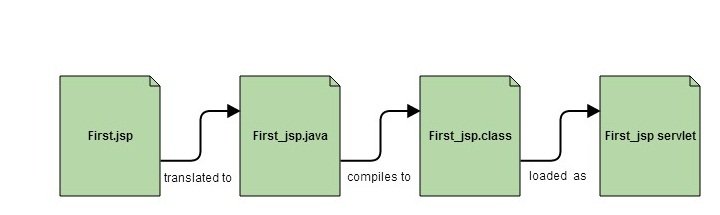
JSP:

**JSP** technology is used to create dynamic web applications. **JSP** pages are easier to maintain then a **Servlet**. JSP pages are opposite of Servlets as a servlet adds [HTML](https://www.studytonight.com/code/html/) code inside Java code, while JSP adds [Java code](https://www.studytonight.com/java/overview-of-java.php) inside HTML using JSP tags. Everything a Servlet can do, a JSP page can also do it.

JSP enables us to write HTML pages containing tags, inside which we can include powerful Java programs. **Using JSP, one can easily separate Presentation and Business logic** as a web designer can design and update JSP pages creating the presentation layer and java developer can write server side complex computational code without concerning the web design. And both the layers can easily interact over HTTP requests.

In the end a JSP becomes a Servlet

**JSP** pages are converted into **Servlet** by the Web Container. The Container translates a JSP page into servlet **class source(.java)** file and then compiles into a Java Servlet class.



Why JSP is preffered over servlets?

* JSP provides an easier way to code dynamic web pages.
* JSP does not require additional files like, java class files, web.xml etc
* Any change in the JSP code is handled by Web Container(Application server like tomcat), and doesn't require re-compilation.
* JSP pages can be directly accessed, and web.xml mapping is not required like in servlets.

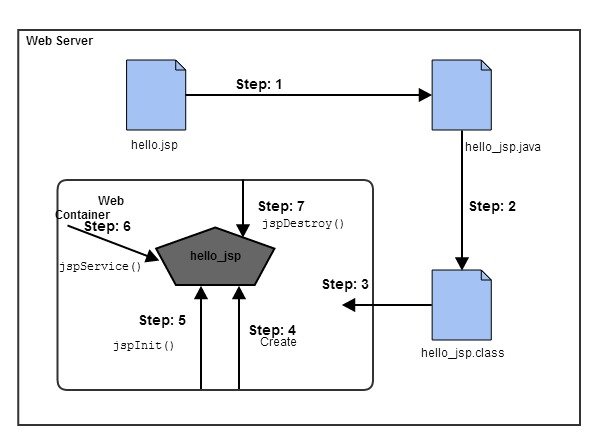
Advantage of JSP

* Easy to maintain and code.
* High Performance and Scalability.
* JSP is built on Java technology, so it is platform independent.

# **Lifecycle of JSP:**

A JSP page is converted into Servlet in order to service requests. The translation of a JSP page to a Servlet is called Lifecycle of JSP. JSP Lifecycle is exactly same as the Servlet Lifecycle, with one additional first step, which is, translation of JSP code to Servlet code. Following are the JSP Lifecycle steps:

1. Translation of JSP to Servlet code.
2. Compilation of Servlet to bytecode.
3. Loading Servlet class.
4. Creating servlet instance.
5. Initialization by calling jspInit() method
6. Request Processing by calling \_jspService() method
7. Destroying by calling jspDestroy() method



**Web Container** translates JSP code into a **servlet class source(.java) file**, then compiles that into a java servlet class. In the third step, the servlet class bytecode is loaded using classloader. The Container then creates an instance of that servlet class.

The initialized servlet can now service request. For each request the **Web Container** call the **\_jspService()** method. When the Container removes the servlet instance from service, it calls the **jspDestroy()** method to perform any required clean up.

#### **What happens to a JSP when it is translated into Servlet**

Let's see what really happens to JSP code when it is translated into Servlet. The code written inside <% %> is JSP code.

<html>

<head>

<title>My First JSP Page</title>

</head>

<%

int count = 0;

%>

<body>

Page Count is:

<% out.println(++count); %>

</body>

</html>

Copy

The above JSP page(hello.jsp) becomes this Servlet,

public class hello\_jsp extends HttpServlet

{

public void \_jspService(HttpServletRequest request, HttpServletResponse response)

throws IOException,ServletException

{

PrintWriter out = response.getWriter();

response.setContenType("text/html");

out.write("<html><body>");

int count=0;

out.write("Page count is:");

out.print(++count);

out.write("</body></html>");

}

}

Copy

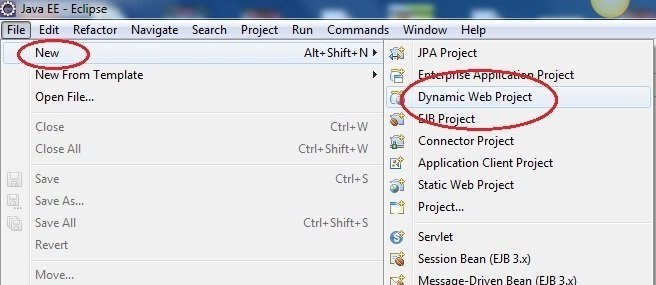
This is just to explain, what happens internally. As a JSP developer, you do not have to worry about how a JSP page is converted to a Servlet, as it is done automatically by the web container.

A JSP page looks similar to an [HTML page](https://www.studytonight.com/code/html/), but a JSP page also has Java code in it. We can put any regular Java Code in a JSP file using a **scriplet tag** which start with <% and ends with %>. JSP pages are used to develop dynamic responses.

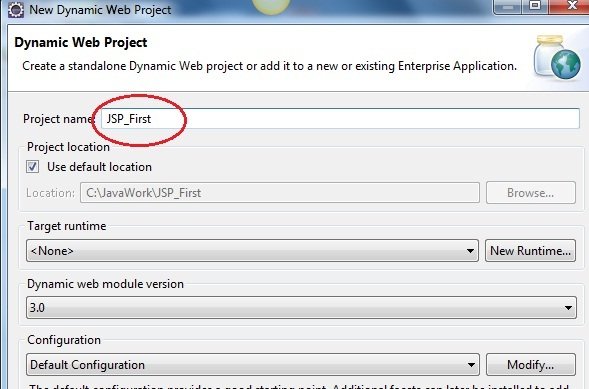
To learn HTML, go to [HTML Interactive Course](https://www.studytonight.com/code/html) and learn HTML while practicing it side by side.

Example of creating a JSP Page in Eclipse

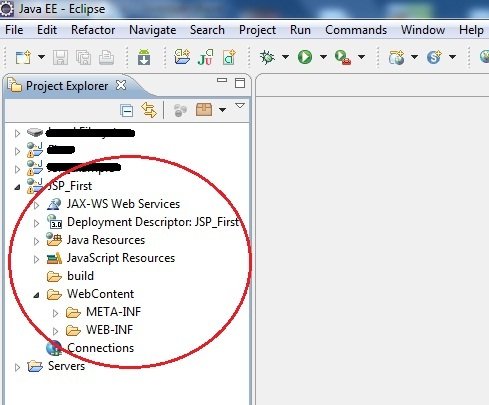
* Open Eclipse, Click on **New → Dynamic Web Project**



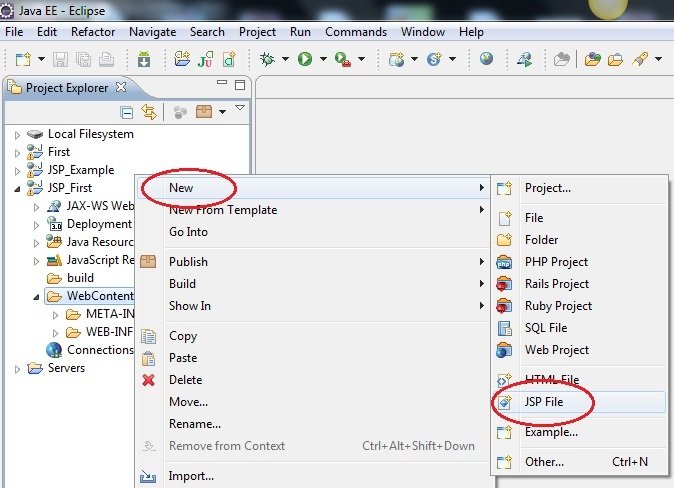
* Give a name to your project and click on OK



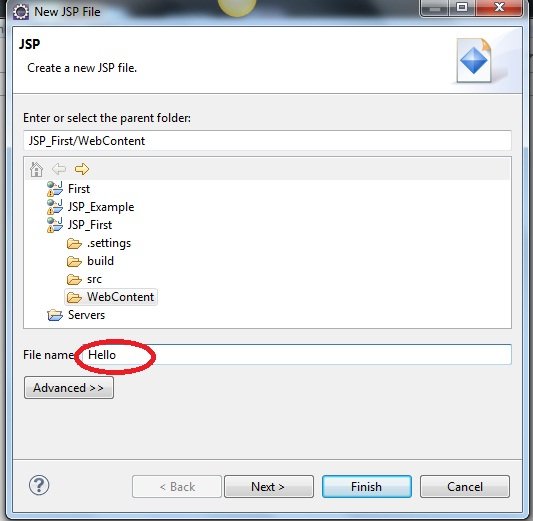
* You will see a new project created in Project Explorer



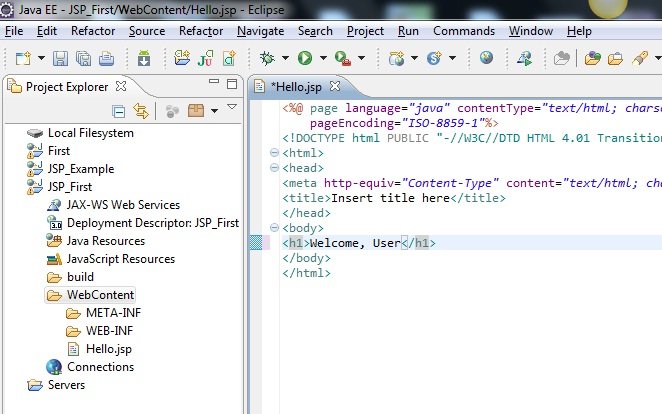
* To create a new JSP file right click on Web Content directory, **New → JSP file**



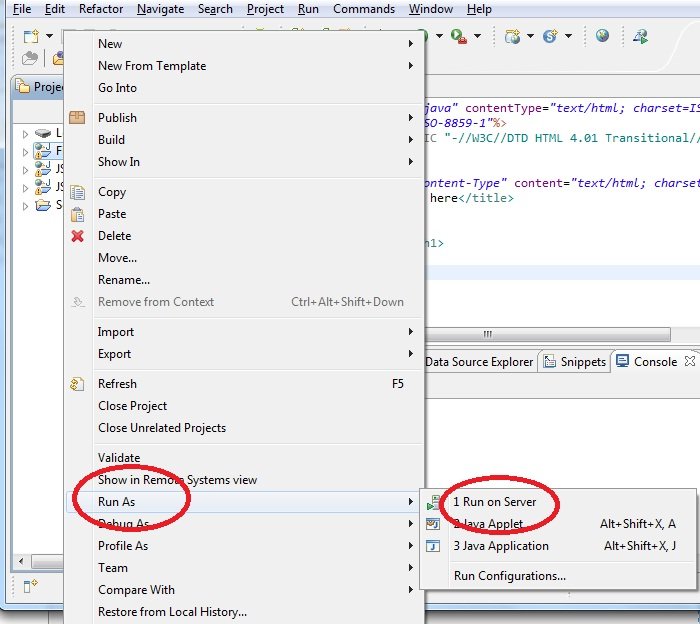
* Give a name to your JSP file and click Finish.



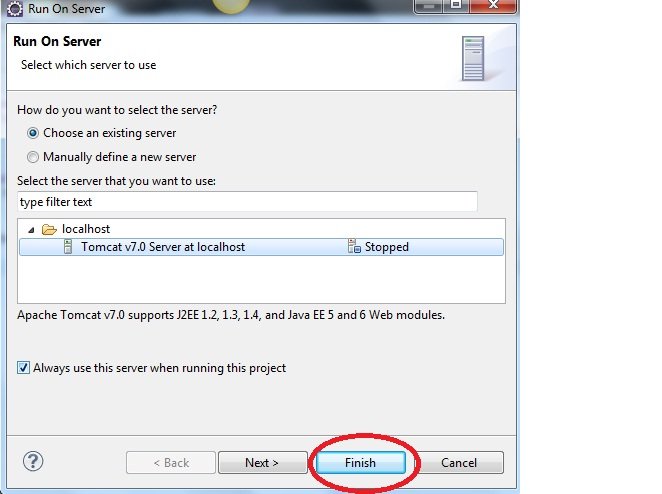
* Write something in your JSP file. The complete HTML and the JSP code, goes inside the <body> tag, just like HTML pages.



* To run your project, right click on **Project**, select **Run As → Run on Server**



* To start the server, Choose existing server name and click on finish



* See the Output in your browser.



# **JSP Scripting Element**

JSP Scripting element are written inside <% %> tags. These code inside <% %> tags are processed by the JSP engine during translation of the JSP page. Any other text in the JSP page is considered as HTML code or plain text.

**Example:**

<html>

<head>

<title>My First JSP Page</title>

</head>

<%

int count = 0;

%>

<body>

Page Count is <% out.println(++count); %>

</body>

</html>

Copy

Just to experiment, try removing the <% %> scriplet tag from the above code and run it as JSP. You will see that everything is printed as it is on the browser, because without the scriplet tag, everything is considered plain HTML code.

### There are five different types of scripting elements

|  |  |
| --- | --- |
| **Scripting Element** | **Example** |
| **Comment** | <%-- comment --%> |
| **Directive** | <%@ directive %> |
| **Declaration** | <%! declarations %> |
| **Scriptlet** | <% scriplets %> |
| **Expression** | <%= expression %> |

## JSP Comment

JSP Comment is used when you are creating a JSP page and want to put in comments about what you are doing. JSP comments are only seen in the JSP page. These comments are not included in servlet source code during translation phase, nor they appear in the HTTP response. Syntax of JSP comment is as follows :

<%-- JSP comment --%>

Copy

**Simple Example of JSP Comment**

<html>

<head>

<title>My First JSP Page</title>

</head>

<%

int count = 0;

%>

<body>

<%-- Code to show page count --%>

Page Count is <% out.println(++count); %>

</body>

</html>

Scriptlet:

Scriptlet Tag allows you to write java code inside JSP page. Scriptlet tag implements the \_jspService method functionality by writing script/java code. Syntax of Scriptlet Tag is as follows :

<% JAVA CODE %>

Copy

## JSP: Example of Scriptlet

In this example, we will show number of page visit.

<html>

<head>

<title>My First JSP Page</title>

</head>

<%

int count = 0;

%>

<body>

Page Count is <% out.println(++count); %>

</body>

</html>

Copy

We have been using the above example since last few lessons and in this scriptlet tags are used. Everything written inside the scriptlet tag is compiled as java code. Like in the above example, we initialize count variable of type int with a value of 0. And then we print it while using ++ operator to perform addition on it.

JSP makes it so easy to perform calculations, database interactions etc directly from inside the HTML code. Just write your java code inside the scriptlet tags.

### Example of JSP Scriptlet Tag

In this example, we will create a simple JSP page which retrieves the name of the user from the request parameter. The **index.html** page will get the username from the user.

**index.html**

<form method="POST" action="welcome.jsp">

Name <input type="text" name="user" >

<input type="submit" value="Submit">

</form>

Copy

In the above HTML file, we have created a form, with an input text field for user to enter his/her name, and a Submit button to submit the form. On submission an HTTP Post request ( method="**POST**" ) is made to the welcome.jsp file ( action="**welcome.jsp**" ), with the form values.

**welcome.jsp**

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Welcome Page</title>

</head>

<%

String user = request.getParameter("user");

%>

<body>

Hello, <% out.println(user); %>

</body>

</html>

Copy

As we know that a JSP code is translated to Servlet code, in which \_jspService method is executed which has **HttpServletRequest** and **HttpServletResponse** as argument. So in the **welcome.jsp** file, **request** is the HTTP Request and it has all the parameters sent from the form in index.html page, which we can be easily get using getParameter() with name of parameter as argument, to get its value.

### Mixing scriptlet Tag and HTML

Let's see how we can utilize the power of JSP scripting with HTML to build dynamic webpages with help of a few examples.

If we want to create a Table in HTML with some dynamic data, for example by reading data from some MySQL table or file. How to do that? Here we will describe you the technique by creating a table with numbers 1 to n.

<table border = 1>

<%

for ( int i = 0; i < n; i++ ) {

%>

<tr>

<td>Number</td>

<td><%= i+1 %></td>

</tr>

<%

}

%>

</table>

Copy

The above piece of code will go inside the <body> tag of the JSP file and will work when you initialize n with some value.

Also, observer closely we have only included the java code inside the scriptlet tag, and all the HTML part is outside of it. Similarly we can do plenty of stuff.

Here is one more very simple example :

<%

if ( hello ) {

%>

<p>Hello, world</p>

<%

} else {

%>

<p>Goodbye, world</p>

<%

}

%>

Copy

Above code is using if-else condition to evaluate what to show, based on the value of a **boolean** variable named hello.

You can even ask user to enter the value of hello, using HTML Form and evaluate your JSP code based on that.

# **JSP Declaration Tag**

We know that at the end a JSP page is translated into Servlet class. So when we declare a variable or method in JSP inside **Declaration Tag**, it means the declaration is made inside the Servlet class but outside the service(or any other) method. You can declare static member, instance variable and methods inside **Declaration Tag**. Syntax of Declaration Tag :

<%! declaration %>

Copy

### Example of Declaration Tag

<html>

<head>

<title>My First JSP Page</title>

</head>

<%!

int count = 0;

%>

<body>

Page Count is:

<% out.println(++count); %>

</body>

</html>

Copy

In the above code, we have used the declaration tag to declare variable count. The above JSP page becomes this Servlet :

public class hello\_jsp extends HttpServlet

{

int count=0;

public void \_jspService(HttpServletRequest request, HttpServletResponse response)

throws IOException,ServletException

{

PrintWriter out = response.getWriter();

response.setContenType("text/html");

out.write("<html><body>");

out.write("Page count is:");

out.print(++count);

out.write("</body></html>");

}

}

Copy

In the above servlet, we can see that variable count is declared outside the \_jspservice() method. If we declare the same variable using [scriptlet tag](https://www.studytonight.com/jsp/jsp-scriptlet-tag.php" \t "_blank), it will come inside the service method, as seen in the last lesson.

### When to use Declaration tag and not scriptlet tag

If you want to include any method in your JSP file, then you must use the declaration tag, because during translation phase of JSP, methods and variables inside the declaration tag, becomes instance methods and instance variables and are also assigned default values.

For example:

<html>

<head>

<title>My First JSP Page</title>

</head>

<%!

int count = 0;

int getCount() {

System.out.println( "In getCount() method" );

return count;

}

%>

<body>

Page Count is:

<% out.println(getCount()); %>

</body>

</html>

Copy

Above code will be translated into following servlet :

public class hello\_jsp extends HttpServlet

{

int count = 0;

int getCount() {

System.out.println( "In getCount() method" );

return count;

}

public void \_jspService(HttpServletRequest request, HttpServletResponse response)

throws IOException,ServletException

{

PrintWriter out = response.getWriter();

response.setContenType("text/html");

out.write("<html><body>");

out.write("Page count is:");

out.print(getCount());

out.write("</body></html>");

}

}

Copy

While, anything we add in scriptlet tag, goes inside the \_jspservice() method, therefore we cannot add any function inside the scriptlet tag, as on compilation it will try to create a function getCount() inside the service method, and in [Java](https://www.studytonight.com/java/overview-of-java.php), method inside a method is not allowed.

# **JSP Directive Tag**

**Directive Tag** gives special instruction to Web Container at the time of page translation. Directive tags are of three types: **page**, **include** and **taglib**.

|  |  |
| --- | --- |
| **Directive** | **Description** |
| <%@ page ... %> | defines page dependent properties such as language, session, errorPage etc. |
| <%@ include ... %> | defines file to be included. |
| <%@ taglib ... %> | declares tag library used in the page |

We'll discuss about **include** and **taglib** directive later. You can place page directive anywhere in the JSP file, but it is good practice to make it as the first statement of the JSP page.

The **Page directive** defines a number of page dependent properties which communicates with the Web Container at the time of translation. Basic syntax of using the page directive is <%@ page attribute="value" %> where attributes can be one of the following :

* import attribute
* language attribute
* extends attribute
* session attribute
* isThreadSafe attribute
* isErrorPage attribute
* errorPage attribute
* contentType attribute
* autoFlush attribute
* buffer attribute

### import attribute

The import attribute defines the set of classes and packages that must be imported in servlet class definition. For example

<%@ page import="java.util.Date" %>

or

<%@ page import="java.util.Date,java.net.\*" %>

Copy

### language attribute

language attribute defines scripting language to be used in the page.

### extends attribute

extends attribute defines the class name of the superclass of the servlet class that is generated from the JSP page.

### session attribute

session attribute defines whether the JSP page is participating in an HTTP session. The value is either true or false.

### isThreadSafe attribute

isThreadSafe attribute declares whether the JSP is thread-safe. The value is either true or false

### isErrorPage attribute

isErrorPage attribute declares whether the current JSP Page represents another JSP's error page.

### errorPage attribute

errorPage attribute indicates another JSP page that will handle all the run time exceptions thrown by current JSP page. It specifies the URL path of another page to which a request is to be dispatched to handle run time exceptions thrown by current JSP page.

### contentType attribute

contentType attribute defines the MIME type for the JSP response.

### autoFlush attribute

autoFlush attribute defines whether the buffered output is flushed automatically. The default value is "true".

### buffer attribute

buffer attribute defines how buffering is handled by the implicit **out** object.

# **JSP Expression Tag**

Expression Tag is used to print out java language expression that is put between the tags. An expression tag can hold any java language expression that can be used as an argument to the **out.print()** method. Syntax of Expression Tag

<%= Java Expression %>

Copy

**When the Container sees this**

<%= (2\*5) %>

Copy

**It turns it into this:**

out.print((2\*5));

Copy

**Note:** Never end an expression with semicolon inside Expression Tag. Like this:

<%= (2\*5); %>

Copy

### Example of Expression Tag

<html>

<head>

<title>My First JSP Page</title>

</head>

<%

int count = 0;

%>

<body>

Page Count is <%= ++count %>

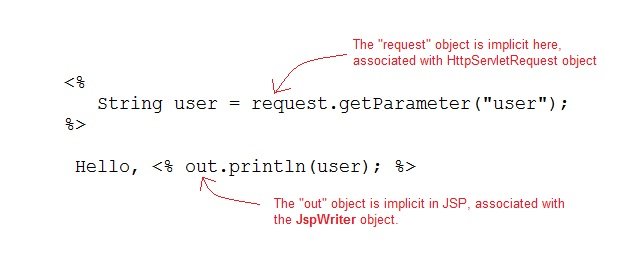
</body>

</html>

Copy

# **Implicit Objects in JSP**

JSP provide access to some implicit object which represent some commonly used objects for servlets that JSP page developers might need to use. For example you can retrieve HTML form parameter data by using **request** variable, which represent the **HttpServletRequest** object.



#### **Following are the JSP implicit object**

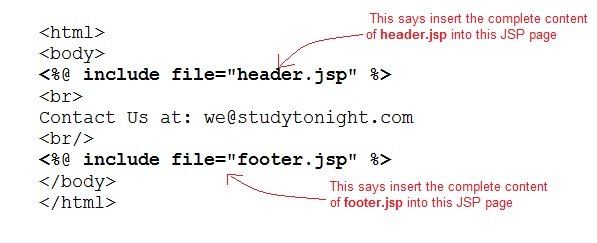
|  |  |
| --- | --- |
| **Implicit Object** | **Description** |
| **request** | The **HttpServletRequest** object associated with the request. |
| **response** | The **HttpServletRequest** object associated with the response that is sent back to the browser. |
| **out** | The **JspWriter** object associated with the output stream of the response. |
| **session** | The **HttpSession** object associated with the session for the given user of request. |
| **application** | The **ServletContext** object for the web application. |
| **config** | The **ServletConfig** object associated with the servlet for current JSP page. |
| **pageContext** | The **PageContext** object that encapsulates the enviroment of a single request for this current JSP page |
| **page** | The **page** variable is equivalent to **this** variable of Java programming language. |
| **exception** | The **exception** object represents the **Throwable** object that was thrown by some other JSP page. |

# **JSP Include Directive**

The *include* directive tells the Web Container to copy everything in the included file and paste it into current JSP file. Syntax of **include** directive is:

<%@ include file="filename.jsp" %>

Copy



### Example of include directive

**welcome.jsp**

<html>

<head>

<title>Welcome Page</title>

</head>

<body>

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

Welcome, User

</body>

</html>

Copy

**header.jsp**

<html>

<body>

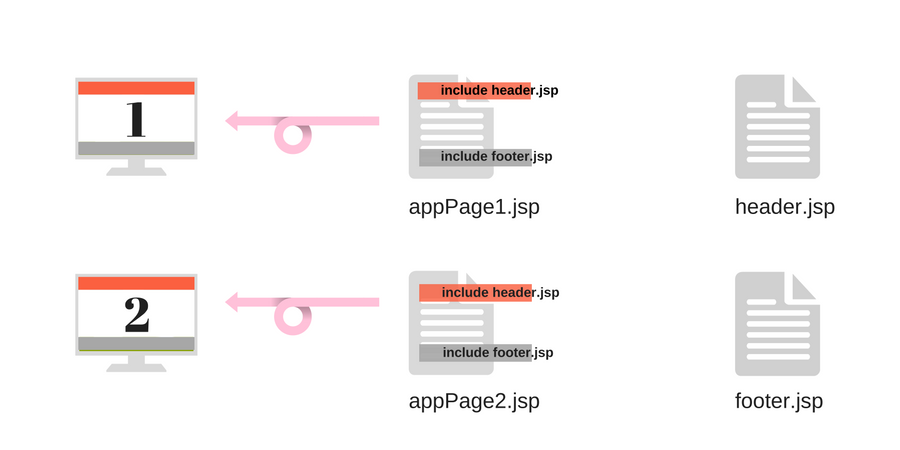
<img src="header.jpg" alt="This is Header image" / >

</body>

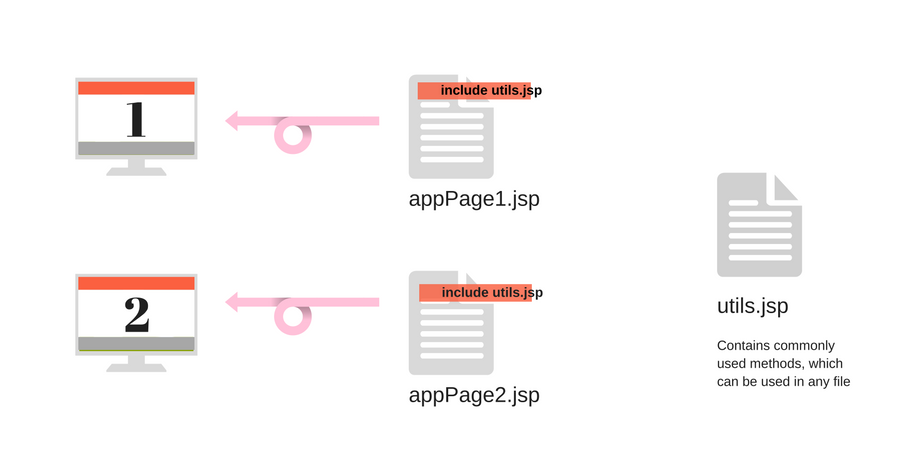
</html>

Copy

The example above is showcasing a very standard practice. Whenever we are building a web application, with webpages, all of which have the top navbar and bottom footer same. We make them as separate jsp files and include them using the include directive in all the pages. Hence whenever we have to update something in the top navbar or footer, we just have to do it at one place. Handy, isn't it?



One more standard application of include directive is, if you create a separate jsp file, with some commonly used functions, kind of like a util jsp file. Which can be included in the web pages wherever you want to use those functions.



Similarly, there are many ways in which this directive proves to be quite useful in giving a structure to your web application code.

# **JSP Taglib Directive**

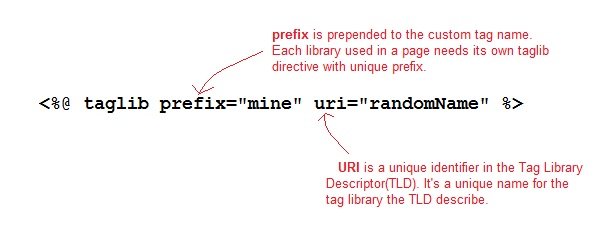
The taglib directive is used to define tag library that the current JSP page uses. A JSP page might include several tag library. JavaServer Pages Standard Tag Library (JSTL), is a collection of useful JSP tags, which provides mahy commonly used core functionalities. It has support for many general, structural tasks such as iteration and conditionals, readymade tags for manipulating XML documents, internationalization tags, and for performing SQL operations. Syntax of taglib directive is:

<%@ taglib prefix="prefixOfTag" uri="uriOfTagLibrary" %>

Copy

The prefix is used to distinguish the custom tag from other libary custom tag. Prefix is prepended to the custom tag name. Every custom tag must have a prefix.

The URI is the unique name for Tag Library.



You can name the prefix anything, but it should be unique.

### JSP: Using Taglib Directive

To use the JSTL in your application you must have the jstl.jar in your webapps /WEB-INF/lib directory. Download the jar file from [Apache Standard Taglib](http://tomcat.apache.org/taglibs/index.html) page.

There are many readymade JST Libraries available which you use to make your life easier. Following is a broad division on different groups of JST libraries :

1. Core Tags - URI → http://java.sun.com/jsp/jstl/core
2. Formatting Tags - URI → http://java.sun.com/jsp/jstl/fmt
3. SQL Tags - URI → http://java.sun.com/jsp/jstl/sql
4. XML Tags - URI → http://java.sun.com/jsp/jstl/xml
5. JSTL Functions - URI → http://java.sun.com/jsp/jstl/functions

# **JSP Exception Handling**

Exception Handling is a process of handling exceptional condition that might occur in your application. Exception Handling in JSP is much easier than Java Technology exception handling. Although JSP Technology also uses the same exception class objects.

It is quite obvious that you dont want to show error stack trace to any random user surfing your website. You can't prevent all errors in your application but you can atleast give a user friendly error response page.

## Ways to perform Exception Handling in JSP

JSP provide 3 different ways to perform exception handling:

1. Using **isErrorPage** and **errorPage** attribute of page directive.
2. Using **<error-page>** tag in **Deployment Descriptor**.
3. Using simple try...catch block.

### Example of isErrorPage and errorPage attribute

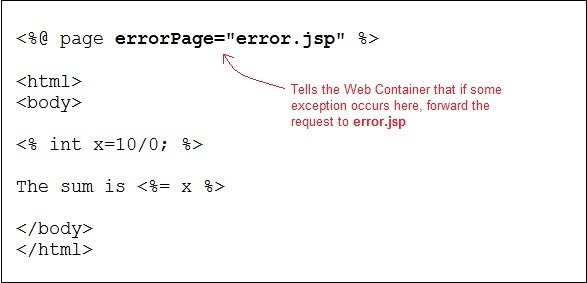
isErrorPage attribute in page directive officially appoints a JSP page as an error page.

**error.jsp**



errorPage attribute in a page directive informs the Web Container that if an exception occurs in the current page, forward the request to the specified error page.

**sum.jsp**



Whenever an exception occurs in sum.jsp page the user is redirected to the error.jsp page, where either you can display a nice message, or you can also print the exception trace into a file/database in the background, to check later what caused the error.

### Declaring error page in Deployment Descriptor

You can also declare error pages in the DD for the entire Web Apllication. Using <error-page> tag in the **Deployment Descriptor**. You can even configure different error pages for different exception types, or HTTP error code type(503, 500 etc).

**Declaring an error page for all type of exception**

<error-page>

<exception-type>java.lang.Throwable</exception-type>

<location>/error.jsp</location>

</error-page>

Copy

**Declaring an error page for more detailed exception**

<error-page>

<exception-type>java.lang.ArithmeticException</exception-type>

<location>/error.jsp</location>

</error-page>

Copy

**Declaring an error page based on HTTP Status code**

<error-page>

<error-code>404</error-code>

<location>/error.jsp</location>

</error-page>

Copy

### Using the try...catch block

Using try...catch block is just like how it is used in [Core Java](https://www.studytonight.com/java/overview-of-java.php).

<html>

<head>

<title>Try...Catch Example</title>

</head>

<body>

<%

try{

int i = 100;

i = i / 0;

out.println("The answer is " + i);

}

catch (Exception e){

out.println("An exception occurred: " + e.getMessage());

}

%>

</body>

</html>

# **JSP Standard Tag(Action Element)**

JSP specification provides **Standard**(Action) tags for use within your JSP pages. These tags are used to remove or eliminate scriptlet code from your JSP page because scriplet code are technically not recommended nowadays. It's considered to be bad practice to put java code directly inside your JSP page.

Standard tags begin with the jsp: prefix. There are many JSP Standard Action tag which are used to perform some specific task.

The following are some JSP Standard Action Tags available:

|  |  |
| --- | --- |
| **Action Tag** | **Description** |
| jsp:forward | forward the request to a new page  Usage : <jsp:forward page="Relative URL" /> |
| [jsp:useBean](https://www.studytonight.com/jsp/usebean-tag.php) | instantiates a JavaBean  Usage : <jsp:useBean id="beanId" /> |
| [jsp:getProperty](https://www.studytonight.com/jsp/getproperty-tag.php) | retrieves a property from a JavaBean instance.  Usage :  <jsp:useBean id="beanId" ... />  ...  <jsp:getProperty name="beanId" property="someProperty" .../>  Copy  Where, **beanName** is the name of pre-defined bean whose property we want to access. |
| [jsp:setProperty](https://www.studytonight.com/jsp/getproperty-tag.php) | store data in property of any JavaBeans instance.  Usage :  <jsp:useBean id="beanId" ... />  ...  <jsp:setProperty name="beanId" property="someProperty"  value="some value"/>  Copy  Where, **beanName** is the name of pre-defined bean whose property we want to access. |
| jsp:include | includes the runtime response of a JSP page into the current page. |
| jsp:plugin | Generates client browser-specific construct that makes an OBJECT or EMBED tag for the Java Applets |
| jsp:fallback | Supplies alternate text if java plugin is unavailable on the client. You can print a message using this, if the included jsp plugin is not loaded. |
| jsp:element | Defines XML elements dynamically |
| jsp:attribute | defines dynamically defined XML element's attribute |
| jsp:body | Used within standard or custom tags to supply the tag body. |
| jsp:param | Adds parameters to the request object. |
| jsp:text | Used to write template text in JSP pages and documents.  Usage : <jsp:text>Template data</jsp:text> |

# **JSP JavaBean Components**

A JavaBeans component is a Java class with the following features:

* A no-argument constructor.
* Properties defined with accessors and mutators(getter and setter method).
* Class must not define any public instance variables.
* The class must implement the **java.io.Serializable** interface.

## Example of a JavaBean

Let's take a simple Java code example to understand what do we mean when we say JavaBean,

import java.io.Serializable;

public class StudentBean implements Serializable

{

private String name;

private int roll;

// constructor

public StudentBean()

{

this.name = "";

this.roll = "";

}

// getters and setters

public void setName(String name)

{

this.name = name;

}

public String getName()

{

return name;

}

public int getRoll()

{

return roll;

}

public void setRoll(int roll)

{

this.roll = roll;

}

}

Copy

As you can see in the code above, a JavaBean is nothing but a Java class which implements the interface Serializable.

### Using a JavaBean in JSP page

JavaBeans can be used in any JSP page using the <jsp:useBean> tag, For example:

<jsp:useBean id="bean name" scope="fully qualified path of bean" typeSpec/>

Copy

### Using any JavaBean property in JSP page

JavaBeans can be used in any JSP page using the [<jsp:useBean>](https://www.studytonight.com/jsp/usebean-tag.php) tag, [<jsp:setProperty>](https://www.studytonight.com/jsp/setproperty-tag.php) tag and [<jsp:getProperty>](https://www.studytonight.com/jsp/getproperty-tag.php) tag , For example:

<jsp:useBean id="id" class="bean class name" scope="fully qualified path of bean">

<jsp:setProperty name="beans id" property="property name" value="value"/>

<jsp:getProperty name="beans id" property="property name"/>

...........

</jsp:useBean>

# **JSP jsp:useBean Tag**

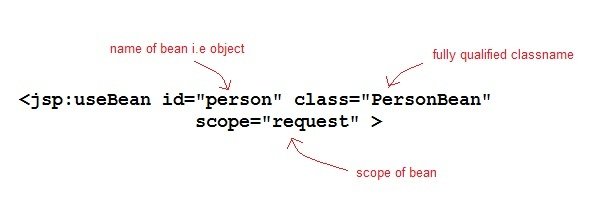
If you want to interact with a [JavaBeans](https://www.studytonight.com/jsp/javabeans-component.php) component using the Action tag in a JSP page, you must first declare a bean. The <jsp:useBean> is a way of declaring and initializing the actual bean object. By **bean** we mean JavaBean component object. Syntax of **<jsp:useBean>** tag

<jsp:useBean id = "beanName" class = "className"

scope = "page | request | session | application">

Copy

Here the **id** attribute specifies the name of the bean. **Scope** attribute specify where the bean is stored. The **class** attribute specify the fully qualified classname.



Given a useBean declaration of following :

<jsp:useBean id="myBean" class="PersonBean" scope="request" />

Copy

is equivalent to the following java code,

PersonBean myBean = (PersonBean)request.getAttribute("myBean");

if(myBean == null)

{

myBean = new PersonBean();

request.setAttribute("myBean", myBean);

}

Copy

If **jsp:useBean** tag is used with a body, the content of the body is only executed if the bean is created. If the bean already exists in the named scope, the body is skipped.

### Time for an Example

In this example we will see how <jsp:useBean> standard tag is used to declare and initialize a bean object. We will use PersonBean class as JavaBean Component.

**PersonBean.java**

import java.io.Serializable;

public class PersonBean implements Serializable

{

private String name;

public PersonBean()

{

this.name="";

}

public void setName(String name)

{

this.name = name;

}

public String getName()

{

return name;

}

}

Copy

**hello.jsp**

<html>

<head>

<title>Welcome Page</title>

</head>

<jsp:useBean id="person" class="PersonBean" scope="request" />

<body>

//Use the bean here...

</body>

</html>

Copy

Here **jsp:useBean** declares a "person" bean in the jsp page which can be used there. How to use it, modify it, we will study in coming lessons.

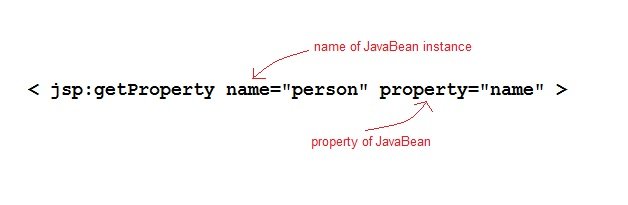
# **JSP jsp:getProperty Tag**

The getProperty tag is used to retrieve a property from a JavaBeans instance. The syntax of the getProperty tag is as follows:

<jsp:getProperty name="beanName" property="propertyName" />

Copy

The name attribute represents the name of the JavaBean instance. The property attribute represents the property of the JavaBean whose value we want to get.



### Example of getProperty with Java Bean

Following is our Java class.

PersonBean.java

import java.io.Serializable;

public class PersonBean implements Serializable

{

private String name;

public PersonBean()

{

this.name="";

}

public void setName(String name)

{

this.name = name;

}

public String getName()

{

return name;

}

}

Copy

hello.jsp

<html>

<head>

<title>Welcome Page</title>

</head>

<jsp:useBean id="person" class="PersonBean" scope="request" />

<body>

Name of Person is : <jsp:getProperty name="person" property="name" />

</body>

</html>

Copy

This will print the value of the property. What if you need to change the value of the property. Let's learn how to set value of the property in our next lesson.

# **JSP jsp:setProperty Tag**

The setProperty tag is used to store data in JavaBeans instances. The syntax of setProperty tag is:

<jsp:setProperty name="beanName" property="\*">

<!-- or -->

<jsp:setProperty name="beanName" property="propertyName">

<!-- or -->

<jsp:setProperty name="beanName" property="propertyName" param="parameterName">

<!-- or -->

<jsp:setProperty name="beanName" property="propertyName" value="propertyValue">

Copy

The name attribute specifies the name of javaBean instances. This must match the id attribute specified in the [jsp:useBean](https://www.studytonight.com/jsp/usebean-tag.php" \t "_blank) tag. The **property** attribute specifies which property of the bean to access.

### Example of setProperty with Java Bean

Following is our Java class.

PersonBean.java

import java.io.Serializable;

public class PersonBean implements Serializable

{

private String name;

public PersonBean()

{

this.name="";

}

public void setName(String name)

{

this.name = name;

}

public String getName()

{

return name;

}

}

Copy

hello.jsp

<html>

<head>

<title>Welcome Page</title>

</head>

<jsp:useBean id="person" class="PersonBean" scope="request" />

<jsp:setProperty name="person" property="name" value="Viraj" />

<body>

Name of Person is : <jsp:getProperty name="person" property="name" />

</body>

</html>

Copy

Output will be → Name of Person is : Viraj

Similarly we can have a very complex Java Bean as well, with many properties. We can easily get and set all the properties using the [jsp:useBean](https://www.studytonight.com/jsp/usebean-tag.php), [jsp:setProperty](https://www.studytonight.com/jsp/setproperty-tag.php" \t "_blank), [jsp:getProperty](https://www.studytonight.com/jsp/getproperty-tag.php" \t "_blank).

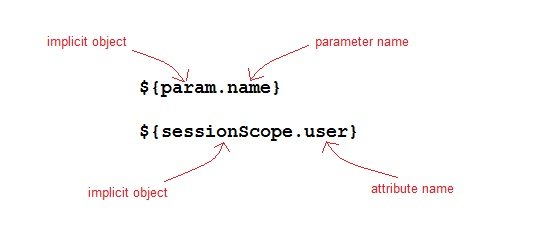
# **JSP Expression Language**

Expression Language(EL) was added to JSP 2.0 specification. The purpose of EL is to produce scriptless JSP pages. The syntax of EL in a JSP is as follows:

${expr}

Copy

Here expr is a valid EL expression. An expression can be mixed with static text/values and can also be combined with other expressions to form larger expression.



## How EL expression is used?

EL expression can be used in two ways in a JSP page

1. As attribute values in standard and custom tags. Example:
2. <jsp:include page="${location}">

Copy

Where location variable is separately defines in the jsp page.

Expressions can also be used in [jsp:setProperty](https://www.studytonight.com/jsp/setproperty-tag.php" \t "_blank) to set a properties value, using other bean properties like : If we have a bean named Square with properties length, breadth and area.

<jsp:setProperty name="square" property="area" value="${square.length\*square.breadth}" />

Copy

1. To output in HTML tag:
2. <h1>Welcome ${name}</h1>

Copy

To deactivate the evaluation of EL expressions, we specify the isELIgnored attribute of the page directive as below:

<%@ page isELIgnored ="true|false" %>

Copy

### JSP EL Implicit Objects

Following are the implicit objects in EL :

|  |  |
| --- | --- |
| **Implicit Object** | **Description** |
| pageContext | It represents the PageContext object. |
| pageScope | It is used to access the value of any variable which is set in the Page scope |
| requestScope | It is used to access the value of any variable which is set in the Request scope. |
| sessionScope | It is used to access the value of any variable which is set in the Session scope |
| applicationScope | It is used to access the value of any variable which is set in the Application scope |
| param | Map a request parameter name to a single value |
| paramValues | Map a request parameter name to corresponding array of string values. |
| header | Map containing header names and single string values. |
| headerValues | Map containing header names to corresponding array of string values. |
| cookie | Map containing cookie names and single string values. |

### Example of JSP EL

Let's take a simple example for understanding the JSP expression language,

**index.jsp**

<form method="POST" action="welcome.jsp">

Name <input type="text" name="user" >

<input type="submit" value="Submit">

</form>

Copy

**welcome.jsp**

<html>

<head>

<title>Welcome Page</title>

</head>

<body>

<h1>Welcome ${param.name}</h1>

</body>

</html>

Copy

### Arithmetic Operations available in EL

Following are the arithmetic operators avilable in EL:

|  |  |
| --- | --- |
| **Arithmetic Operation** | **Operator** |
| Addition | + |
| Substraction | - |
| Multiplication | \* |
| Division | / and div |
| Remainder | % and mod |

### Logical and Relational Operators available in EL

Following are the logical operator and comparators avilable in EL:

|  |  |
| --- | --- |
| **Logical and Relational Operator** | **Operator** |
| Equals | == and eq |
| Not equals | != and ne |
| Less Than | < and lt |
| Greater Than | > and gt |
| Greater Than or Equal | >= and ge |
| Less Than or Equal | <= and le |
| And | && and and |
| Or | || and or |
| Not | ! and not |