**JSP Tutorial**

**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 Tags, etc.

**Advantages of JSP over Servlet**

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

**1) Extension to Servlet**

JSP technology is the extension to Servlet technology. We can use all the features of the 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 many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

**The Lifecycle of a JSP Page**

The JSP pages follow these phases:

* Translation of JSP Page
* Compilation of JSP Page
* Classloading (the classloader loads class file)
* Instantiation (Object of the Generated Servlet is created).
* Initialization ( the container invokes jspInit() method).
* Request processing ( the container invokes \_jspService() method).
* Destroy ( the container invokes jspDestroy() method).

**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 the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

**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 saved 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 where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

1. <html>
2. <body>
3. <% out.print(2\*5); %>
4. </body>
5. </html>

It will print **10** on the browser.

**How to run a simple JSP Page?**

Follow the following steps to execute this JSP page:

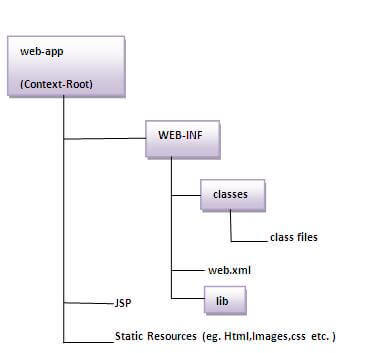
* Start the server
* Put the JSP file in a folder and deploy on the server
* Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

**Do I need to follow the directory structure to run a simple JSP?**

No, there is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

**The Directory structure of JSP**

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



# 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:

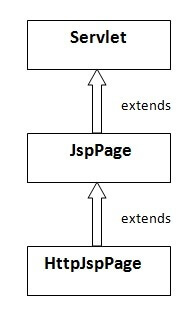
1. JspPage
2. HttpJspPage

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:

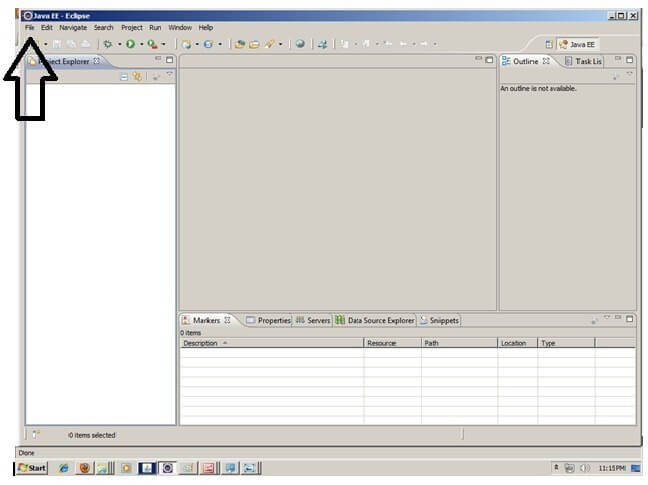
1. **public void \_jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore \_ signifies that you cannot override this method.

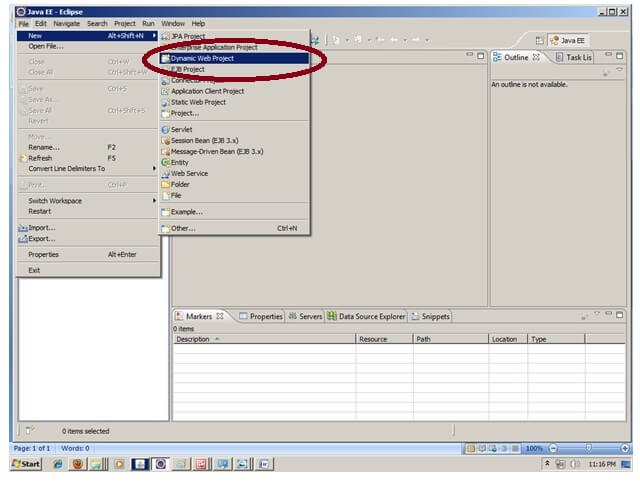
**Creating JSP in Eclipse IDE with Tomcat server**

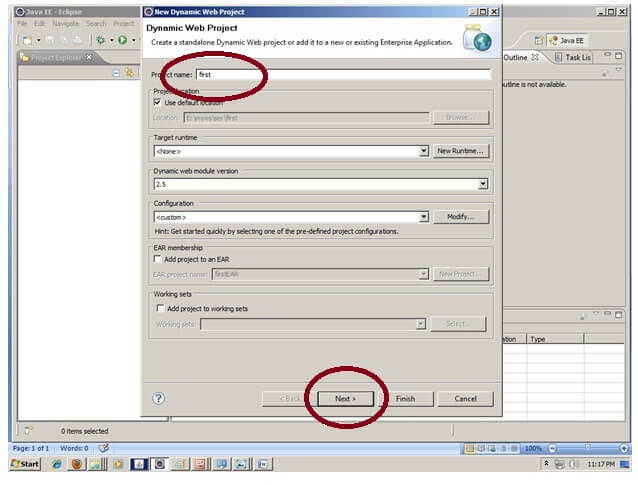
* Create a Dynamic web project
* create a jsp
* start tomcat server and deploy the project

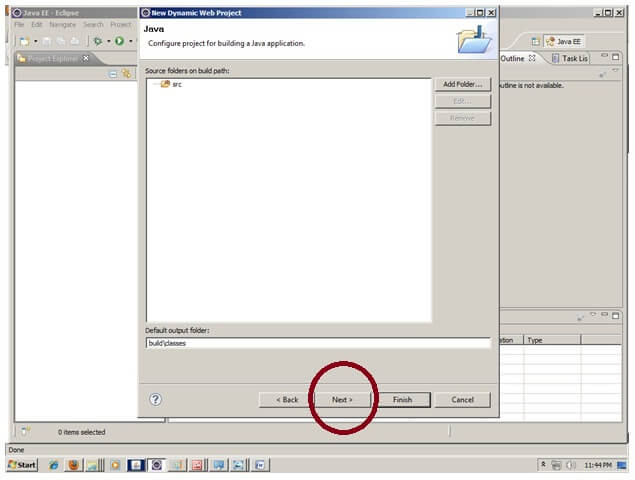
**1) Create the dynamic web project**

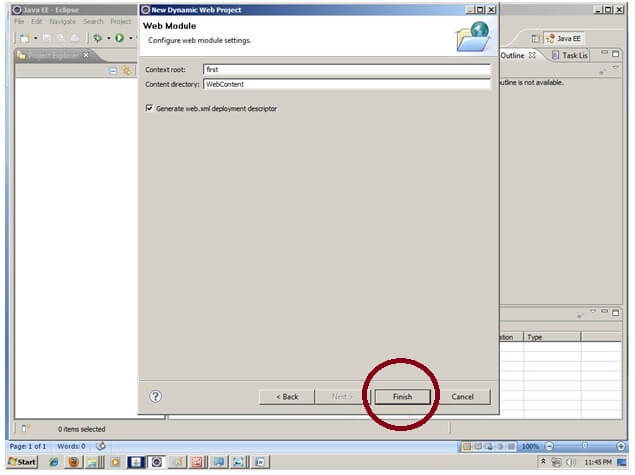
For creating a dynamic web project click on File Menu -> New -> dynamic web project -> write your project name e.g. first -> Finish.

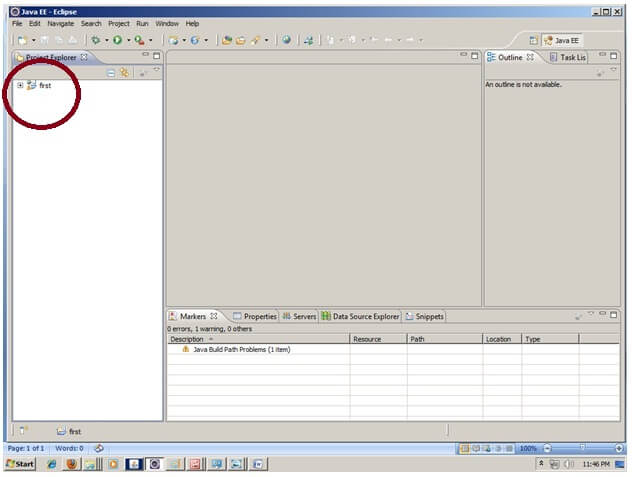






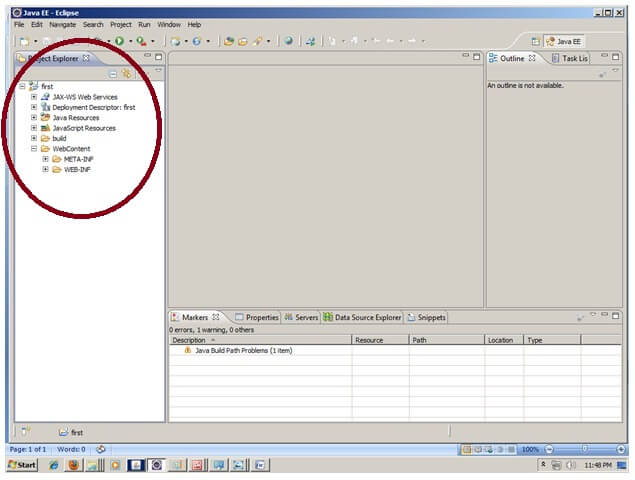


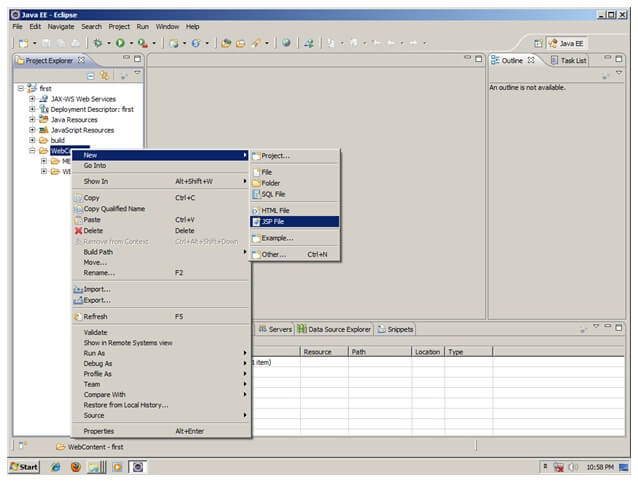


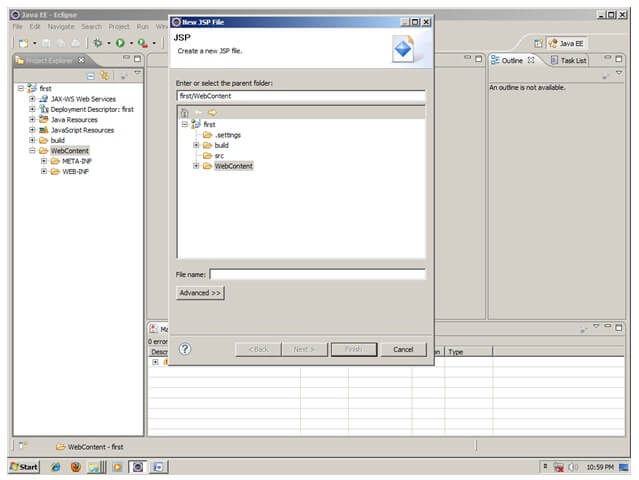


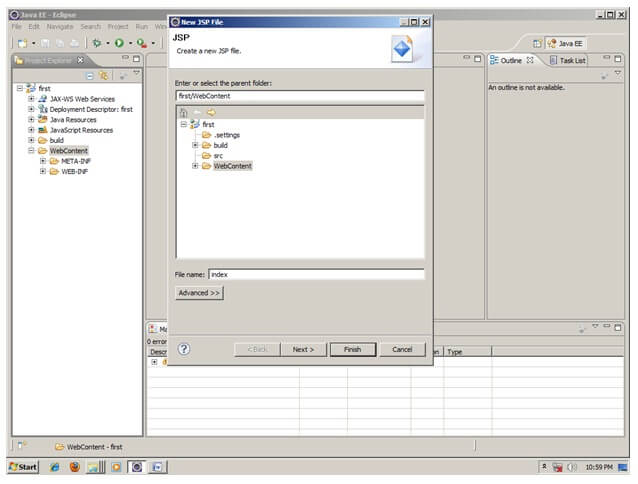
**2) Create the JSP file in eclipse IDE**

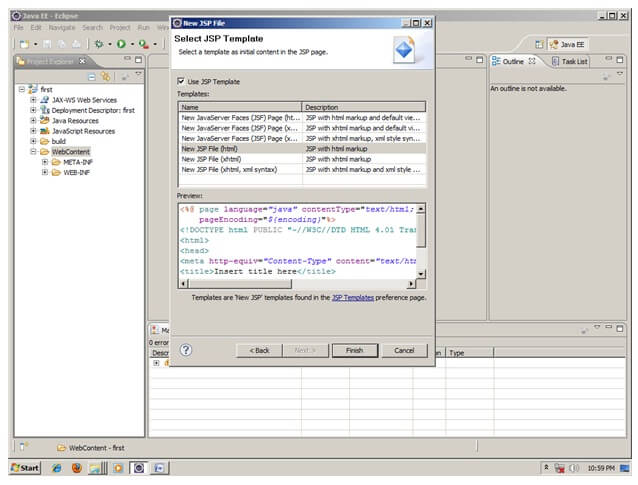
For creating a jsp file explore the project by clicking the + icon -> right click on WebContent -> New -> jsp -> write your jsp file name e.g. index -> next -> Finish.

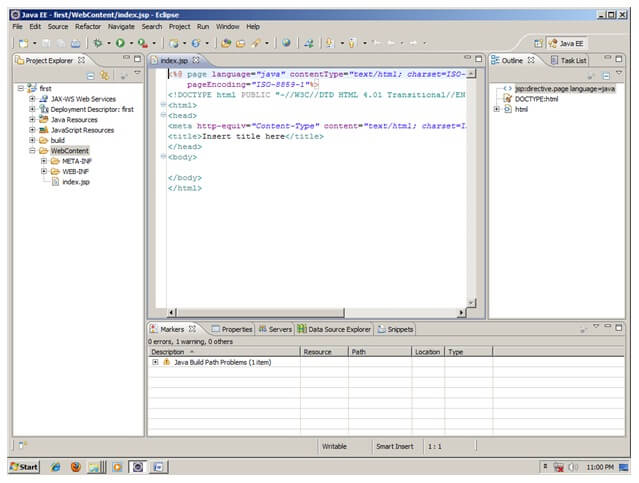




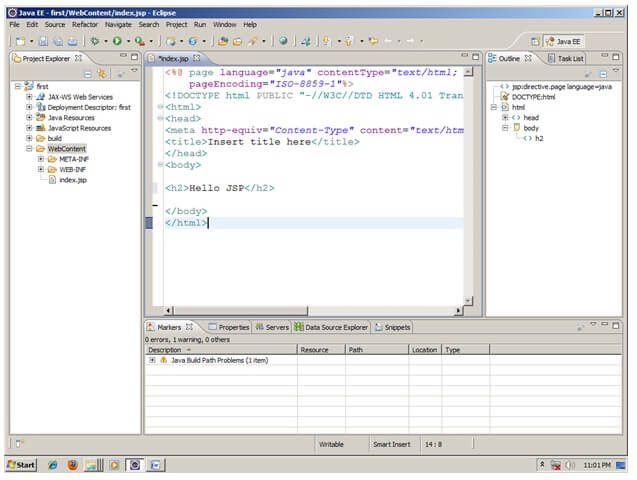








Now JSP file is created, let's write some code.



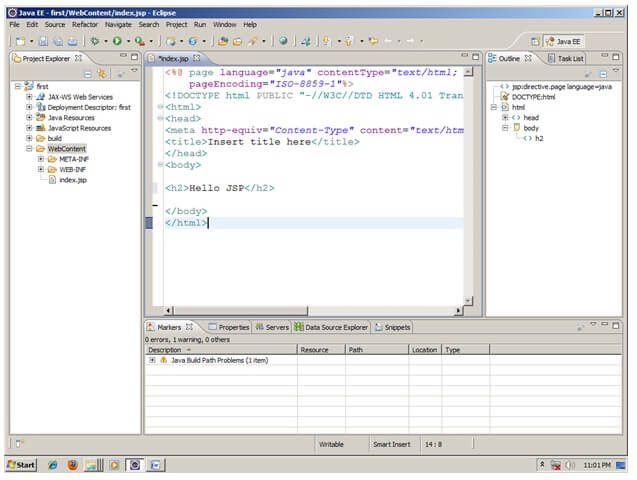
**3) Start the server and deploy the project:**

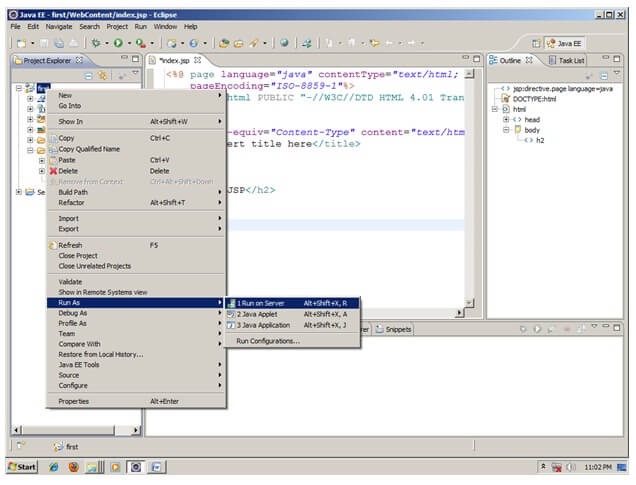
For starting the server and deploying the project in one step Right click on your project -> Run As -> Run on Server -> choose tomcat server -> next -> addAll -> finish.

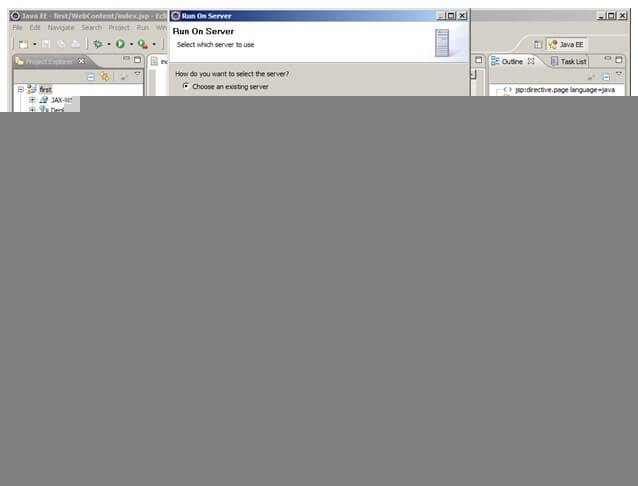
If you are using Eclipse IDE first time, you need to configure the tomcat server First

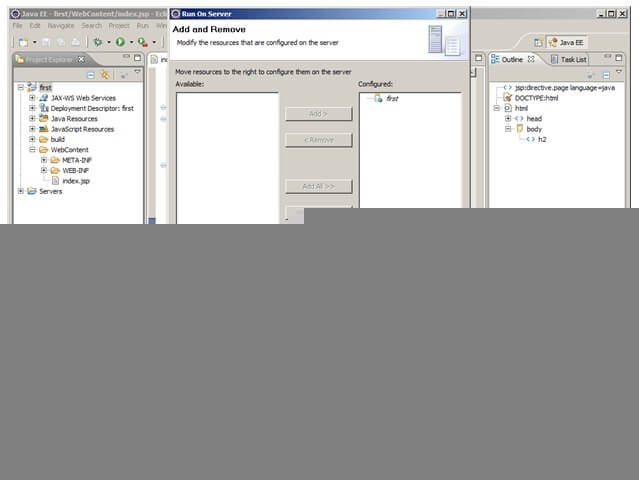
**Now start the tomcat server and deploy project**

For starting the server and deploying the project in one step Right click on your project -> Run As -> Run on Server -> choose tomcat server -> next -> addAll -> finish.

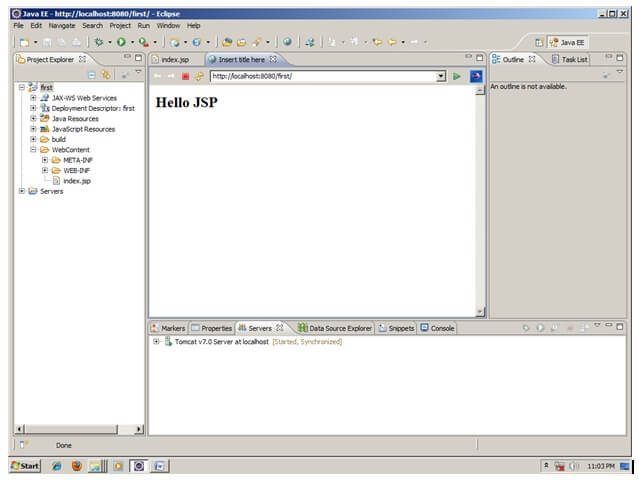








Yes, Let's see JSP is successfully running now.



# 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.

## JSP 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

### JSP scriptlet tag

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

1. <%  java source code %>

### Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

1. <html>
2. <body>
3. <% out.print("welcome to jsp"); %>
4. </body>
5. </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.

File: index.html

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

File: welcome.jsp

1. <html>
2. <body>
3. <%
4. String name=request.getParameter("uname");
5. out.print("welcome "+name);
6. %>
7. </form>
8. </body>
9. </html>

**JSP expression tag**

The code placed within **JSP 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**

1. <%=  statement %>

**Example of JSP expression tag**

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

1. <html>
2. <body>
3. <%= "welcome to jsp" %>
4. </body>
5. </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

1. <html>
2. <body>
3. Current Time: <%= java.util.Calendar.getInstance().getTime() %>
4. </body>
5. </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.

File: index.jsp

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname"><br/>
5. <input type="submit" value="go">
6. </form>
7. </body>
8. </html>

File: welcome.jsp

1. <html>
2. <body>
3. <%= "Welcome "+request.getParameter("uname") %>
4. </body>
5. </html>

**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:

1. <%!  field or method declaration %>

**Difference between JSP Scriptlet tag and 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**

1. <html>
2. <body>
3. <%! int data=50; %>
4. <%= "Value of the variable is:"+data %>
5. </body>
6. </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**

1. <html>
2. <body>
3. <%!
4. int cube(int n)
5. {
6. return n\*n\*n;
7. }
8. %>
9. <%= "Cube of 3 is:"+cube(3) %>
10. </body>
11. </html>

**JSP Implicit Objects**

There are **9 jsp implicit objects**. These objects are *created 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) JSP 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:

1. 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**

1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>

**JSP request implicit object**

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

**Example of JSP request implicit object**

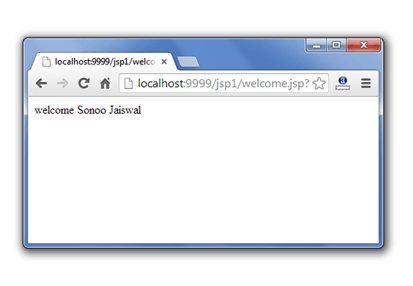
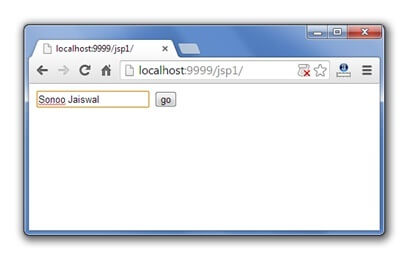
**index.html**

1. <form action="welcome.jsp">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>

**welcome.jsp**

1. <%
2. String name=request.getParameter("uname");
3. out.print("welcome "+name);
4. %>

**Output**



**3) JSP response implicit object**

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

Let's see the example of response implicit object where we are redirecting the response to the Google.

**Example of response implicit object**

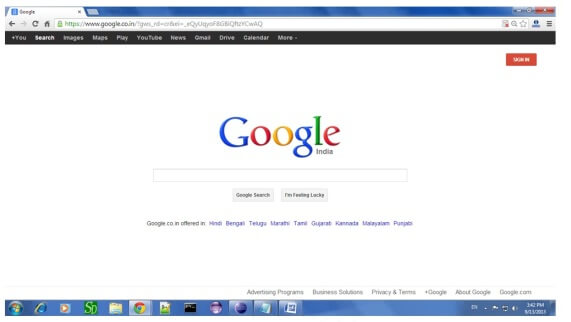
**index.html**

1. <form action="welcome.jsp">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>

**welcome.jsp**

1. <%
2. response.sendRedirect("http://www.google.com");
3. %>

**Output**



**4) JSP config implicit object**

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

Generally, it is used to get initialization parameter from the web.xml file.

**Example of config implicit object:**

**index.html**

1. <form action="welcome">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>

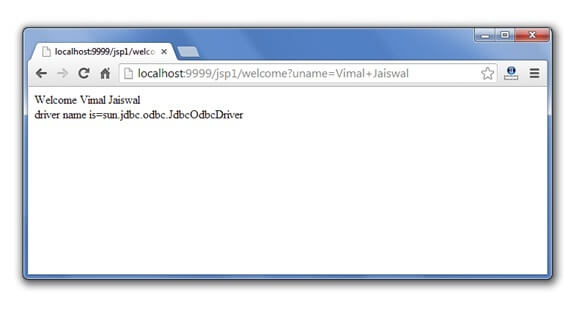
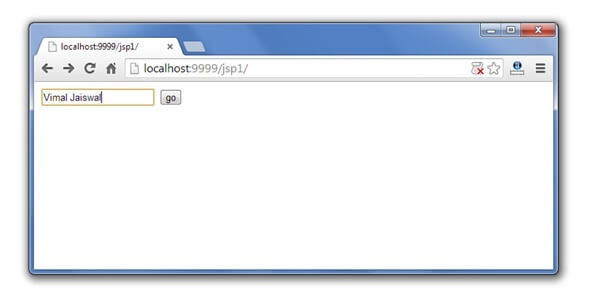
**web.xml file**

1. <web-app>
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <jsp-file>/welcome.jsp</jsp-file>
7. <init-param>
8. <param-name>dname</param-name>
9. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
10. </init-param>
12. </servlet>
14. <servlet-mapping>
15. <servlet-name>sonoojaiswal</servlet-name>
16. <url-pattern>/welcome</url-pattern>
17. </servlet-mapping>
19. </web-app>

**welcome.jsp**

1. <%
2. out.print("Welcome "+request.getParameter("uname"));
4. String driver=config.getInitParameter("dname");
5. out.print("driver name is="+driver);
6. %>

**Output**



**5) JSP application implicit object**

In JSP, application is an implicit object of type *ServletContext*.

The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

This initialization parameter can be used by all jsp pages.

**Example of application implicit object:**

**index.html**

1. <form action="welcome">
2. <input type="text" name="uname">
3. <input type="submit" value="go"><br/>
4. </form>

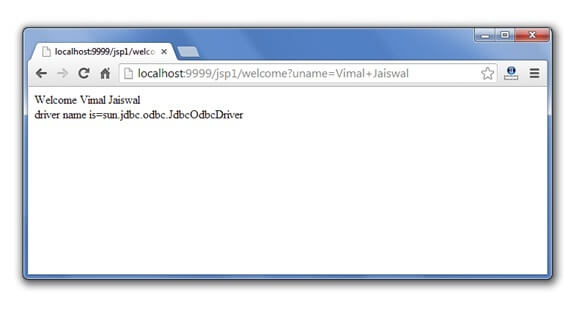
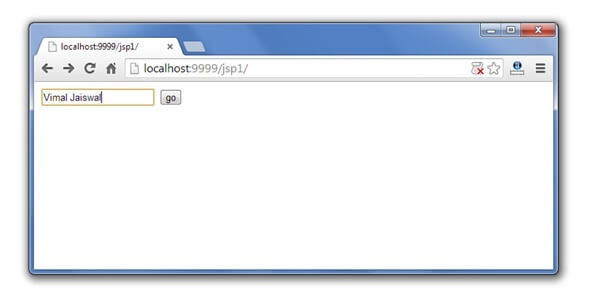
**web.xml file**

1. <web-app>
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <jsp-file>/welcome.jsp</jsp-file>
6. </servlet>
8. <servlet-mapping>
9. <servlet-name>sonoojaiswal</servlet-name>
10. <url-pattern>/welcome</url-pattern>
11. </servlet-mapping>
13. <context-param>
14. <param-name>dname</param-name>
15. <param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
16. </context-param>
18. </web-app>

**welcome.jsp**

1. <%
3. out.print("Welcome "+request.getParameter("uname"));
5. String driver=application.getInitParameter("dname");
6. out.print("driver name is="+driver);
8. %>

**Output**



**6) session implicit object**

|  |
| --- |
| In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information. |

**Example of session implicit object**

**index.html**

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

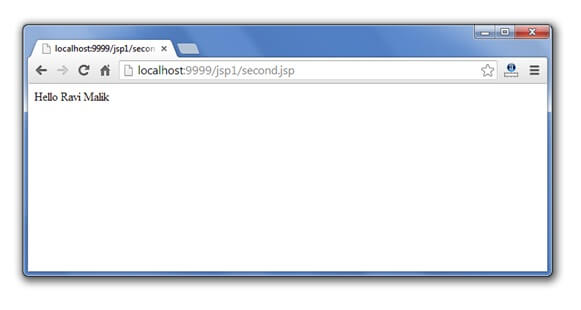
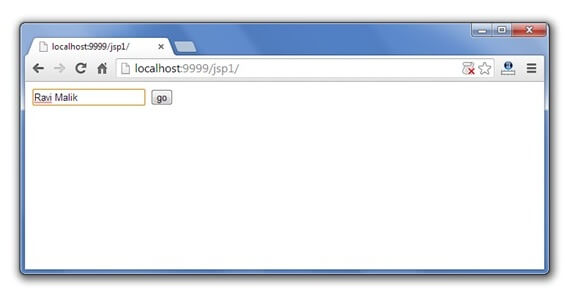
**welcome.jsp**

1. <html>
2. <body>
3. <%
5. String name=request.getParameter("uname");
6. out.print("Welcome "+name);
8. session.setAttribute("user",name);
10. <a href="second.jsp">second jsp page</a>
12. %>
13. </body>
14. </html>

**second.jsp**

1. <html>
2. <body>
3. <%
5. String name=(String)session.getAttribute("user");
6. out.print("Hello "+name);
8. %>
9. </body>
10. </html>

**Output**



**7) pageContext implicit object**

|  |
| --- |
| In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:   * page * request * session * application |
| In JSP, page scope is the default scope. |

**Example of pageContext implicit object**

**index.html**

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

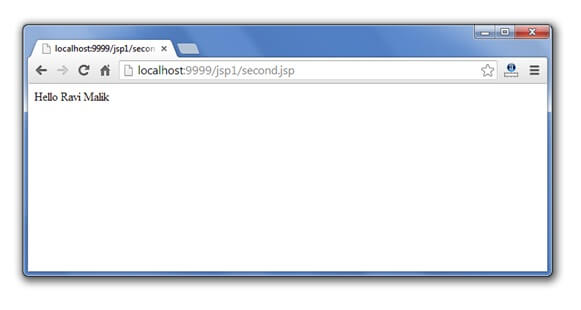
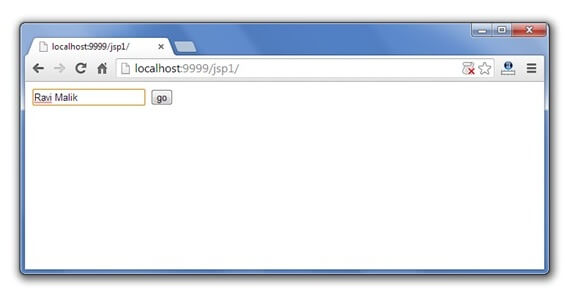
**welcome.jsp**

1. <html>
2. <body>
3. <%
5. String name=request.getParameter("uname");
6. out.print("Welcome "+name);
8. pageContext.setAttribute("user",name,PageContext.SESSION\_SCOPE);
10. <a href="second.jsp">second jsp page</a>
12. %>
13. </body>
14. </html>

**second.jsp**

1. <html>
2. <body>
3. <%
5. String name=(String)pageContext.getAttribute("user",PageContext.SESSION\_SCOPE);
6. out.print("Hello "+name);
8. %>
9. </body>
10. </html>

**Output**



# 8) page implicit object:

|  |
| --- |
| In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class. It is written as: |
| Object page=this; |
| For using this object it must be cast to Servlet type.For example: |
| <% (HttpServlet)page.log("message"); %> |
| Since, it is of type Object it is less used because you can use this object directly in jsp.For example: |
| <% this.log("message"); %> |

**9) exception implicit object**

|  |
| --- |
| In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive. Let's see a simple example: |

**Example of exception implicit object:**

**error.jsp**

1. <%@ page isErrorPage="true" %>
2. <html>
3. <body>
5. Sorry following exception occured:<%= exception %>
7. </body>
8. </html>

**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**

1. <%@ directive attribute="value" %>

**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**

1. <%@ 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.

1. <html>
2. <body>
4. <%@ include file="header.html" %>
6. Today is: <%= java.util.Calendar.getInstance().getTime() %>
8. </body>
9. </html>

**Note: The include directive includes the original content, so the actual page size grows at runtime.**

**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**

1. <%@ 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.

1. <html>
2. <body>
4. <%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
6. <mytag:currentDate/>
8. </body>
9. </html>

**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**

1. <form action="process.jsp">
2. No1:<input type="text" name="n1" /><br/><br/>
3. No1:<input type="text" name="n2" /><br/><br/>
4. <input type="submit" value="divide"/>
5. </form>

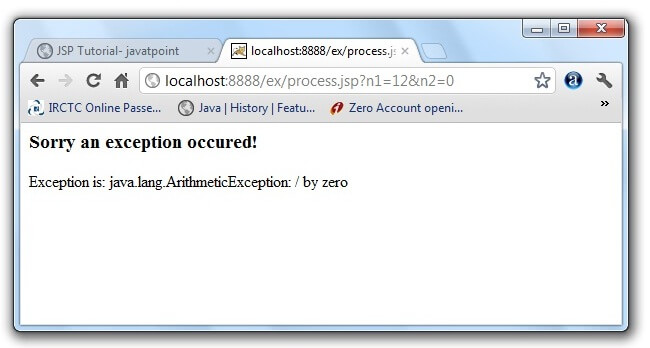
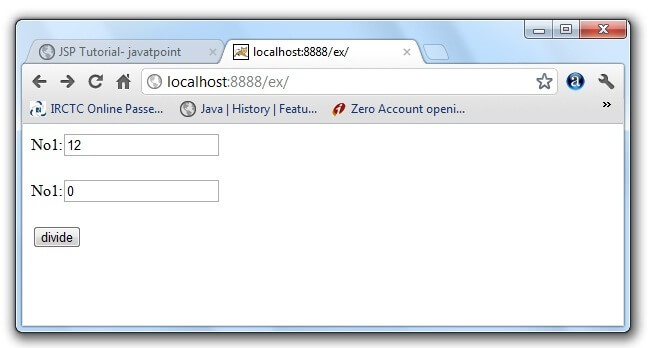
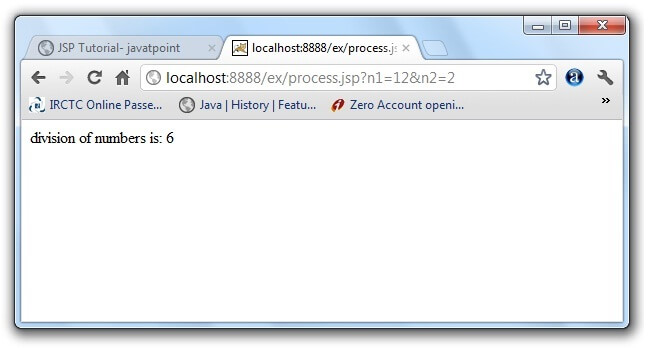
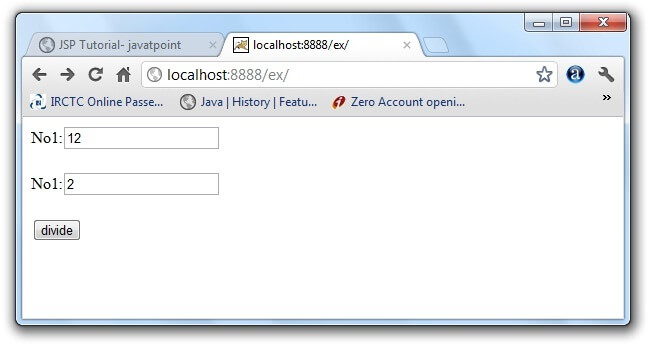
**process.jsp**

1. <%@ page errorPage="error.jsp" %>
2. <%
4. String num1=request.getParameter("n1");
5. String num2=request.getParameter("n2");
7. int a=Integer.parseInt(num1);
8. int b=Integer.parseInt(num2);
9. int c=a/b;
10. out.print("division of numbers is: "+c);
12. %>

**error.jsp**

1. <%@ page isErrorPage="true" %>
3. <h3>Sorry an exception occured!</h3>
5. Exception is: <%= exception %>

**Output of this example:**



**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**

1. <web-app>
3. <error-page>
4. <exception-type>java.lang.Exception</exception-type>
5. <location>/error.jsp</location>
6. </error-page>
8. </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**

1. <web-app>
3. <error-page>
4. <error-code>500</error-code>
5. <location>/error.jsp</location>
6. </error-page>
8. </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. |

1. <%@ page errorPage="error.jsp" %>
2. <%
4. String num1=request.getParameter("n1");
5. String num2=request.getParameter("n2");
7. int a=Integer.parseInt(num1);
8. int b=Integer.parseInt(num2);
9. int c=a/b;
10. out.print("division of numbers is: "+c);
12. %>

**4) error.jsp file is same as in the above example**

**JSP Action Tags**

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

The action tags are used to control the flow between pages and to use Java Bean. The Jsp action tags are given below.

|  |  |
| --- | --- |
| **JSP Action Tags** | **Description** |
| jsp:forward | forwards the request and response to another resource. |
| jsp:include | includes another resource. |
| jsp:useBean | creates or locates bean object. |
| jsp:setProperty | sets the value of property in bean object. |
| jsp:getProperty | prints the value of property of the bean. |
| jsp:plugin | embeds another components such as applet. |
| jsp:param | sets the parameter value. It is used in forward and include mostly. |
| jsp:fallback | can be used to print the message if plugin is working. It is used in jsp:plugin. |

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**

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

**Syntax of jsp:forward action tag with parameter**

1. <jsp:forward page="relativeURL | <%= expression %>">
2. <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
3. </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**

1. <html>
2. <body>
3. <h2>this is index page</h2>
5. <jsp:forward page="printdate.jsp" />
6. </body>
7. </html>

**printdate.jsp**

1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>

[download this example](https://static.javatpoint.com/src/jsp/forwardaction.zip)

**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**

1. <html>
2. <body>
3. <h2>this is index page</h2>
5. <jsp:forward page="printdate.jsp" >
6. <jsp:param name="name" value="javatpoint.com" />
7. </jsp:forward>
9. </body>
10. </html>

**printdate.jsp**

1. <html>
2. <body>
4. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
5. <%= request.getParameter("name") %>
7. </body>
8. </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 pages** because there might be changes in future.

The jsp:include tag can be used to include static as well as dynamic pages.

**Advantage of jsp:include action tag**

**Code reusability** : We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.

**Difference between jsp include directive and include action**

|  |  |
| --- | --- |
| **JSP include directive** | **JSP include action** |
| includes resource at translation time. | includes resource at request time. |
| better for static pages. | better for dynamic pages. |
| includes the original content in the generated servlet. | calls the include method. |

**Syntax of jsp:include action tag without parameter**

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

**Syntax of jsp:include action tag with parameter**

1. <jsp:include page="relativeURL | <%= expression %>">
2. <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
3. </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

1. <h2>this is index page</h2>
3. <jsp:include page="printdate.jsp" />
5. <h2>end section of index page</h2>

File: printdate.jsp

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

[download this example](https://static.javatpoint.com/src/jsp/includeaction.zip)

