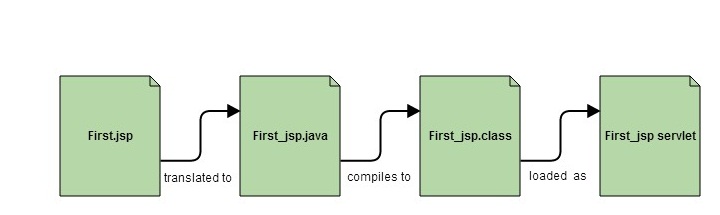
**Introduction to JSP**

**JSP** technology is used to create web application. It focuses more on presentation logic of the web apllication.**JSP** pages are easier to maintain then a **Servlet**. JSP pages are opposite of Servlets. Servlet adds HTML code inside Java code while JSP adds Java code inside HTML. Everything a Servlet can do, a JSP page can also do it.

JSP enables us to write HTML pages containing tags that run powerful Java programs. **JSP separates presentation and business logic** as Web designer can design and update JSP pages without learning the Java language and Java Developer can also write code without concerning the web design.

**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.



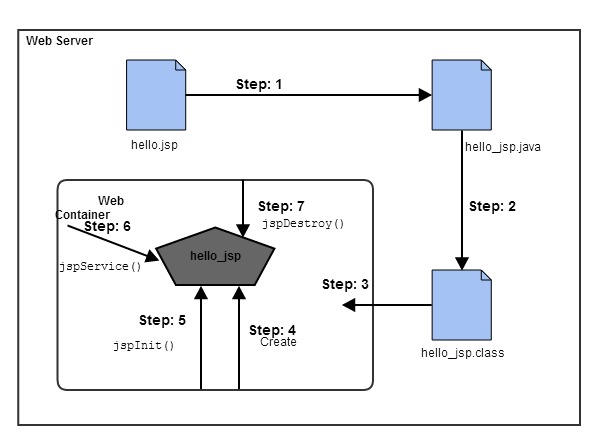
**Advantage of JSP**

* Easy to maintain
* 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 consists of following 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

<html>

<head>

<title>My First JSP Page</title>

</head>

**<%**

**int count = 0;**

**%>**

<body>

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

</body>

</html>

**The above JSP page 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>");

}

}

**Creating a JSP Page**

A JSP page look similar to a HTML page, but a JSP page can also have Java Technology code. We can put regular Java Code in a JSP using a **scriplet tag** which start with **<%** and end with **%>**. JSP page are used to make dynamic response.

**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 on 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 ok
* Write something in your JSP file
* To run your project, right click on Project, select Run --> Run on Server
* To start the server, Choose existing server name and click on finish
* See the Output

### 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.

#### 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 follow

<%-- JSP comment --%>

### Scriptlet Tag

Scriptlet Tag allows you to write java code inside JSP page. Syntax of Scriptlet Tag is as follows :

<% *java code* %>

#### 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>

### 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 %>

**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.

* *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,java.net.\*" %>

***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.

***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.

### 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

<%= JavaExpression %>

**When the Container sees this**

<%= (2\*5) %>

**It turns it into this:**

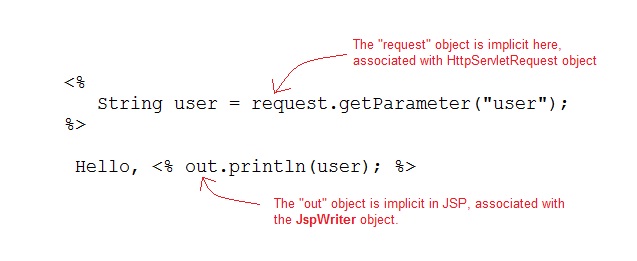
out.print((2\*5));

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

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

### 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.



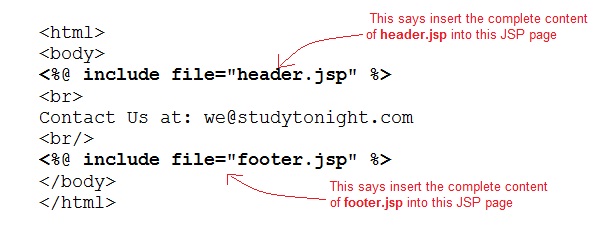
#### Following are the JSP implicit object

|  |  |
| --- | --- |
| 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 environment 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. |

### Include Directive

The *include* directive tells the Web Container to copy everything in the included file and paste it into curre;nt JSP file. Syntax of **include** directive.

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



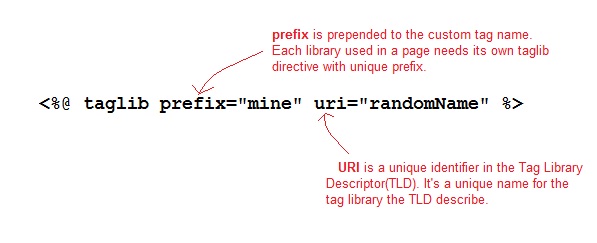
### Taglib Directive

The **taglib** directive is used to define tag library that the current JSP page uses. A JSP page might include several tag library. We will look into it in more details in Custom Tag section. Syntax of taglib directive:

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

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.

**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 object.

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

**Ways to perform exception handling in JSP**

JSP provide two different way to perform exception handling.

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

### 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 tags which are used to perform some specific task.

The following are some JSP Standard Action Tag available:

|  |  |
| --- | --- |
| Action Tag | Description |
| jsp:forward | forward the request to a new page |
| jsp:getProperty | retrieve a property from a JavaBean instance. |
| jsp:include | include the runtime response of a JSP page. |
| jsp:plugin | Generates client browser-specific construct that makes an OBJECT or EMBED tag for the Java Applets |
| jsp:element | Defines XML elements dynamically |
| jsp:attribute | defines dynamically defined XML element's attribute |
| jsp:param | Adds parameters to the request |
| jsp:fallback | Supplies alternate text if java applet is unavailable on the client |
| jsp:body | Used within standard or custom tags to supply the tag body. |
| jsp:text | Use to write template text in JSP pages and documents. |
| jsp:setProperty | Store data in JavaBeans instance. |
| jsp:useBean | instantiates a JavaBean |

**JavaBean Components**

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

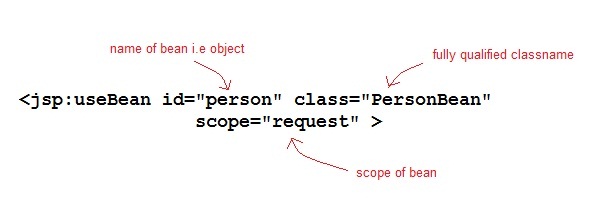
* A no-argument constructor.
* Properties defined with accessors and mutators(getter and setter method).
* Class must not define any public instance variables.
* The class must implements the **java.io.Serializable** interface.

### jsp:useBean Tag

If you want to interact with a JavaBeans 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">

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.(along with the package name)



Given a useBean declaraction of following

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

is equivalent to the following java code

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

if(myBean=null)

{

myBean = new PersonBean();

request.setAttribute("myBean",myBean);

}

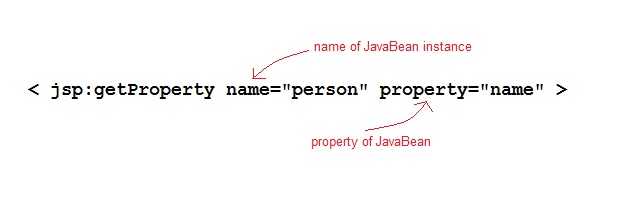
If jsp:useBean tag is used with a body, the content of the body is only executed if the bean is created. If the bean already exists in the named scope, the body is skipped.

### jsp:getProperty Tag

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

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

The name attribute represents the name of the JavaBeans instance. The property attribute represents the property of the JavaBean.



Given a getProperty tag of following

<jsp:getProperty name="person" property="name" />

is equivalent to following java code

out.print(person.getName());

### 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" >

The **name** attribute specifies the name of javaBean instances. This must match the **id** attribute used in the **useBean** tag. The **property** attribute specifies the property within the bean.

Given a setProperty tag of following

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

is equivalent to following java code

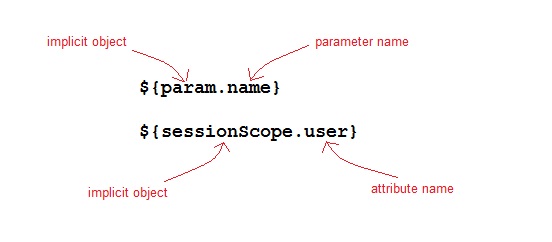
person.setName(request.getParameter("name"));

**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}

Here **expr** is a valid EL expression. The expression can be mixed with static text and can be combined with other expression to form larger expression.



**How EL expression is use**

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}">
3. As a output with HTML tag
4. <h1>Welcome ${name}</h1>

**EL Implicits Objects**

The following are the implicit object in EL

|  |  |
| --- | --- |
| Implicit Object | Description |
| pageContext | Represent the PageContext object. |
| pageScope | It is used to access the value of variable which is set in the **Page** scope |
| requestScope | It is used to access the value of variable which is set in the **request** scope. |
| sessionScope | It is used to access the value of variable which is set in the **session** scope |
| applicationScope | It is used to access the value of 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 string arrays. |
| header | Map containing header names and single string values. |
| headerValues | Map containing header names to corresponding string arrays. |
| cookie | Map containing cookie names and single string values. |

**Arithmetic Operation available in EL**

The following are the five arithmetic operator available in EL.

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

**Logical and Relational Operator available in EL**

The following are the logical operator and Comparators available 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 |