURLTestAction called");
15         return "success";
16     }
17 }
```

The Console tab at the bottom shows the output of the application running on Tomcat 9.0 Server at localhost, indicating that both 'execute()' methods were called.

The screenshot shows the same Eclipse IDE interface as above, but with a green callout box highlighting the '@Actions' annotation in the code. A green text overlay below the code states: 'You can also write the @Actions annotation above the execute() method or any other Action method as well'.

The code remains the same as in the first screenshot.