**JSP**

Java Server Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

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.

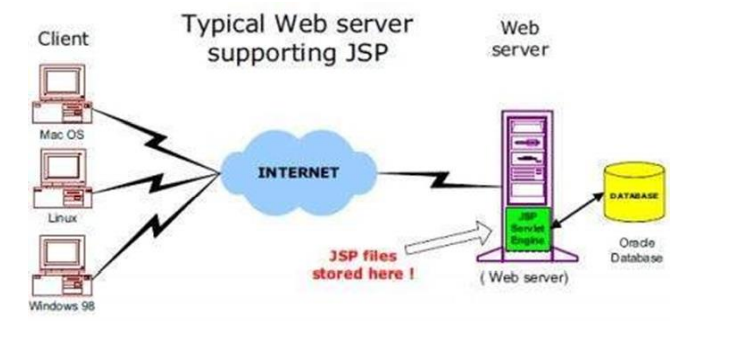
It provides some additional features such as Expression Language, Custom Tags, etc.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.

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

JSP ARCHETECTURE

The web server needs a JSP engine, i.e., a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development. A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs. Following diagram shows the position of JSP container and JSP files in a Web application.



**JSP Processing**

The following steps explain how the web server creates the Webpage using JSP:

• As with a normal page, your browser sends an HTTP request to the web server.

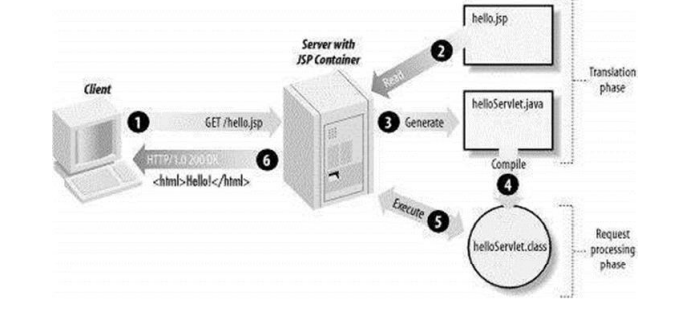
• The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.

• The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.

• The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.

* A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. This output is further passed on to the web server by the servlet engine inside an HTTP response.

• The web server forwards the HTTP response to your browser in terms of static HTML content. • Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.



### 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.

**Diagram

Description automatically generated**

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

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

### How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

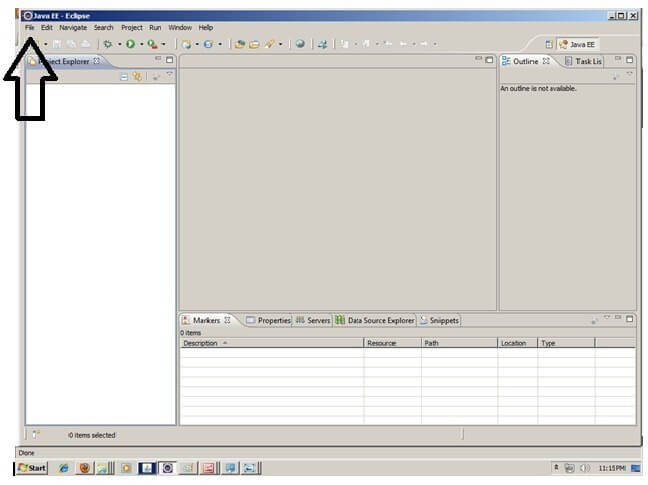
* Start the server
* Put the JSP file in a folder and deploy on the server
* Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

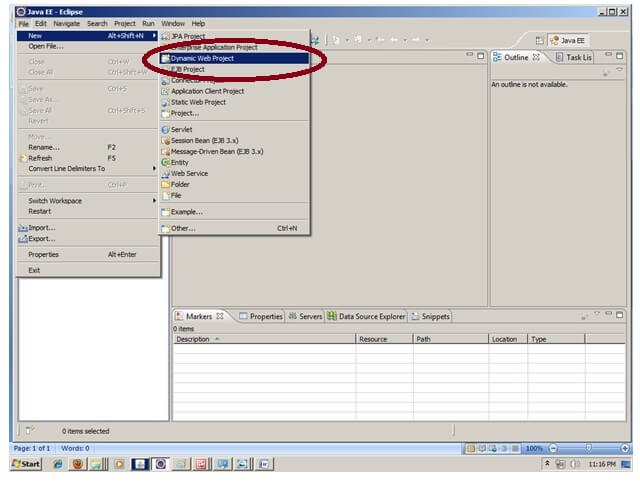
# Creating JSP in Eclipse IDE with Tomcat server

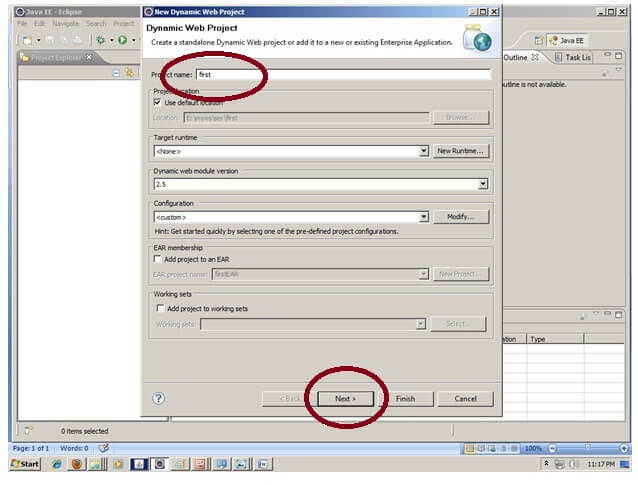
* Create a Dynamic web project
* create a jsp
* start tomcat server and deploy the project

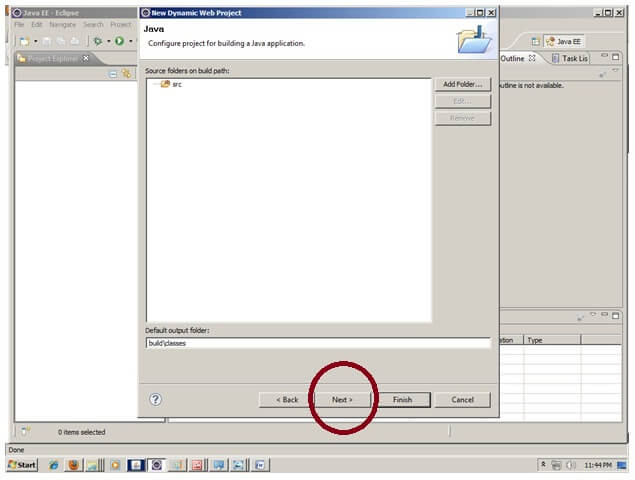
### 1) Create the dynamic web project

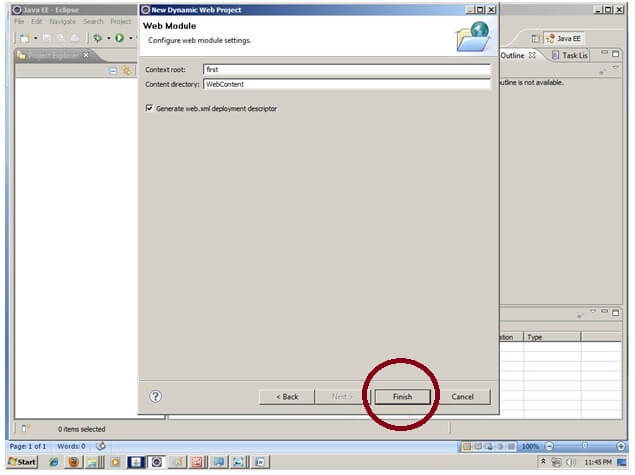
For creating a dynamic web project click on File Menu -> New -> dynamic web project -> write your project name e.g. first -> Finish.

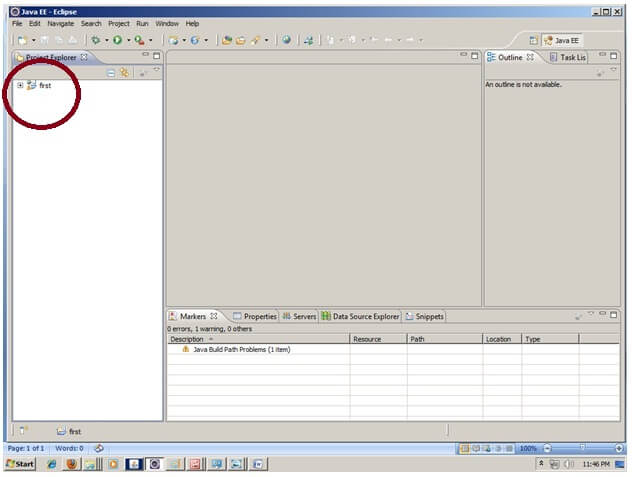






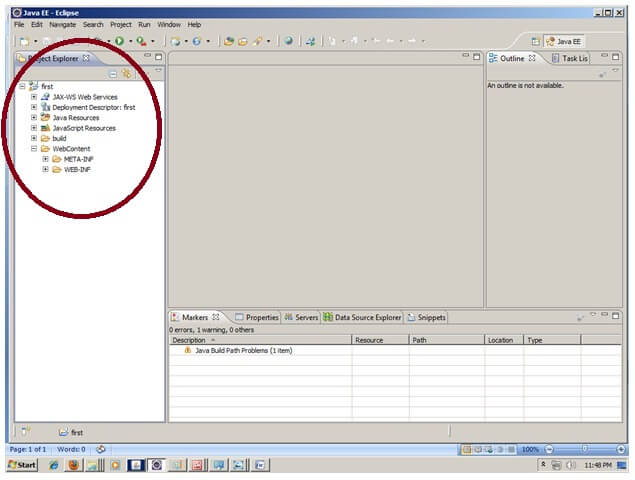


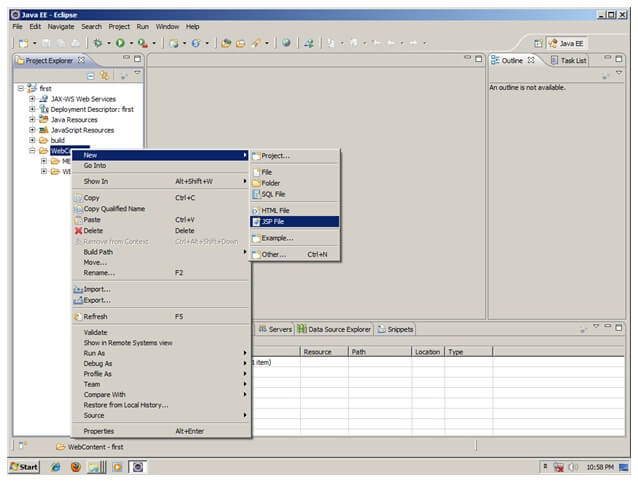


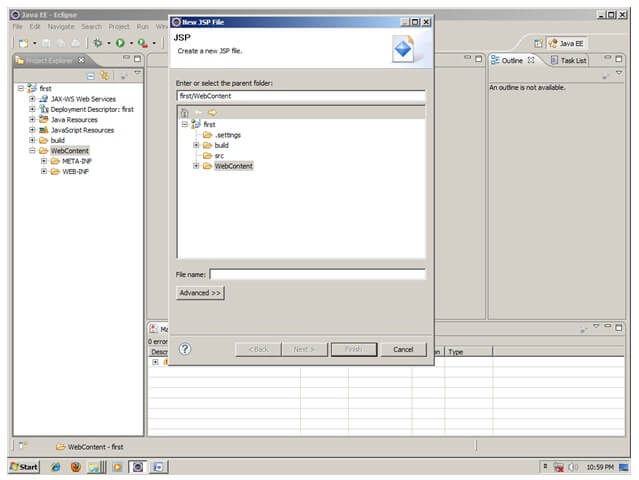


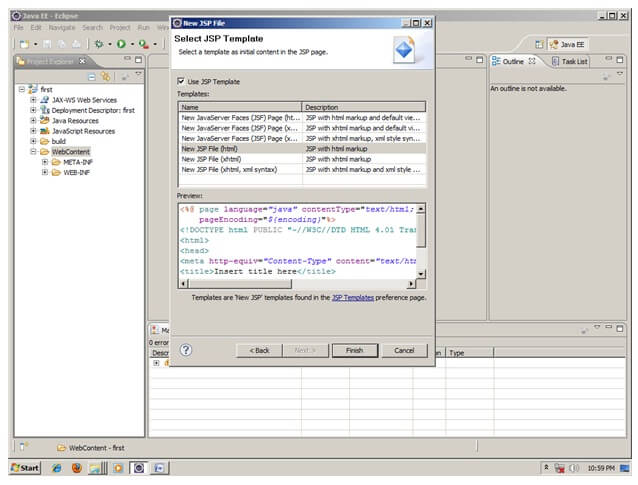
### 2) Create the JSP file in eclipse IDE

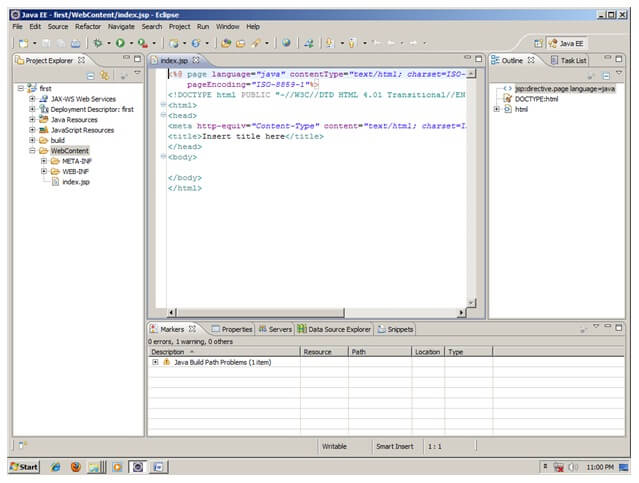
For creating a jsp file explore the project by clicking the + icon -> right click on WebContent -> New -> jsp -> write your jsp file name e.g. index -> next -> Finish.



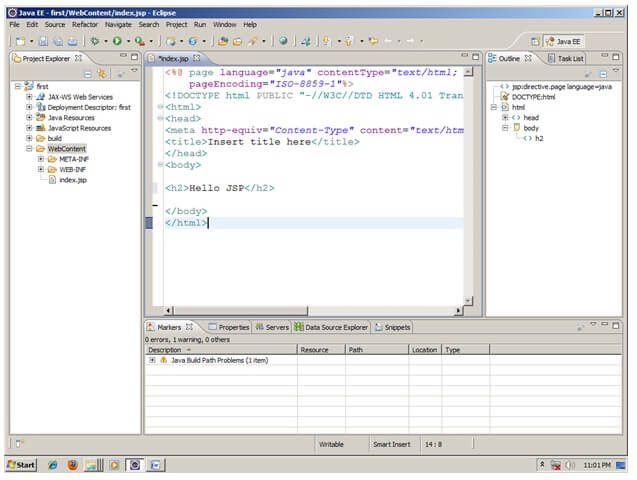








Now JSP file is created, let's write some code.



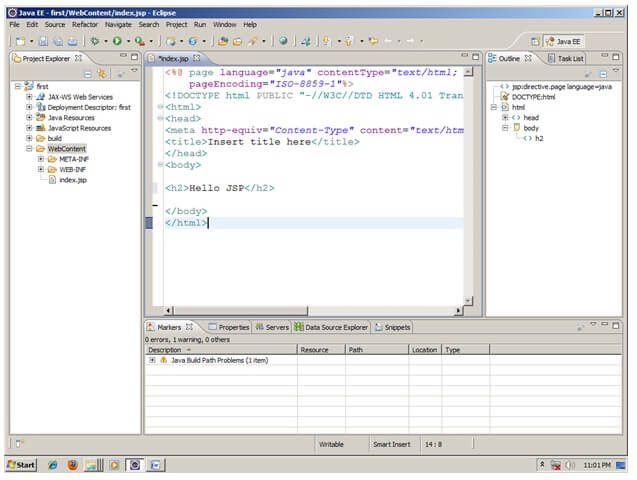
### 3) Start the server and deploy the project:

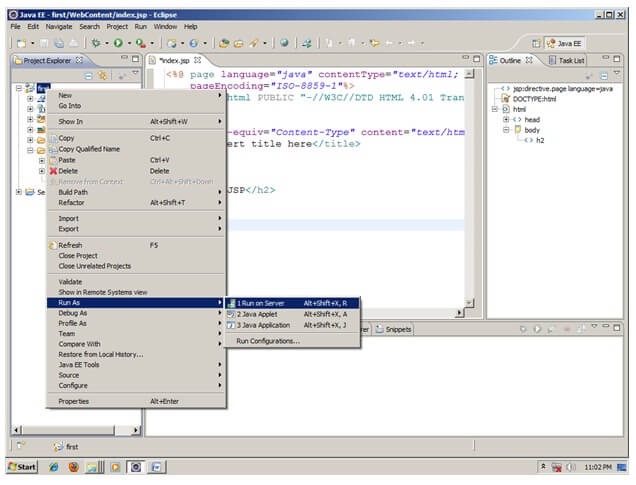
For starting the server and deploying the project in one step Right click on your project -> Run As -> Run on Server -> choose tomcat server -> next -> addAll -> finish.

If you are using Eclipse IDE first time, you need to configure the tomcat server First. Click for [How to configure tomcat server in eclipse IDE](https://www.javatpoint.com/how-to-configure-tomcat-server-in-eclipse-ide)

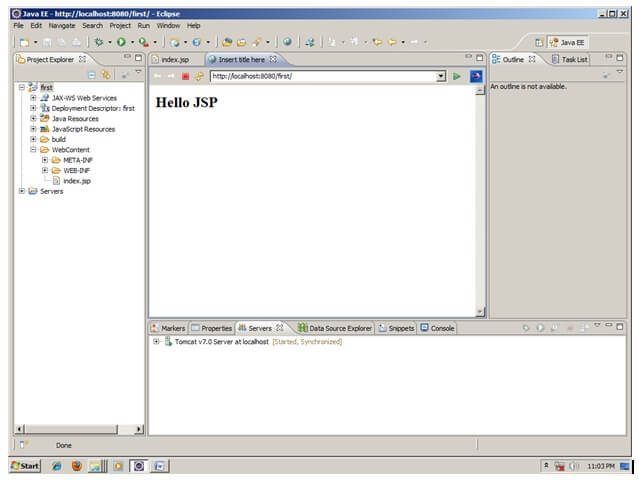
### Now start the tomcat server and deploy project

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.



**How JSP Pages Are Processed**

This section describes how JSP pages are processed by the Web server. Two key steps are involved:

1) compilation of the JSP page into a Java Servlet class;

2) execution of the compiled class.

Before looking at how a JSP page is processed, let's see how **a static HTML page** is processed by a Web server first:

* Step 1: The Web browser sends a HTTP request to the Web server with the path name of the HTML page.
* Step 2: The Web server picks up the HTML page by following the specified path name.
* Step 3: The Web server puts the content of the HTML page into the HTTP response without any changes.
* Step 4: The Web server sends the HTTP response back to the Web browser.

Now look at how **a JSP page is processed** by a Web server:

* Step 1: The Web browser sends a HTTP request to the Web server with the path name of the JSP page.
* Step 2: The Web server checks to see if there is a compiled version of the requested JSP page. If yes, continue with Step 4. If no, continue with Step 3.
* Step 3. The Web server converts the JSP page to a Java Servlet source code and compiles it into a Java bytecode (the compiled version of the JSP page).
* Step 4: The Web server executes the compiled version of the requested JSP page, and collects the output of the execution.
* Step 5: The Web server puts the output of the execution into a HTTP response.
* Step 6: The Web server sends the HTTP response back to the Web browser.

There are two key steps involved in serving a JSP page:

* **Compilation**: A JSP page must be compiled into a Java Servlet class, before it can be executed. The Web server can compile a JSP page in real-time if the page is requested for the first time and the page is not pre-compiled.
* **Execution**: When a JSP page is requested, its compiled class will be executed on the server. The server will send back the output of the execution, not the content of the JSP page.

### **Advantages of JSP over Servlets?**

* We can generate HTML response from servlets also but the process is cumbersome and error prone, when it comes to writing a complex HTML response, writing in a servlet will be a nightmare. JSP helps in this situation and provide us flexibility to write normal HTML page and include our java code only where it’s required.
* JSP provides additional features such as tag libraries, expression language, custom tags that helps in faster development of user views.
* JSP pages are easy to deploy, we just need to replace the modified page in the server and container takes care of the deployment. For servlets, we need to recompile and deploy whole project again.

### **Life cycle of JSP Page**

JSP life cycle is also managed by container. Usually every web container that contains servlet container also contains JSP container for managing JSP pages.

JSP pages life cycle phases are:

* **Translation** – JSP pages doesn’t look like normal java classes, actually JSP container parse the JSP pages and translate them to generate corresponding servlet source code. If JSP file name is home.jsp, usually its named as home\_jsp.java.
* **Compilation** – If the translation is successful, then container compiles the generated servlet source file to generate class file.
* **Class Loading** – Once JSP is compiled as servlet class, its lifecycle is similar to servlet and it gets loaded into memory.
* **Instance Creation** – After JSP class is loaded into memory, its object is instantiated by the container.
* **Initialization** – The JSP class is then initialized and it transforms from a normal class to servlet. After initialization, ServletConfig and ServletContext objects become accessible to JSP class.
* **Request Processing** – For every client request, a new thread is spawned with ServletRequest and ServletResponse to process and generate the HTML response.
* **Destroy** – Last phase of JSP life cycle where it’s unloaded into memory.

# 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:

### 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. %>

# 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. %>

# 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. <%
4. String name=request.getParameter("uname");
5. out.print("Welcome "+name);
6. session.setAttribute("user",name);
7. <a href="second.jsp">second jsp page</a>
8. %>
9. </body>
10. </html>

### second.jsp

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

# 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 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 Expression Language (EL)

1. <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
2. pageEncoding="ISO-8859-1"%>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
4. <html>
5. <head>
6. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7. <title>Guru JSP1</title>
8. </head>
9. <body>
10. <a>Expression is:</a>
11. ${1+2};
12. </body>
13. </html>

**Explanation of Code:**

* **Code Line 11:** Expression Language (EL) is set where we are adding two numbers 1+2, hence it will give output as 3.

# 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 %>

# Pass values from form to form

Java program to pass values from one form to another form in JSP

There are many occasions when the programmer need to pass some value to many pages in their application. For example in a shopping cart application programmer wants that if user have logged once then until it logs out his name should be displayed on the other page of the application.  So it is most frequently used in to the different applications. Here in the **JSP** (Java Server Pages) we can pass values by the two ways:

   First, if we have to only pass values from one page to another consecutive page which is being called then we can get these values by using the implicit object **request** to call the method **request.getAttribute("parameter name")**.

   Second, we can pass the parameters value to many pages by making the session true for those pages and the value which is to pass must be set by the method **session.setAttribute("attribute name")**.

In our this example program we have used both ways to pass values to different pages. Here we have created three JSP pages.

**FirstForm.jsp**

|  |
| --- |
| <%@ page language="java" session="true"%> <h1> <font face="Comic Sans MS" color="#000080">   Input Information -First Page</font> </h1> <form name="frm" method="post" action="SecondForm.jsp"> <table width="485" height="172" border="0"    bordercolor="#003300" bgcolor="#003300">   <tr>   <td width="121" height="37"> <font size="4"     color="#FFFFFF" face="Comic Sans MS">   <b> First Name</b></font> </td>   <td colspan="2" height="37" width="348">    <input name="firstname" type="text" align="left" size="38"/>   </td>   </tr>   <tr>   <td height="39" width="121"><b><font size="4"     color="#FFFFFF" face="Comic Sans MS">    Last Name</font></b> </td>   <td colspan="2" height="39" width="348">   <input name="lastname" type="text" align="left" size="38"/>   </td>   </tr>   <tr>   <td height="39" width="121"><b><font size="4"    color="#FFFFFF" face="Comic Sans MS">Password</font>   </b></td>   <td colspan="2" height="39" width="348">    <input name="password" type="password" align="left"/>    </td>   </tr>  <tr>   <td colspan="3" height="41" width="475">   <input name="submit" type="submit" align="left" value="Next"/>    </td>   </tr> </table> </form> |

**SecondForm.jsp**

|  |
| --- |
| <%@ page language="java" session="true"%> <div align="right" ><font face="Comic Sans MS"    color="#000080">Welcome <u>    <%=request.getParameter("firstname")%></u>   </font> </div> <% session.setAttribute("password",request.getParameter("password"));%> <h1> <font face="Comic Sans MS" color="#000080">    Fill more information</font> </h1> <form name="frm2" method="post" action="FinalResult.jsp"> <table width="485" height="172" border="0"    bordercolor="#003300" bgcolor="#003300">   <tr>   <td width="121" height="37"> <font size="4"    color="#FFFFFF" face="Comic Sans MS">   <b> First Name</b></font> </td>   <td colspan="2" height="37" width="348">   <input name="firstname2" type="text" align="left" size="38"     value="<%=request.getParameter("firstname")%>"/></td>   </tr>   <tr>   <td height="39" width="121"><b><font size="4"    color="#FFFFFF" face="Comic Sans MS">Last Name   </font></b> </td>   <td colspan="2" height="39" width="348">   <input name="lastname2" type="text" align="left" size="38"    value="<%=request.getParameter("lastname")%>"/></td>   </tr>   <tr>   <td height="39" width="121"><b><font size="4"     color="#FFFFFF" face="Comic Sans MS">Address</font></b></td>   <td colspan="2" height="39" width="348">    <input name="address" type="text" align="left"/></td>   </tr>  <tr>   <td colspan="3" height="41" width="475">    <input name="submit" type="submit" align="left" value="Show"/>   </td>   </tr> </table> </form> |

**FinalResult.jsp**

|  |
| --- |
| <%@ page language="java" session="true"%>  First Name : <%=request.getParameter("firstname2") %><br> Last Name  : <%=request.getParameter("lastname2") %><br> Password : <%=session.getAttribute("password") %><br> Address  : <%=request.getParameter("address") %><br> |

# 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 page directive**

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

### **Syntax of JSP page directive**

1. <%@ 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**

1. <html>
2. <body>
3. <%@ page **import**="java.util.Date" %>
4. Today is: <%= **new** Date() %>
5. </body>
6. </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**

1. <html>
2. <body>
3. <%@ page contentType=application/msword %>
4. Today is: <%= **new** java.util.Date() %>
5. </body>
6. </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**

1. <html>
2. <body>
3. <%@ page info="composed by Sonoo Jaiswal" %>
4. Today is: <%= **new** java.util.Date() %>
5. </body>
6. </html>

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

1. **public** String getServletInfo() {
2. **return** "composed by Sonoo Jaiswal";
3. }

### **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**

1. <html>
2. <body>
3. <%@ page buffer="16kb" %>
4. Today is: <%= **new** java.util.Date() %>
5. </body>
6. </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. |

1. <%@ 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:

1. **public** **class** SimplePage\_jsp **extends** HttpJspBase
2. **implements** SingleThreadModel{
3. .......
4. }

### **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**

1. //index.jsp
2. <html>
3. <body>
4. <%@ page errorPage="myerrorpage.jsp" %>
5. <%= 100/0 %>
6. </body>
7. </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**

1. //myerrorpage.jsp
2. <html>
3. <body>
4. <%@ page isErrorPage="true" %>
5. Sorry an exception occured!<br/>
6. The exception is: <%= exception %>
7. </body>
8. </html>

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>
3. <%@ include file="header.html" %>
4. Today is: <%= java.util.Calendar.getInstance().getTime() %>
5. </body>
6. </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>
3. <%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
4. <mytag:currentDate/>
5. </body>
6. </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. <%
3. String num1=request.getParameter("n1");
4. String num2=request.getParameter("n2");
5. **int** a=Integer.parseInt(num1);
6. **int** b=Integer.parseInt(num2);
7. **int** c=a/b;
8. out.print("division of numbers is: "+c);
9. %>

#### **error.jsp**

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

### **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. |

### **Example of jsp:forward action tag**

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>
4. <jsp:forward page="printdate.jsp" />
5. </body>
6. </html>

### **printdate.jsp**

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