LECTURE NOTES

UNIT 4: JSP & STRUTS PROGRAMMING

- **Introduction to JSP and problem with Servelts.**
- **JSP Elements**
- **JSP Directives: page directive, include directive, taglib directive.**
- **JSP Declaration,JSP Expression, JSP Scriplets**
- **Implicit Objects**
- **Attributes: Application, request, session, page.**
- **JSP Life Cycle.**
- **Struts API**
- **Struts Requests**
- **Struts Architecture, Benefits of Struts.**
- **MVC**
- **ActionForward,ActionForm, ActionMapping, ActionServlets**
- **Struts Validation**

1.JSP (JAVA SERVER PAGES)

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc. A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

1.1.ADVANTAGE OF JSP OVER SERVLET

There are many advantages of JSP over servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

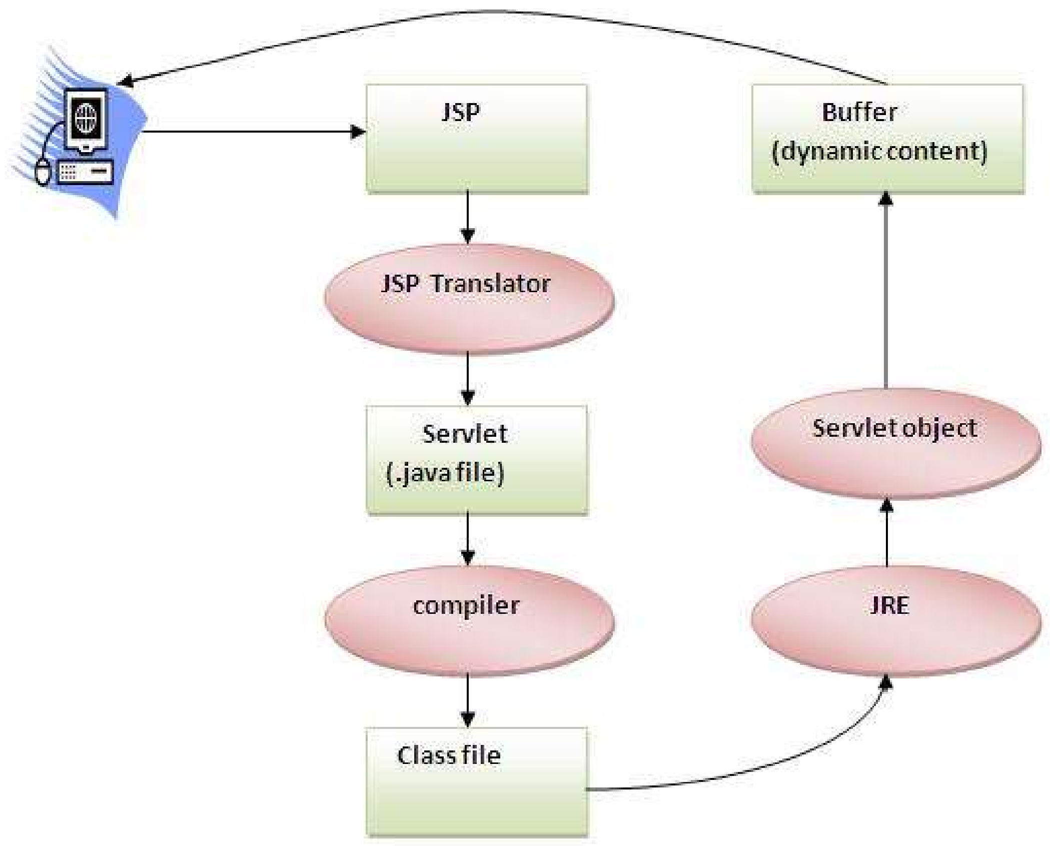
1.2.LIFE CYCLE OF A JSP PAGE

The JSP pages follows these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (class file is loaded by the classloader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).

- Reqeust processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).

Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.



As depicted in the above diagram, JSP page is translated into servlet by the help of JSP translator. The JSP translator is a part of webserver that is responsible to translate the JSP page into servlet. Afterthat Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happens in servlet is performed on JSP later like initialization, committing response to the browser and destroy.

1.4.CREATING A SIMPLE JSP PAGE

To create the first jsp page, write some html code as given below, and save it by .jsp extension. We have save this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the jsp page.

index.jsp

Let's see the simple example of JSP, here we are using the scriptlet tag to put java code in the JSP page. We will learn scriptlet tag later.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

It will print 10 on the browser.

1.4. THE JSP API

The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

JAVAX.SERVLET.JSP PACKAGE

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

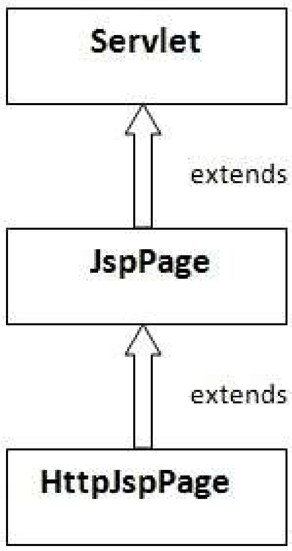
1. JspPage
2. HttpJspPage

The classes are as follows:

- JspWriter
- PageContext
- JspFactory
- JspEngineInfo
- JspException
- JspError

THE JSPPAGE INTERFACE

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.



Methods of JspPage interface

1. public void jspInit(): It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. public void jspDestroy(): It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

THE HTTPJSPPAGE INTERFACE

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

Method of HttpJspPage interface:

1. public void _jspService(): It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.

We will learn all other classes and interfaces later.

1.5.JSP SCRIPTLET TAG (SCRIPTING ELEMENTS)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag

1.5.1. JSP SCRIPTLET TAG

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

```
<%  java source code %>
```

Simple Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
```

```
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

1.5.2. JSP EXPRESSION TAG

The code placed within expression tag is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

```
<%=  statement %>
```

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>

</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

index.html

```
<html>
<body>

<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</form>
</body>
</html>
```

1.5.3. JSP DECLARATION TAG

The JSP declaration tag is used to declare fields and methods.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

```
<%!  field or method declaration %>
```

Difference between the jsp scriptlet tag and jsp declaration tag ?

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>

<%! int data=50; %>
<%= "Value of the variable is:"+data %>

```

```
</body>
</html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
<body>

<%!
int cube(int n){
return n*n*n*;
}
%>

<%= "Cube of 3 is:"+cube(3) %>

</body>
</html>
```

1.6. JSP IMPLICIT OBJECTS

There are 9 jsp implicit objects. These objects arecreated by the web container that are available to all the jsp pages.

The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

1) out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

```
PrintWriter out=response.getWriter();
```

But in JSP, you don't need to write this code.

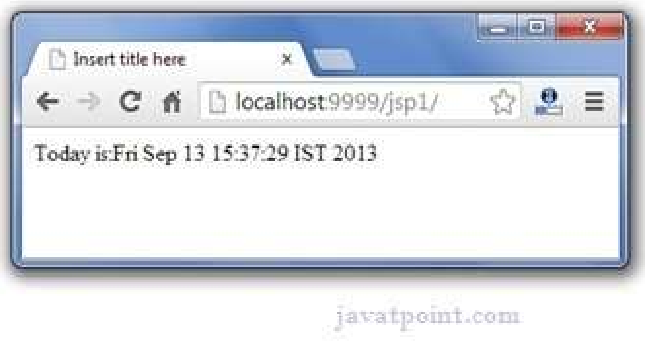
Example of out implicit object

In this example we are simply displaying date and time.

index.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Output



```
Today is:Fri Sep 13 15:37:29 IST 2013
```

1.7. JSP DIRECTIVES

The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

- page directive
- include directive
- taglib directive

Syntax of JSP Directive

```
<%@ directive attribute="value" %>
```

1.7.1. JSP PAGE DIRECTIVE

The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

```
<%@ page attribute="value" %>
```

Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

---

1)import

The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.

Example of import attribute

```
<html>
<body>

<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>

</body>
</html>
```

2)contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

Example of contentType attribute

```
<html>
<body>

<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

4)info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

Example of info attribute

```
<html>
<body>

<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

The web container will create a method getServletInfo() in the resulting servlet.For example:

```
public String getServletInfo() {
  return "composed by Sonoo Jaiswal";
}
```

5)buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

Example of buffer attribute

```
<html>
<body>

<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

6)language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

7)isELIgnored

We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later.

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

8)isThreadSafe

Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like:

```
<%@ page isThreadSafe="false" %>
```

The web container in such a case, will generate the servlet as:

```
public class SimplePage_jsp extends HttpJspBase
  implements SingleThreadModel{
.......
}
```

9)errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example of errorPage attribute

```
//index.jsp
<html>
<body>

<%@ page errorPage="myerrorpage.jsp" %>

 <%= 100/0 %>

</body>
</html>
```

---

10)isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

```
//myerrorpage.jsp
<html>
<body>
<%@ page isErrorPage="true" %>
 Sorry an exception occured!<br/>
The exception is: <%= exception %>
</body>
</html>
```

1.7.2. JSP INCLUDE DIRECTIVE

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

Advantage of Include directive

Code Reusability

Syntax of include directive

```
<%@ include file="resourceName" %>
```

Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

```
<html>
<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>
</html>
```

1.7.3. JSP TAGLIB DIRECTIVE

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

Syntax JSP Taglib directive

```
<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

Example of JSP Taglib directive

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>

<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>

<mytag:currentDate/>

</body>
</html>
```

1.8. EXCEPTION HANDLING IN JSP

The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In JSP, there are two ways to perform exception handling:

1. By errorPage and isErrorPage attributes of page directive
2. By <error-page> element in web.xml file

Example of exception handling in jsp by the elements of page directive

In this case, you must define and create a page to handle the exceptions, as in the error.jsp page. The pages where may occur exception, define the errorPage attribute of page directive, as in the process.jsp page.

There are 3 files:

- index.jsp for input values
- process.jsp for dividing the two numbers and displaying the result
- error.jsp for handling the exception

index.jsp

```
<form action="process.jsp">
No1:<input type="text" name="n1" /><br/><br/>
No1:<input type="text" name="n2" /><br/><br/>
<input type="submit" value="divide"/>
</form>
```

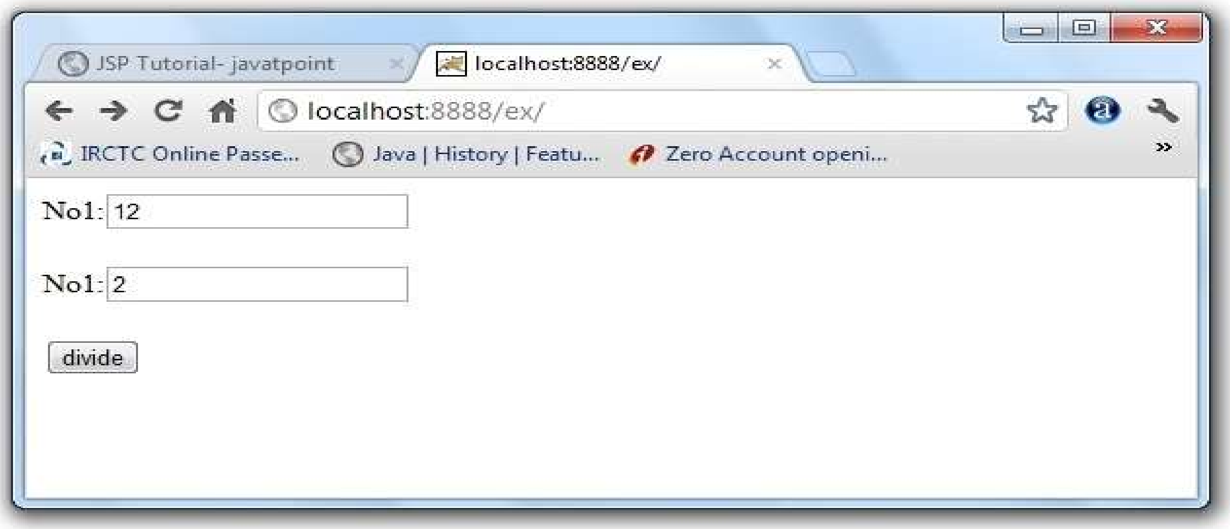
process.jsp

```
<%@ page errorPage="error.jsp" %>
<%
String num1=request.getParameter("n1");
String num2=request.getParameter("n2");
int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);
%>
```

error.jsp

```
<%@ page isErrorPage="true" %>
<h3>Sorry an exception occured!</h3>
Exception is: <%= exception %>
```

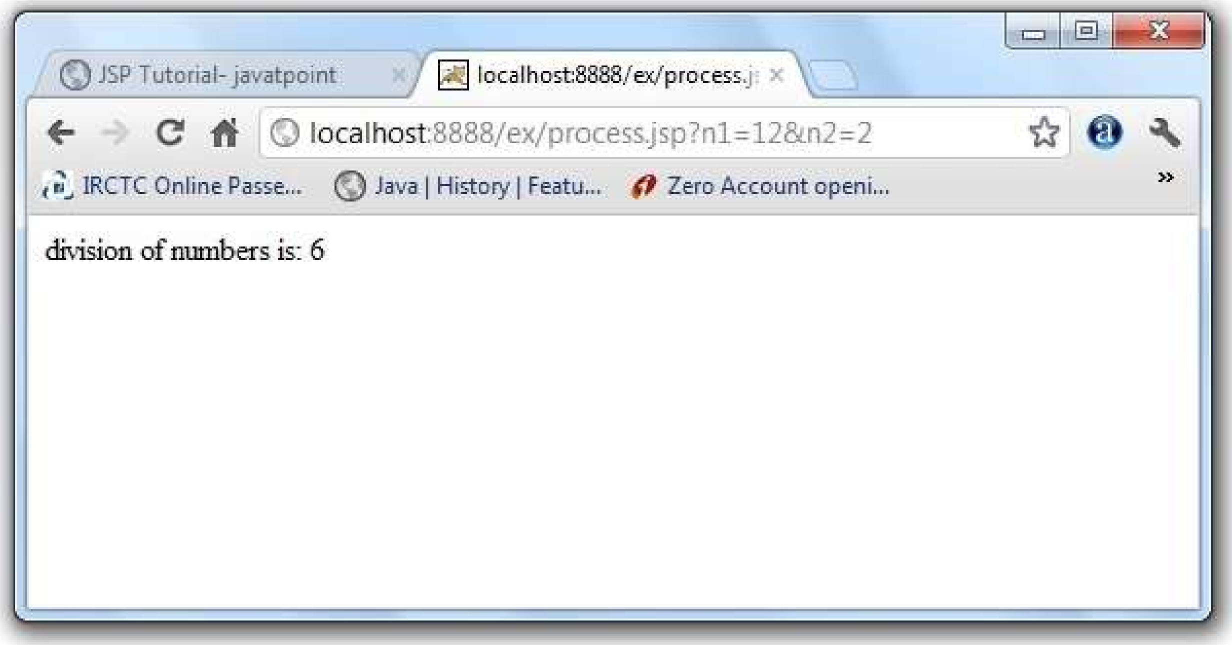
Output of this example:



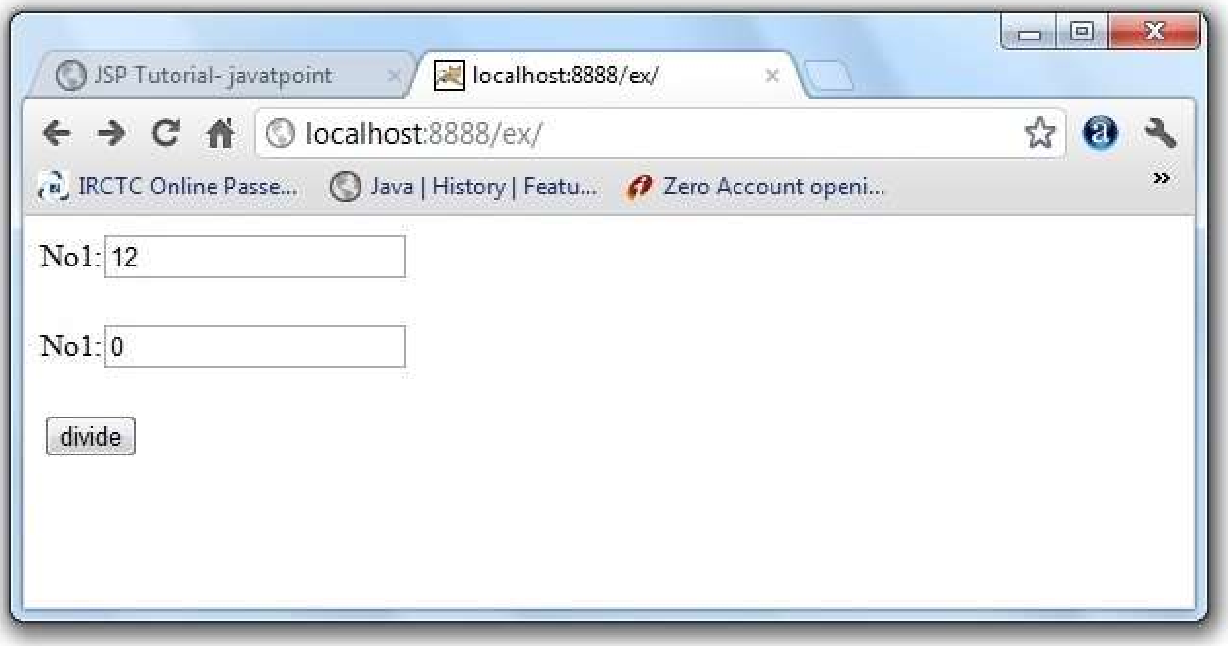
No1: 12

No1: 2

divide

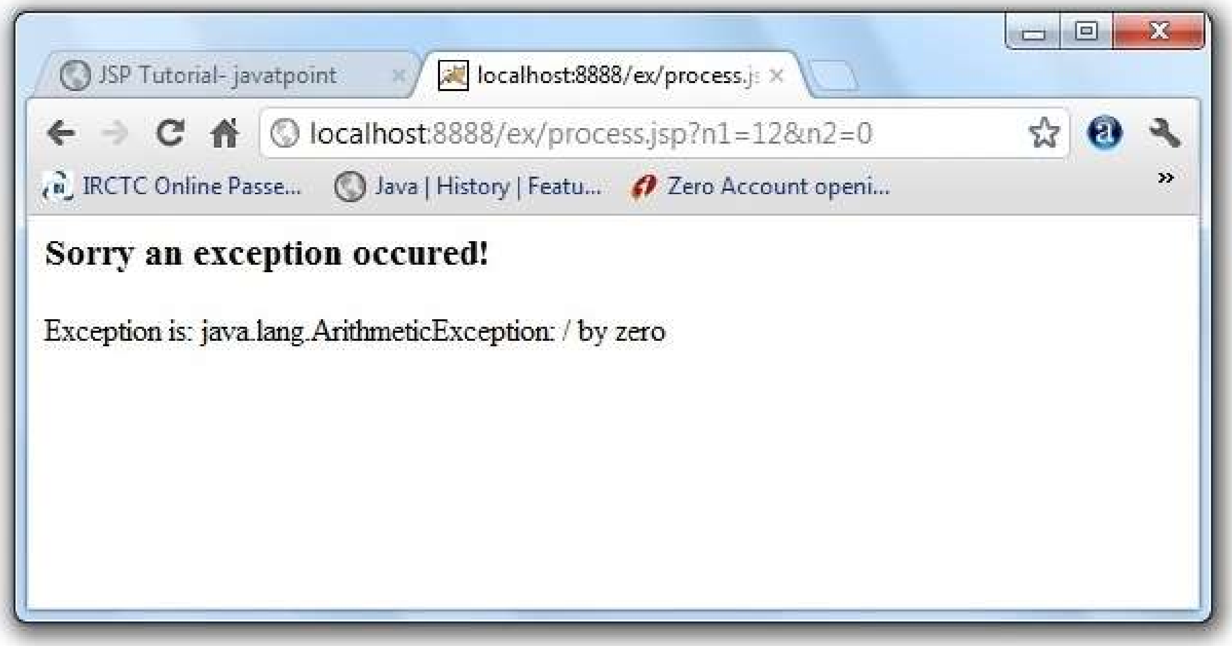

JSP Tutorial- javatpoint
localhost:8888/ex/process.j
localhost:8888/ex/process.jsp?n1=12&n2=2
IRCTC Online Passe...
Java | History | Featu...
Zero Account openi...

division of numbers is: 6



No1: 12

No1: 0


JSP Tutorial- javatpoint
localhost:8888/ex/process.j
localhost:8888/ex/process.jsp?n1=12&n2=0
IRCTC Online Passe...
Java | History | Featu...
Zero Account openi...

Sorry an exception occured!

Exception is: java.lang.ArithmeticException: / by zero

Example of exception handling in jsp by specifying the error-page element in web.xml file

This approach is better because you don't need to specify the errorPage attribute in each jsp page. Specifying the single entry in the web.xml file will handle the exception. In this case, either specify exception-type or error-code with the location element. If you want to handle all the exception, you will have to specify the java.lang.Exception in the exception-type element. Let's see the simple example:

There are 4 files:

- web.xml file for specifying the error-page element
- index.jsp for input values
- process.jsp for dividing the two numbers and displaying the result
- error.jsp for displaying the exception

1) web.xml file if you want to handle any exception

```
<web-app>

 <error-page>
  <exception-type>java.lang.Exception</exception-type>
  <location>/error.jsp</location>
  </error-page>

</web-app>
```

This approach is better if you want to handle any exception. If you know any specific error code and you want to handle that exception, specify the error-code element instead of exception-type as given below:

1) web.xml file if you want to handle the exception for a specific error code

```
<web-app>

 <error-page>
  <error-code>500</error-code>
  <location>/error.jsp</location>
  </error-page>

</web-app>
```

2) index.jsp file is same as in the above example

3) process.jsp

Now, you don't need to specify the errorPage attribute of page directive in the jsp page.

```
<%@ page errorPage="error.jsp" %>
<%

String num1=request.getParameter("n1");
String num2=request.getParameter("n2");

int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);

%>
```

1.9. JSP ACTION TAGS (ACTION ELEMENTS)

There are many JSP action tags or elements. Each tag is used to perform some specific tasks. The action tags basically are used to control the flow between pages and to use Java Bean. Jsp action tags are as follows:

- jsp:forward
- jsp:include
- jsp:useBean
- jsp:setProperty
- jsp:getProperty
- jsp:plugin
- jsp:param
- jsp:fallback

The jsp:useBean, jsp:setProperty and jsp:getProperty tags are used for bean development. So we will see these tags in bean developement.

jsp:forward action tag

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax of jsp:forward action tag without parameter

```
<jsp:forward page="relativeURL | <%= expression %>" />
```

Syntax of jsp:forward action tag with parameter

```
<jsp:forward page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:forward>
```

Example of jsp:forward action tag without parameter

In this example, we are simply forwarding the request to the printdate.jsp file.

index.jsp

```
<html>
<body>
```

```
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" />
</body>
</html>
```

printdate.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Example of jsp:forward action tag with parameter

In this example, we are forwarding the request to the printdate.jsp file with parameter and printdate.jsp file prints the parameter value with date and time.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>
<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="javatpoint.com" />
</jsp:forward>
</body>
</html>
```

printdate.jsp

```
<html>
<body>

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
<%= request.getParameter("name") %>

</body>
</html>
```

jsp:include action tag

The jsp:include action tag is used to include the content of another resource it may be jsp, html or servlet.

The jsp include action tag includes the resource at request time so it is better for dynamic pagesbecause there might be changes in future.

Advantage of jsp:include action tag

code reusability

Syntax of jsp:include action tag without parameter

```
<jsp:include page="relativeURL | <%= expression %>" />
```

Syntax of jsp:include action tag with parameter

```
<jsp:include page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:include>
```

Example of jsp:include action tag without parameter

In this example, index.jsp file includes the content of the printdate.jsp file.

File: index.jsp

```
<html>
<body>
<h2>this is index page</h2>
<jsp:include page="printdate.jsp" />
<h2>end section of index page</h2>
</body>
</html>
```

File: printdate.jsp

```
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
```

1.10. JAVA BEAN

A Java Bean is a java class that should follow following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Simple example of java bean class

```
//Employee.java
package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;
public Employee(){}

public void setId(int id){this.id=id;}

public int getId(){return id;}

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

public String getName(){return name;}

}
```

How to access the java bean class?

To access the java bean class, we should use getter and setter methods.

```
package mypack;
public class Test{
public static void main(String args[]){
Employee e=new Employee();//object is created
e.setName("Arjun");//setting value to the object
System.out.println(e.getName());
}}
```

1.10.1. JSP:USEBEAN ACTION TAG

The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

SYNTAX OF JSP:USEBEAN ACTION TAG

```
<jsp:useBean id= "instanceName" scope= "page | request | session | application"
class= "packageName.className" type= "packageName.className"
beanName="packageName.className | <%= expression >" >
</jsp:useBean>
```

Attributes and Usage of jsp:useBean action tag

1. id: is used to identify the bean in the specified scope.
2. scope: represents the scope of the bean. It may be page, request, session or application. The default scope is page.
    - page: specifies that you can use this bean within the JSP page. The default scope is page.
    - request: specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
    - session: specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
    - application: specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
3. class: instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
4. type: provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
5. beanName: instantiates the bean using the java.beans.Beans.instantiate() method.

Simple example of jsp:useBean action tag

In this example, we are simply invoking the method of the Bean class.

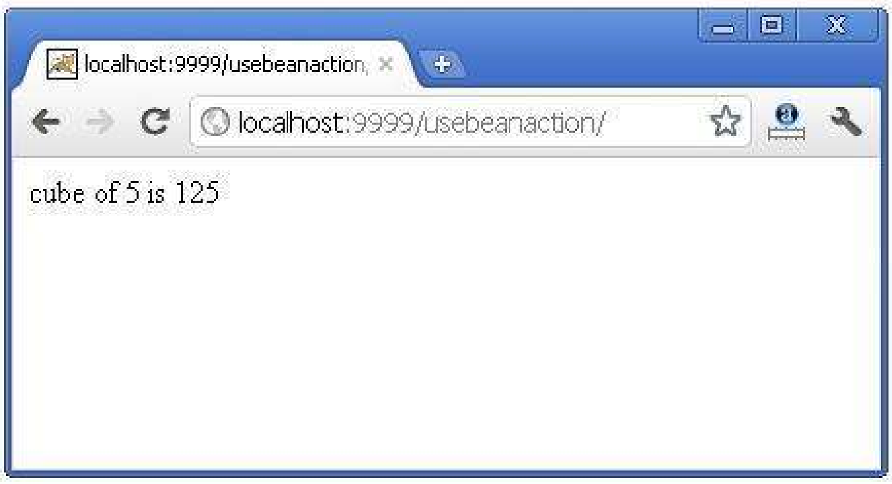
For the example of setProperty, getProperty and useBean tags, visit next page.

Calculator.java (a simple Bean class)

```java
package com.javatpoint;
public class Calculator{
public int cube(int n){return n*n*n;}
}
```

index.jsp file

```jsp
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```



```
cube of 5 is 125
```

1.11. JSTL (JSP STANDARD TAG LIBRARY)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

1. Fast Developement JSTL provides many tags that simplifies the JSP.
2. Code Reusability We can use the JSTL tags in various pages.
3. No need to use scriptlet tag It avoids the use of scriptlet tag.

There JSTL mainly provides 5 types of tags:

Tag Name	Description
core tags	The JSTL core tag provide variable support, URL management, flow control etc. The url for the core tag is http://java.sun.com/jsp/jstl/core . The prefix of core tag is c.
sql tags	The JSTL sql tags provide SQL support. The url for the sql tags ishttp://java.sun.com/jsp/jstl/sql and prefix is sql.
xml tags	The xml sql tags provide flow control, transformation etc. The url for the xml tags is http://java.sun.com/jsp/jstl/xml and prefix is x.
internationalization tags	The internationalization tags provide support for message formatting, number and date formatting etc. The url for the internationalization tags ishttp://java.sun.com/jsp/jstl/fmt and prefix is fmt.
functions tags	The functions tags provide support for string manipulation and string length. The url for the functions tags is http://java.sun.com/jsp/jstl/functions and prefix is fn.

For creating JSTL application, you need to load jstl.jar file.

1.11.1. JSTL CORE TAGS

The JSTL core tags mainly provides 4 types of tags:

- miscellaneous tags: catch and out.
- url management tags: import, redirect and url.
- variable support tags: remove and set.
- flow control tags: forEach, forTokens, if and choose.

Syntax for defining core tags

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

c:catch

It is an alternative apporach of global exception handling of JSP. It handles the exception and doesn't propagate the exception to error page. The exception object thrown at runtime is stored in a variable named var.

Example of c:catch

Let's see the simple example of c:catch.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:catch>
int a=10/0;
</c:catch>
```

c:out

It is just like JSP expression tag but it is used for exression. It renders data to the page.

Example of c:out

Let's see the simple example of c:out.

index.jsp

```
<form action="process.jsp" method="post">
FirstName:<input type="text" name="fname"/><br/>
LastName:<input type="text" name="lname"/><br/>
<input type="submit" value="submit"/>
```

```
</form>
```

process.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
First Name:<c:out value="${param.fname}"></c:out><br/>
Last Name:<c:out value="${param.lname}"></c:out>
```

c:import

It is just like jsp include but it can include the content of any resource either within server or outside the server.

Example of c:import

Let's see the simple example of c:import to display the content of other site.

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<h1>ABC.com</h1>
<hr/>
<c:import url="http://www.javatpoint.com"></c:import>
```

Example of c:import to display the source code

Let's see the simple example of c:import to display the source code of other site.

index.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<h1>ABC.com</h1>
<hr/>
<c:import var="data" url="http://www.javatpoint.com"></c:import>

<h2>Data is:</h2>
<c:out value="${data}"></c:out>
```

c:forEach

It repeats the nested body content for fixed number of times or over collection.

Example of c:forEach

Let's see the simple example of c:forEach.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:forEach var="number" begin="5" end="10">
<c:out value="${number}"></c:out>
</c:forEach>
```

---

c:if

It tests the condition.

Example of c:if

Let's see the simple example of c:if.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:set var="number" value="${200}">
<c:if test="${number<500}">
<c:out value="number is less than 500"></c:out>
</c:if>
```

---

c:redirect

It redirects the request to the given url.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<c:redirect url="http://www.javatpoint.com"></c:redirect>
```

1.11.2. CUSTOM TAGS IN JSP

Custom tags are user-defined tags. They eliminates the possibility of scriptlet tag and separates the business logic from the JSP page.

The same business logic can be used many times by the use of costom tag.

Advantages of Custom Tags

The key advantages of Custom tags are as follows:

1. Eliminates the need of srciptlet tag The custom tags eliminates the need of scriptlet tag which is considered bad programming approach in JSP.
2. Separation of business logic from JSP The custom tags separate the the business logic from the JSP page so that it may be easy to maintain.
3. Reusability The custom tags makes the possibility to reuse the same business logic again and again.

Syntax to use custom tag

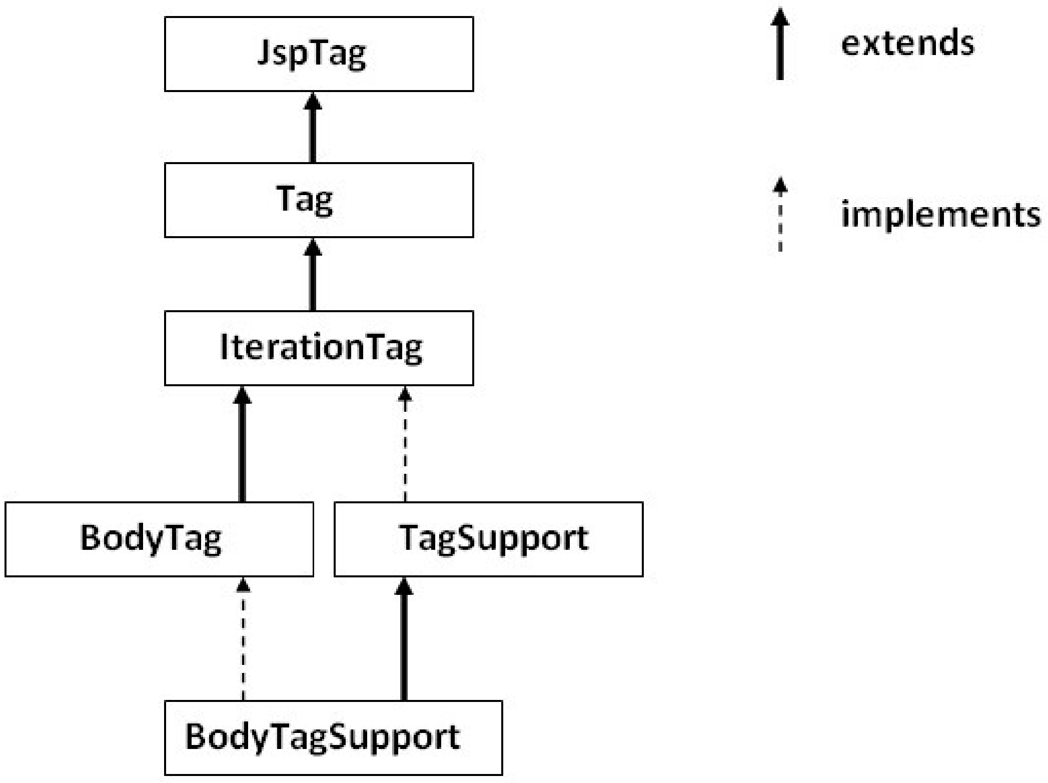
There are two ways to use the custom tag. They are given below:

```
<prefix:tagname attr1=value1....attrn=valuen />
```

```
<prefix:tagname attr1=value1....attrn=valuen >
body code
</prefix:tagname>
```

JSP Custom Tag API

The javax.servlet.jsp.tagext package contains classes and interfaces for JSP custom tag API. The JspTag is the root interface in the Custom Tag hierarchy.



JspTag interface

The JspTag is the root interface for all the interfaces and classes used in custom tag. It is a marker interface.

Tag interface

The Tag interface is the sub interface of JspTag interface. It provides methods to perform action at the start and end of the tag.

Fields of Tag interface

There are four fields defined in the Tag interface. They are:

Field Name	Description
public static int EVAL_BODY_INCLUDE	it evaluates the body content.
public static int EVAL_PAGE	it evaluates the JSP page content after the custom tag.
public static int SKIP_BODY	it skips the body content of the tag.
public static int SKIP_PAGE	it skips the JSP page content after the custom tag.

Methods of Tag interface

The methods of the Tag interface are as follows:

Method Name	Description
public void setPageContext(PageContext pc)	it sets the given PageContext object.
public void setParent(Tag t)	it sets the parent of the tag handler.
public Tag getParent()	it returns the parent tag handler object.
public int doStartTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the start of the tag.
public int doEndTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the end of the tag.
public void release()	it is invoked by the JSP page implementation object to release the state.

IterationTag interface

The IterationTag interface is the sub interface of the Tag interface. It provides an additional method to reevaluate the body.

Field of IterationTag interface

There is only one field defined in the IterationTag interface.

- public static int EVAL_BODY_AGAIN it reevaluates the body content.

Method of Tag interface

There is only one method defined in the IterationTag interface.

- public int doAfterBody()throws JspException it is invoked by the JSP page implementation object after the evaluation of the body. If this method returns EVAL_BODY_INCLUDE, body content will be reevaluated, if it returns SKIP_BODY, no more body cotent will be evaluated.

TagSupport class

The TagSupport class implements the IterationTag interface. It acts as the base class for new Tag Handlers. It provides some additional methods also.

Example of JSP Custom Tag

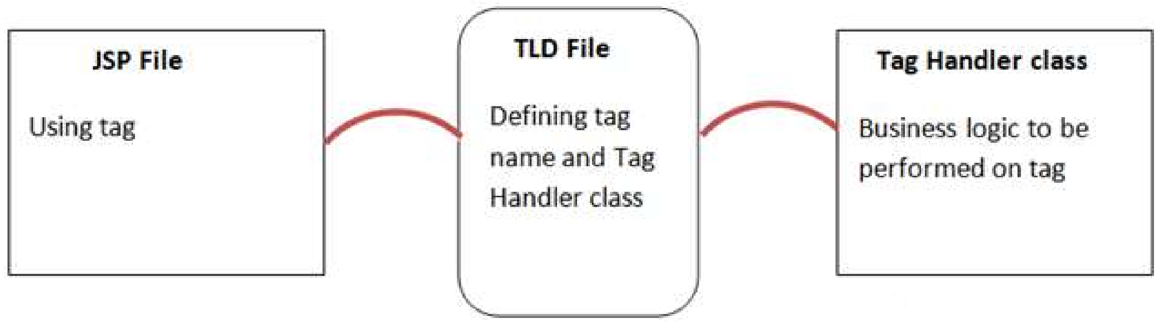
In this example, we are going to create a custom tag that prints the current date and time. We are performing action at the start of tag.

For creating any custom tag, we need to follow following steps:

1. Create the Tag handler class and perform action at the start or at the end of the tag.
2. Create the Tag Library Descriptor (TLD) file and define tags
3. Create the JSP file that uses the Custom tag defined in the TLD file

---

Understanding flow of custom tag in jsp


JSP File
Using tag
TLD File
Defining tag name and Tag Handler class
Tag Handler class
Business logic to be performed on tag

1) Create the Tag handler class

To create the Tag Handler, we are inheriting the TagSupport class and overriding its methoddoStartTag().To write data for the jsp, we need to use the JspWriter class.

The PageContext class provides getOut() method that returns the instance of JspWriter class. TagSupport class provides instance of pageContext bydefault.

File: MyTagHandler.java

```java
package com.javatpoint.sonoo;
import java.util.Calendar;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
public class MyTagHandler extends TagSupport{

public int doStartTag() throws JspException {
    JspWriter out=pageContext.getOut();//returns the instance of JspWriter
    try{
     out.print(Calendar.getInstance().getTime());//printing date and time using JspWriter
    }catch(Exception e){System.out.println(e);}
    return SKIP_BODY;//will not evaluate the body content of the tag
}
}
```

2) Create the TLD file

Tag Library Descriptor (TLD) file contains information of tag and Tag Hander classes. It must be contained inside the WEB-INF directory.

File: mytags.tld

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
```

```
     PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
   "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

 <tlib-version>1.0</tlib-version>
 <jsp-version>1.2</jsp-version>
 <short-name>simple</short-name>
 <uri>http://tomcat.apache.org/example-taglib</uri>

<tag>
<name>today</name>
<tag-class>com.javatpoint.sonoo.MyTagHandler</tag-class>
</tag>
</taglib>
```

3) Create the JSP file

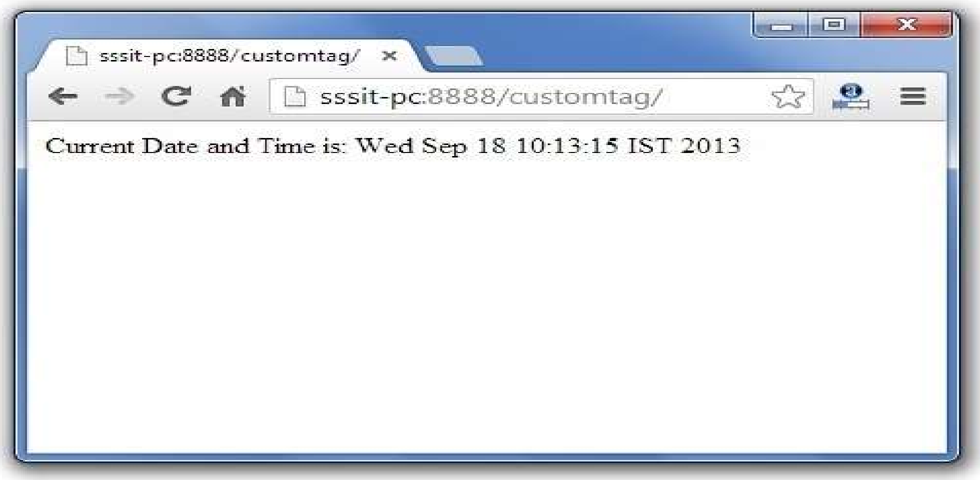
Let's use the tag in our jsp file. Here, we are specifying the path of tld file directly. But it is recommended to use the uri name instead of full path of tld file. We will learn about uri later.

It uses taglib directive to use the tags defined in the tld file.

File: index.jsp

```
<%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>
Current Date and Time is: <m:today/>
```

Output



```
Current Date and Time is: Wed Sep 18 10:13:15 IST 2013
```

1.11.3. ATTRIBUTES IN JSP CUSTOM TAG

There can be defined too many attributes for any custom tag. To define the attribute, you need to perform two tasks:

- Define the property in the TagHandler class with the attribute name and define the setter method
- define the attribute element inside the tag element in the TLD file

Example to use attribute in JSP Custom Tag

In this example, we are going to use the cube tag which return the cube of any given number. Here, we are defining the number attribute for the cube tag. We are using the three file here:

- index.jsp
- CubeNumber.java
- mytags.tld

index.jsp

```
<%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>

Cube of 4 is: <m:cube number="4"></m:cube>
```

CubeNumber.java

```
package com.javatpoint.taghandler;

import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class CubeNumber extends TagSupport{
private int number;

public void setNumber(int number) {
   this.number = number;
}

public int doStartTag() throws JspException {
   JspWriter out=pageContext.getOut();
   try{
      out.print(number*number*number);
   }catch(Exception e){e.printStackTrace();}

```

```
    return SKIP_BODY;
}
}
```

mytags.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
      "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>simple</short-name>
  <uri>http://tomcat.apache.org/example-taglib</uri>
  <description>A simple tab library for the examples</description>

  <tag>
    <name>cube</name>
    <tag-class>com.javatpoint.taghandler.CubeNumber</tag-class>
    <attribute>
    <name>number</name>
    <required>true</required>
    </attribute>
  </tag>
</taglib>
```

2. STRUTS

The struts framework is used to develop MVC-based web application.

The struts framework was initially created by Craig McClanahan and donated to Apache Foundation in May, 2000 and Struts 1.0 was released in June 2001.

The current stable release of Struts is 2.3.15.1 GA in July 16, 2013.

This struts tutorial covers all the topics of Struts Framework with simplified examples for beginners and experienced persons.

2.1. STRUTS FRAMEWORK

The Struts framework is used to develop MVC (Model View Controller) based web applications. Struts is the combination of webwork framework of opensymphony and struts 1.

```
1. struts2 = webwork + struts1
```

The Struts provides supports to POJO based actions, Validation Support, AJAX Support, Integration support to various frameworks such as Hibernate, Spring, Tiles etc, support to various result types such as Freemarker, Velocity, JSP etc.

2.2. STRUTS FEATURES TUTORIAL

Struts provides many features that were not in struts 1. The important features of struts framework are as follows:

1. Configurable MVC components
2. POJO based actions
3. AJAX support
4. Integration support
5. Various Result Types
6. Various Tag support
7. Theme and Template support

1) Configurable MVC components

In struts framework, we provide all the components (view components and action) information in struts.xml file. If we need to change any information, we can simply change it in the xml file.

2) POJO based actions

In struts, action class is POJO (Plain Old Java Object) i.e. a simple java class. Here, you are not forced to implement any interface or inherit any class.

3) AJAX support

Struts provides support to ajax technology. It is used to make asynchronous request i.e. it doesn't block the user. It sends only required field data to the server side not all. So it makes the performance fast.

4) Integration Support

We can simply integrate the struts application with hibernate, spring, tiles etc. frameworks.

5) Various Result Types

We can use JSP, freemarker, velocity etc. technologies as the result in struts.

6) Various Tag support

Struts provides various types of tags such as UI tags, Data tags, control tags etc to ease the development of struts application.

7) Theme and Template support

Struts provides three types of theme support: xhtml, simple and css_xhtml. The xhtml is default theme of struts. Themes and templates can be used for common look and feel.

2.3. MVC ARCHITECTURE

Before developing the web applications, we need to have idea about design models. There are two types of programming models (design models)

1. Model 1 Architecture
2. Model 2 (MVC) Architecture

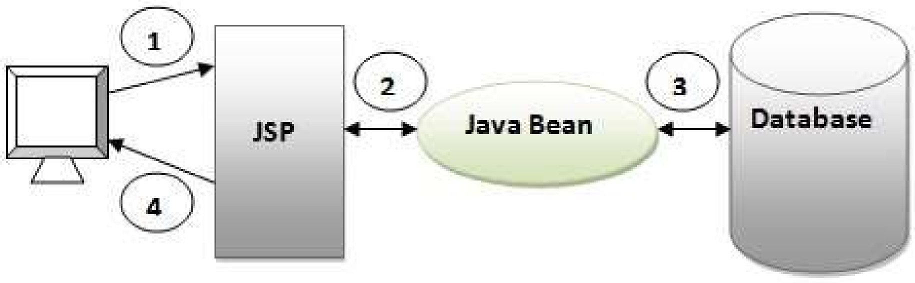
Model 1 Architecture

Servlet and JSP are the main technologies to develop the web applications.

Servlet was considered superior to CGI. Servlet technology doesn't create process, rather it creates thread to handle request. The advantage of creating thread over process is that it doesn't allocate separate memory area. Thus many subsequent requests can be easily handled by servlet.

Problem in Servlet technology Servlet needs to recompile if any designing code is modified. It doesn't provide separation of concern. Presentation and Business logic are mixed up.

JSP overcomes almost all the problems of Servlet. It provides better separation of concern, now presentation and business logic can be easily separated. You don't need to redeploy the application if JSP page is modified. JSP provides support to develop web application using JavaBean, custom tags and JSTL so that we can put the business logic separate from our JSP that will be easier to test and debug.



As you can see in the above figure, there is picture which show the flow of the model1 architecture.

1. Browser sends request for the JSP page
2. JSP accesses Java Bean and invokes business logic
3. Java Bean connects to the database and get/save data
4. Response is sent to the browser which is generated by JSP

Advantage of Model 1 Architecture

- Easy and Quick to develop web application

Disadvantage of Model 1 Architecture

- Navigation control is decentralized since every page contains the logic to determine the next page. If JSP page name is changed that is referred by other pages, we need to change it in all the pages that leads to the maintenance problem.
- Time consuming You need to spend more time to develop custom tags in JSP. So that we don't need to use scriptlet tag.

- Hard to extend It is better for small applications but not for large applications.

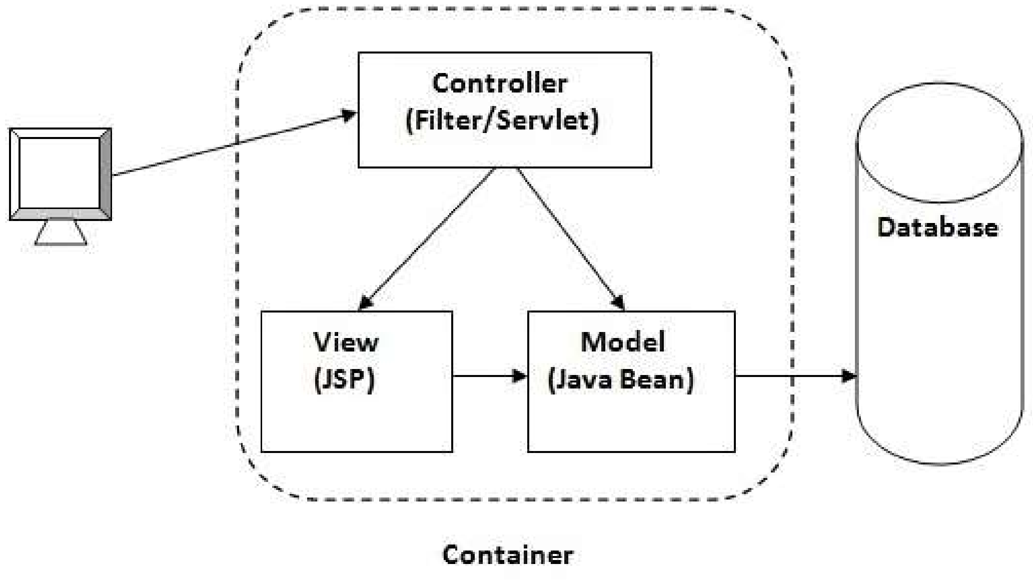
Model 2 (MVC) Architecture

Model 2 is based on the MVC (Model View Controller) design pattern. The MVC design pattern consists of three modules model, view and controller.

Model The model represents the state (data) and business logic of the application.

View The view module is responsible to display data i.e. it represents the presentation.

Controller The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.



Advantage of Model 2 (MVC) Architecture

- Navigation control is centralized Now only controller contains the logic to determine the next page.
- Easy to maintain
- Easy to extend
- Easy to test
- Better separation of concerns

Disadvantage of Model 2 (MVC) Architecture

- We need to write the controller code self. If we change the controller code, we need to recompile the class and redeploy the application.

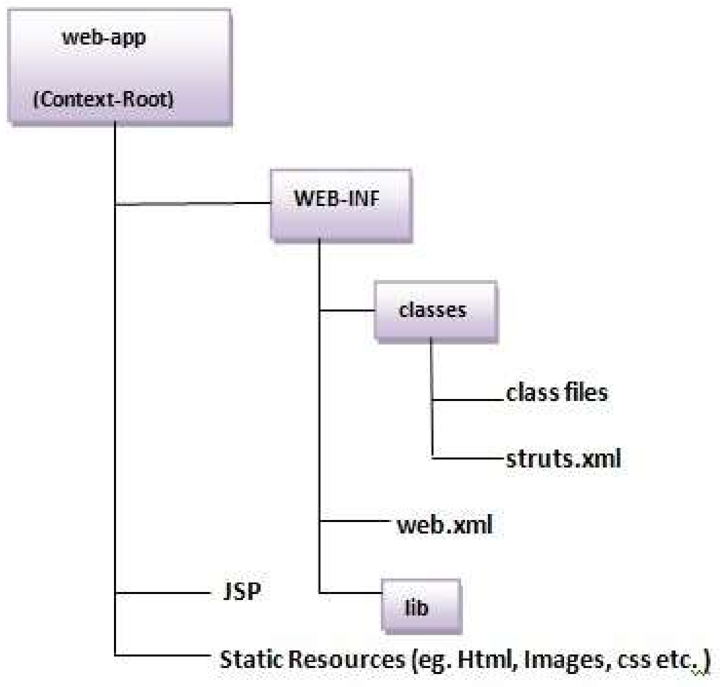
Steps to create Struts Application Example

In this example, we are creating the struts example without IDE. We can simply create the struts application by following these simple steps:

1. Create the directory structure
2. Create input page (index.jsp)
3. Provide the entry of Controller in (web.xml) file
4. Create the action class (Product.java)
5. Map the request with the action in (struts.xml) file and define the view components
6. Create view components (welcome.jsp)
7. load the jar files
8. start server and deploy the project

1) Create the directory structure

The directory structure of struts is same as servlet/JSP. Here, struts.xml file must be located in the classes folder.



2) Create input page (index.jsp)

This jsp page creates a form using struts UI tags. To use the struts UI tags, you need to specify uri /struts-tags. Here, we have used s:form to create a form, s:textfield to create a text field, s:submit to create a submit button.

index.jsp

```
<%@ taglib uri="/struts-tags" prefix="s" %>
<s:form action="product">
<s:textfield name="id" label="Product Id"></s:textfield>
<s:textfield name="name" label="Product Name"></s:textfield>
<s:textfield name="price" label="Product Price"></s:textfield>
<s:submit value="save"></s:submit>
</s:form>
```

3) Provide the entry of Controller in (web.xml) file

In struts, StrutsPrepareAndExecuteFilter class works as the controller. As we know well, struts uses filter for the controller. It is implicitly provided by the struts framework.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <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>
</web-app>
```

4) Create the action class (Product.java)

This is simple bean class. In struts, action is POJO (Plain Old Java Object). It has one extra methodexecute i.e. invoked by struts framework by default.

Product.java

```
package com.javatpoint;

public class Product {
```

```
private int id;
private String name;
private float price;
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public float getPrice() {
    return price;
}
public void setPrice(float price) {
    this.price = price;
}

public String execute(){
    return "success";
}
}
```

5) Map the request in (struts.xml) file and define the view components

It is the important file from where struts framework gets information about the action and decides which result to be invoked. Here, we have used many elements such as struts, package, action and result.

struts element is the root elements of this file. It represents an application.

package element is the sub element of struts. It represents a module of the application. It generally extends the struts-default package where many interceptors and result types are defined.

action element is the sub element of package. It represents an action to be invoked for the incoming request. It has name, class and method attributes. If you don't specify name attribute by default execute() method will be invoked for the specified action class.

result element is the sub element of action. It represents an view (result) that will be invoked. Struts framework checks the string returned by the action class, if it returns success, result page for the action is invoked whose name is success or has no name. It has name and type attributes. Both are optional. If you don't specify the result name, by default success is assumed as the result name. If you don't specify the type attribute, by default dispatcher is considered as the default result type. We will learn about result types later.

struts.xml

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

<action name="product" class="com.javatpoint.Product">
<result name="success">welcome.jsp</result>
</action>

</package>
</struts>
```

6) Create view components (welcome.jsp)

It is the view component the displays information of the action. Here, we are using struts tags to get the information.

The s:property tag returns the value for the given name, stored in the action object.

welcome.jsp

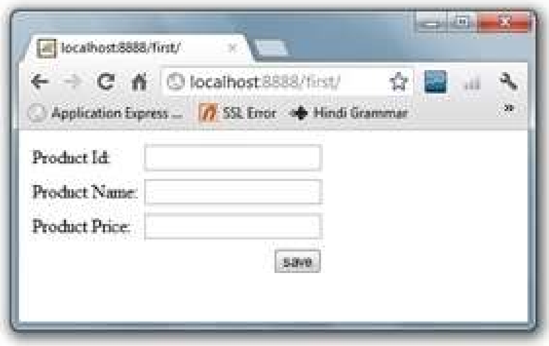
```
<%@ taglib uri="/struts-tags" prefix="s" %>

Product Id:<s:property value="id"/><br/>
Product Name:<s:property value="name"/><br/>
Product Price:<s:property value="price"/><br/>
```

7) Load the jar files

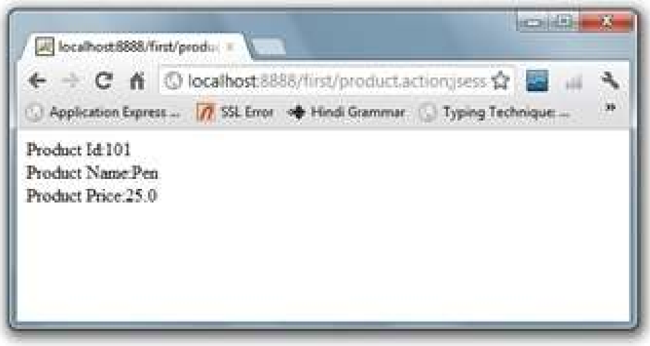
To run this application, you need to have the struts jar files. Here, we are providing all the necessary jar files for struts. Download it and put these jar files in the lib folder of your project.

8) start server and deploy the project

Finally, start the server and deploy the project and access it.



Product Id:
Product Name:
Product Price:



Product Id:101
Product Name:Pen
Product Price:25.0

2.4. STRUTS VALIDATION

To avoid the wrong values, we need to perform validation on forms where user submits some values. For example, if user writes his/her email id as abc, we need to give error message to the user that the given email id is not correct. So that we can have only valuable informations.

There are three ways to perform validation in struts.

1) By Custom Validation Here, we must implement the Validateable interface (or extend ActionSupport class) and provide the implementation of validate method.

2) By Input Validation (built-in validators) Struts provides a lot of predefined that can be used in struts application to perform validation.

Struts provides following bundled validators.

- requiredstring validator
- stringlength validator
- email validator
- date validator
- int validator

- double validator
- url validator
- regex validator

3) By Ajax Validation (built-in validators with ajax) If we don't want to refresh the page, we can use jsonValidation interceptor to perform validation with ajax.

Struts Custom Validation - Workflow Interceptor

We can define our own validation logic (custom validation) in struts by implementing theValidateable interface in the action class.

The workflow interceptor is used to get information about the error messages defined in the action class.

Workflow Interceptor

The workflow interceptor checks if there is any validation errors or not. It doesn't perform any validation.

It is applied when action class implements the Validateable interface. The input is the default parameter for this interceptor that determines the result to be invoked for the action or field error.

It is found in the defaultStack so we don't need to define it explicitly.

Parameters of workflow interceptor

There is only 1 parameter defined for workflow interceptor.

Parameter	Description
inputResultName	specifies the result name to be returned if field error or action error is found. It is set to input bydefault.

Validateabale interface

The Validateable interface must be implemented to perform validation logic in the action class. It contains only one method validate() that must be overridden in the action class to define the validation logic. Signature of the validate method is:

```
public void validate();
```

ValidationAware interface

The ValidationAware interface can accept the field level or action class level error messages. The field level messages are kept in Map and Action class level messages are kept in collection. It should be implemented by the action class to add any error message.

Methods of ValidatationAware interface

The methods of ValidationAware interface are as follows:

Method	Description
void addFieldError(String fieldName,String errorMessage)	adds the error message for the specified field.
void addActionError(String errorMessage)	adds an Action-level error message for this action.
void addActionMessage(String message)	adds an Action-level message for this action.
void setFieldErrors(Map<String,List<String>> map)	sets a collection of error messages for fields.
void setActionErrors(Collection<String> errorMessages)	sets a collection of error messages for this action.
void setActionMessages(Collection<String> messages)	sets a collection of messages for this action.
boolean hasErrors()	checks if there are any field or action errors.
boolean hasFieldErrors()	checks if there are any field errors.
boolean hasActionErrors()	checks if there are any Action-level error messages.
boolean hasActionMessages()	checks if there are any Action-level messages.
Map<String,List<String>> getFieldErrors()	returns all the field level error messages.

Collection<String> getActionErrors()	returns all the Action-level error messages.
Collection<String> getActionMessages()	returns all the Action-level messages.

Note: ActionSupport class implements Validateable and ValidationAware interfaces, so we can inherit the ActionSupport class to define the validation logic and error messages.

Steps to perform custom validation

The steps are as follows:

1. create the form to get input from the user
2. Define the validation logic in action class by extending the ActionSupport class and overriding the validate method
3. Define result for the error message by the name input in struts.xml file

Example to perform custom validation

In this example, we are creating 4 pages :

1. index.jsp for input from the user.
2. RegisterAction.java for defining the validation logic.
3. struts.xml for defining the result and action.
4. welcome.jsp for the view component.

1) Create index.jsp for input

This jsp page creates a form using struts UI tags. It receives name, password and email id from the user.

index.jsp
```
<%@ taglib uri="/struts-tags" prefix="s" %>
<s:form action="register">
<s:textfield name="name" label="Name"></s:textfield>
<s:password name="password" label="Password"></s:password>
<s:submit value="register"></s:submit>
</s:form>
```

2) Create the action class

This action class inherits the ActionSupport class and overrides the validate method to define the validation logic.

RegisterAction.java

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

public class RegisterAction extends ActionSupport{
private String name,password;
public void validate() {
    if(name.length()<1)
        addFieldError("name","Name can't be blank");
    if(password.length()<6)
        addFieldError("password","Password must be greater than 5");
}

//getters and setters

public String execute(){
//perform business logic here
    return "success";
}
}
```

3) Define a input result in struts.xml

This xml file defines an extra result by the name input, that will be invoked if any error message is found in the action class.

struts.xml

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

<package name="default" extends="struts-default">
<action name="register" class="com.javatpoint.RegisterAction">
<result>welcome.jsp</result>
<result name="input">index.jsp</result>
</action>
</package>
```

```
12. </struts>
```

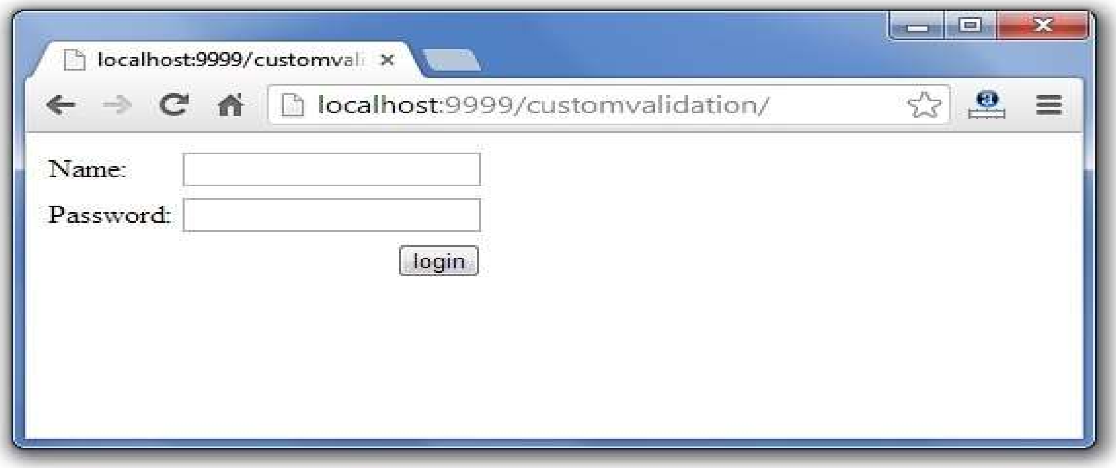
4) Create view component

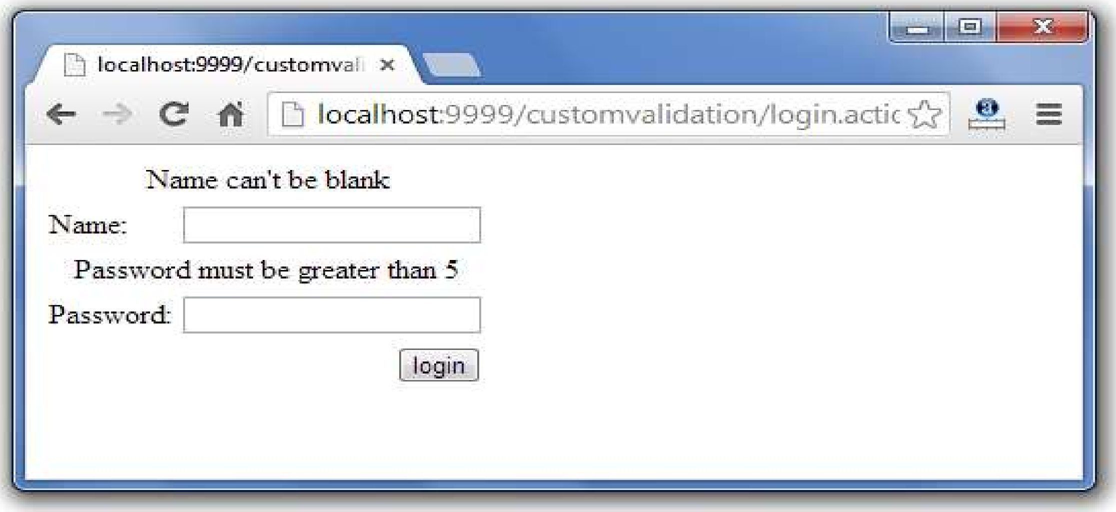
It is the simple jsp file displaying the information of the user.

welcome.jsp

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

Output





Name can't be blank
Name:
Password must be greater than 5
Password:

Defining action level error message

The action level error message works for the whole form. We can define the action level error message byaddActionError() method of ValidationAware interface in validate() method. For example:

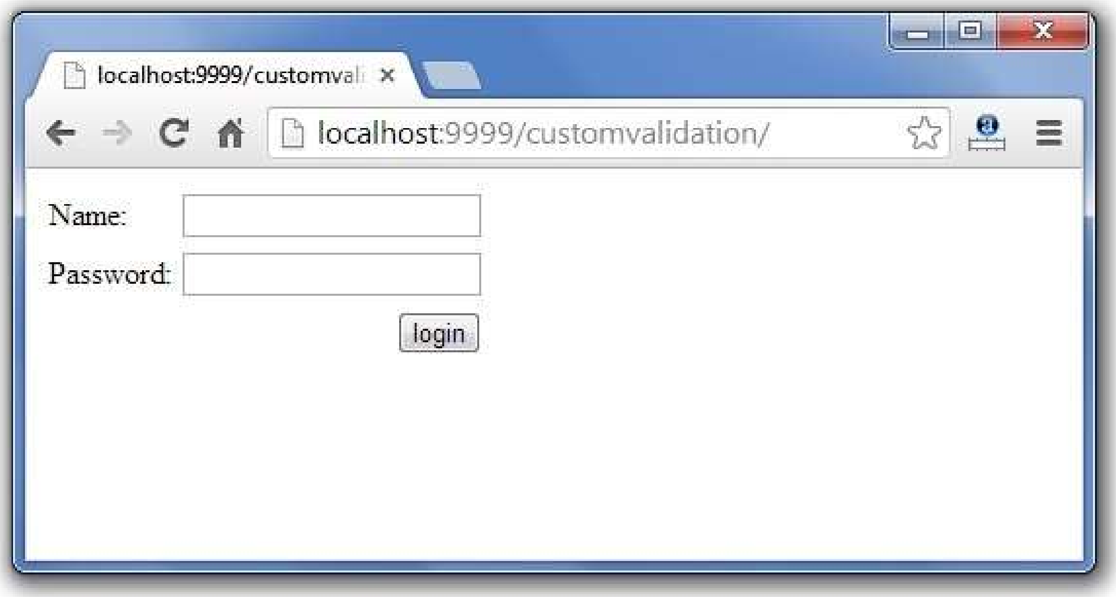
```java
package com.javatpoint;
import com.opensymphony.xwork2.ActionSupport;
public class RegisterAction extends ActionSupport{
private String name,password,email;
public void validate() {
    if(name.trim().length()<1 || password.trim().length()<1){
        addActionError("Fields can't be blank");
    }
}

//getters and setters

public String execute(){
    return "success";
}
}
```

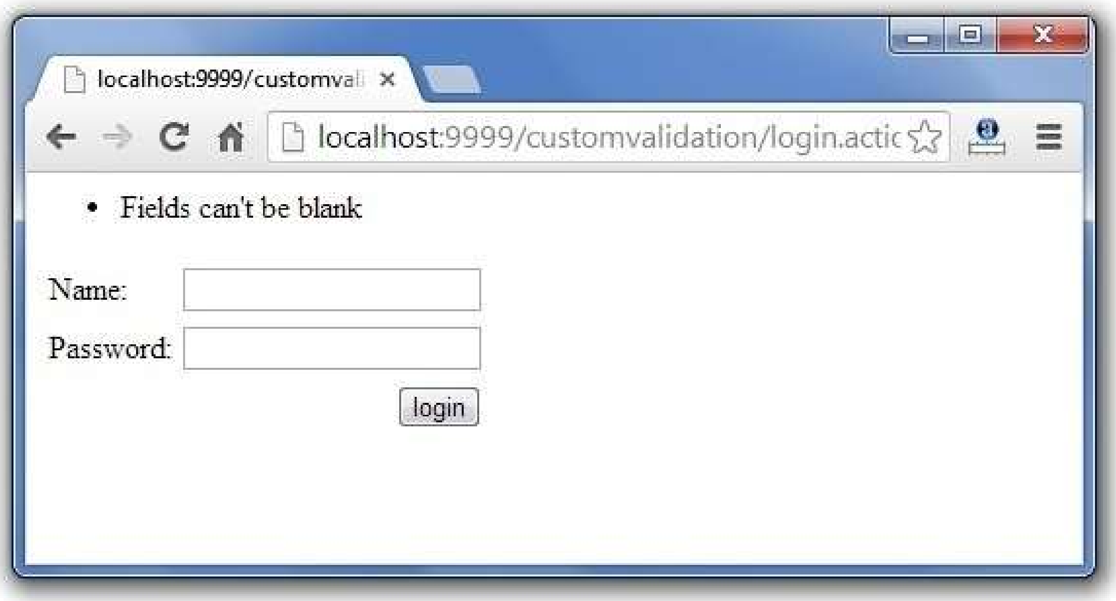
Now you need to use actionerror tag in index.jsp file to display the action level error message.

index.jsp

```jsp
<%@ taglib uri="/struts-tags" prefix="s" %>
<s:actionerror/>
<s:form action="register">
<s:textfield name="name" label="Name"></s:textfield>
<s:password name="password" label="Password"></s:password>
<s:textfield name="email" label="Email Id"></s:textfield>
<s:submit value="register"></s:submit>
</s:form>
```

Output




localhost:9999/customvali
localhost:9999/customvalidation/login.actic

- Fields can't be blank

2.5. STRUTS VALIDATION BY BUNDLED VALIDATORS

Struts validation framework provides many built-in validators also known as bundled validators for email, string, int, double, url, date etc

So, we don't need to provide explicit logic for email, double, url etc. For providing specific validation logic, we can use regex, which we will see later.

validation interceptor

It performs validation against the specified validation rules and adds the field-level and action-level error messages.

It works in conjunction with the workflow interceptor to display the error messages.

There is no parameter defined for this interceptor.

Advantage of Bundled validators

fast development because we don't need to specify the common validators such as email date, string length etc.

Bundled validators

Struts provides following bundled validators.

- requiredstring validator
- stringlength validator
- email validator
- date validator
- int validator
- double validator
- url validator
- regex validator

Two ways to use bundled validators

There are two ways to use bundled validators:

1. Plain-Validator (non-field validator) Syntax
2. Field-Validator Syntax

In the next page we will see the full example of bundled validators. Let's now understand the difference between plain-validator syntax and field-validator syntax.

Plain-Validator (non-field validator) Syntax

Plain-validator syntax can be used for action level validator. In such case, a single validator can be applied to many fields.

But disadvantage of this approach is we can't apply many validators to single field.

Let's see simple example of plain-validator.

```
<validators>
  <!-- Plain-Validator Syntax -->
  <validator type="requiredstring">
    <param name="fieldName">username</param>
    <param name="trim">true</param>
    <message>username is required</message>
  </validator>

</validators>
```

Field-Validator Syntax

The field-validator syntax can be used for field level validator. In such case, multiple validator can be applied to one field. For example, we can apply required and email validators on email field. Moreover, each field can display different messages.

But disadvantage of this approach is we can't apply common validator to many field like plain-validator.

Let's see simple example of field-validator.

```
<validators>
  <!-- Field-Validator Syntax -->
```

```
    <field name="username">
        <field-validator type="requiredstring">
          <param name="trim">true</param>
          <message>username is required</message>
      </field-validator>
    </field>

</validators>
```