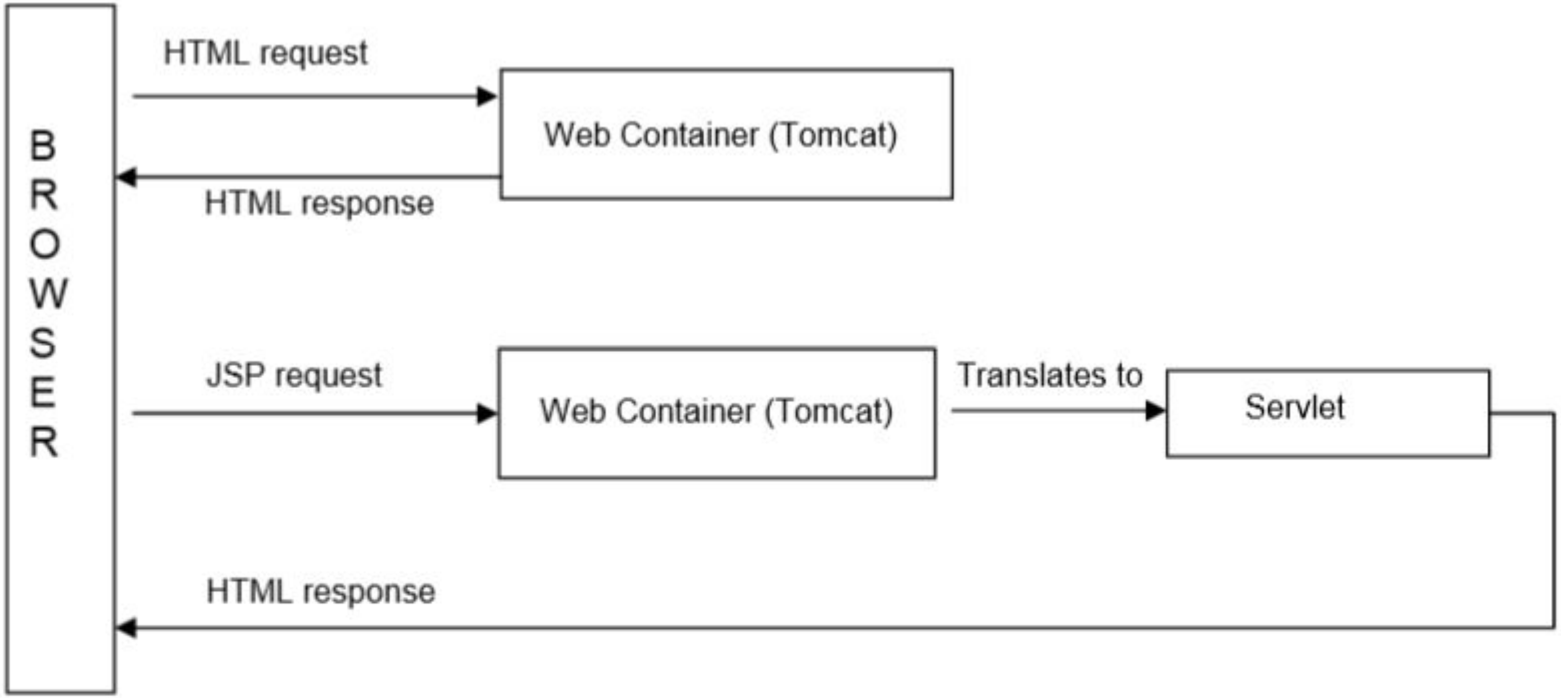
JSP
Java Server Page

JSP Basics

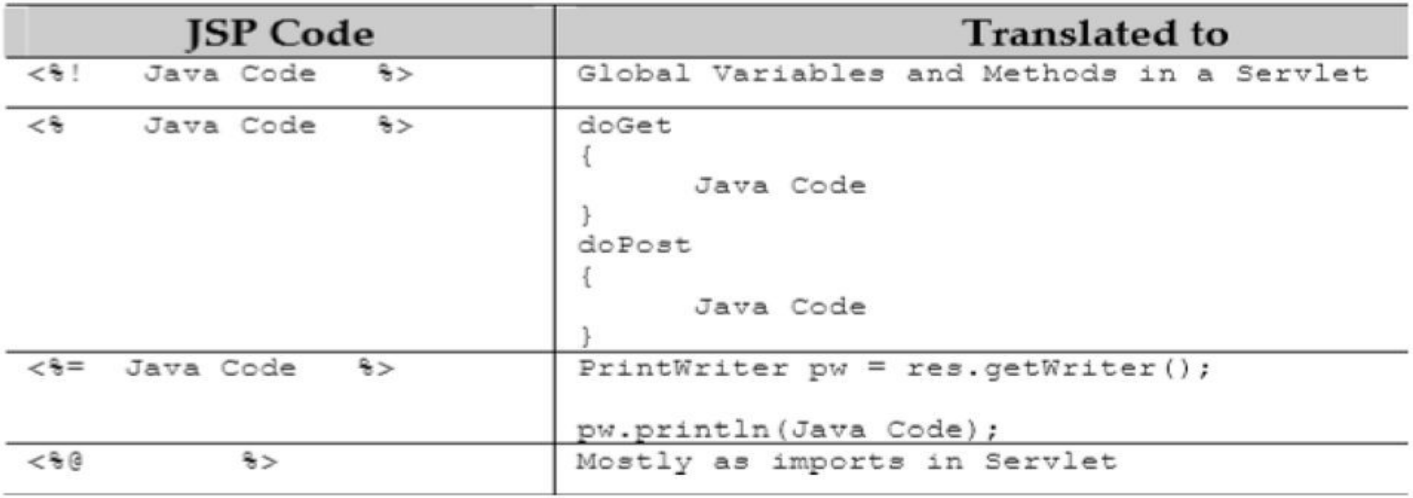
- A typical JSP page very much looks like html page with all the html markup except that you also see Java code in it. So, in essence,
  - HTML + Java = JSP
- Therefore, JSP content is a mixed content, with a mixture of HTML and Java.
- Can we save this mixed content in a file with “.html” extension?
- No we can’t, because the html formatter will also treat the Java code as plain text which is not what we want.
- We want the Java code to be executed and display the dynamic content in the page. For this to happen, we need to use a different extension which is the “.jsp” extension.
- Good. To summarize, a html file will only have html markup, and a jsp file will have both html markup and Java code. Point to be noted.
-

- JSP is a server side technology which is used for creating web application.
- It is used to create dynamic web content.
- JSP tags are used to insert JAVA code into HTML pages.
- It is an advanced version of Servlet Technology.
- In JSP, Java code can be inserted in HTML/ XML pages or both.
- JSP is first converted into servlet by JSP container before processing the client’s request.


B R O W S E R
HTML request
Web Container (Tomcat)
HTML response
JSP request
Web Container (Tomcat)
Translates to
Servlet
HTML response

- When the browser requests for html file, the web container simply responds with a html response without any processing.
- However, when the browser sends a JSP page request, the web container assumes that the JSP page might include Java code, and translates the page into a Servlet.
- The servlet then processes the Java code, and returns the complete html response including the dynamic content generated by the Java code.
- Though the web container does the translation of JSP to Servlet, we need to know how it does this translation.
- For a web container to translate JSP, it needs to identify from the JSP page, as to which is HTML markup and which is Java code.
- According to J2EE specification, a JSP page must use special symbols as placeholders for Java code.

- In JSP, we use four different types of placeholders for Java code. Following table shows these placeholders along with how the web container translates the Java code with them.



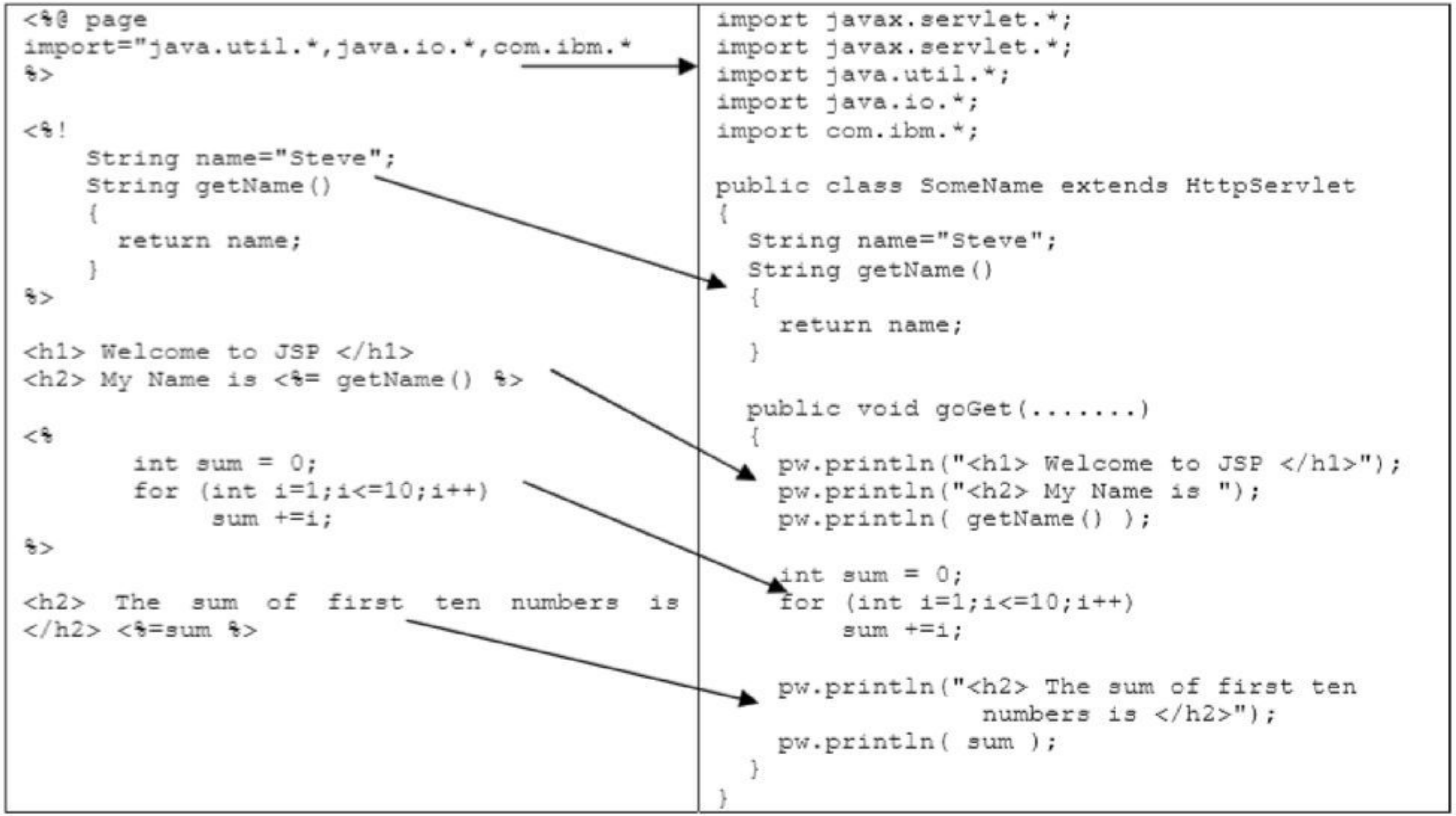
JSP Code	Translated to
`<%! Java Code %>`	`Global Variables and Methods in a Servlet`

`<% Java Code %>`	`doGet`
	`{`
	`Java Code`
	`}`
	`doPost`
	`{`
	`Java Code`
	`}`

`<%= Java Code %>`	`PrintWriter pw = res.getWriter();`
	
	`pw.println(Java Code);`

`<%@ %>`	`Mostly as imports in Servlet`

- To better understand the above four place holders, let’s take a sample JSP page and see how the web container translates it into a Servlet. See the following JSP and Servlet code snippets.



```
<%@ page
import="java.util.*,java.io.*,com.ibm.*
%>

<%!
    String name="Steve";
    String getName()
    {
      return name;
    }
%>

<h1> Welcome to JSP </h1>
<h2> My Name is <%= getName() %>

<%
       int sum = 0;
       for (int i=1;i<=10;i++)
            sum +=i;
%>

<h2> The sum of first ten numbers is
</h2> <%=sum %>
```

```
import javax.servlet.*;
import javax.servlet.*;
import java.util.*;
import java.io.*;
import com.ibm.*;

public class SomeName extends HttpServlet
{
  String name="Steve";
  String getName()
  {
    return name;
  }

  public void goGet(.......)
  {
    pw.println("<h1> Welcome to JSP </h1>");
    pw.println("<h2> My Name is ");
    pw.println( getName() );

    int sum = 0;
    for (int i=1;i<=10;i++)
        sum +=i;

    pw.println("<h2> The sum of first ten
                     numbers is </h2>");
    pw.println( sum );
  }
}
```

JSP Scripting

- 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 is a piece of Java code that represents processing logic to generate the dynamic content wherever needed in the page.
- Syntax is as follows:

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

Example:

```
<html>
<body>
	<%
		int a=10; int b=20; int c=30;
		out.println(a+b+c);
	%>
</body>
</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

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

Example:

```
<html>

<body>

        <%= "welcome to jsp" %>

</body>

</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.

Syntax:

```
<%! Java code %>
```

Example:

```
<html>

<body>

        <%! int data=50; %>

        <%= "Value of the variable is:"+data %>

</body>

</html>
```

JSP Directives

- Directives are used for declaring classes, session usage etc., and does not produce any response to the client.

  Syntax:

  ```
  <%@ Java code %>
  ```

- A directive uses attributes for declarations. There are 3 types of directives as listed below:
  - page directive
  - include directive
  - taglib directive

The Page Directive

- This directive is used to declare things that are important to the entire JSP page.
- The syntax for this directive is shown below
  - <%@ page attribute list %>

Attribute List

- import
- contentType
- extends
- info
- isThreadSafe
- 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:

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

Today is: <%= new Date() %>
```

2) contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.

Example:

```
<%@ page contentType=application/msword %>

Today is: <%= new java.util.Date() %>
```

3)info

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

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

4)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.

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

5)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.

```
<%@ page errorPage="myerrorpage.jsp" %>
 <%= 100/0 %>
```

6)isErrorPage

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

```
<%@ page isErrorPage=true %>
 Sorry an exception occured!<br/>
The exception is: <%= exception %>
```

Include Directive

This directive is used to include the response of another resource (JSP or html) at the point where the directive is placed in the current JSP. Its usage is shown below.

```
syntax:  <%@ include file=”file name” %>
```

Example:

```
<HTML>

<BODY>

<h4> This is the response from the current JSP page</h4>

<h3> Following is the response from another JSP </h3>

<hr/>

<%@ include file=”/jsps/PageDirectiveDemo.jsp” %>

<hr/>

</BODY>

</HTML>
```

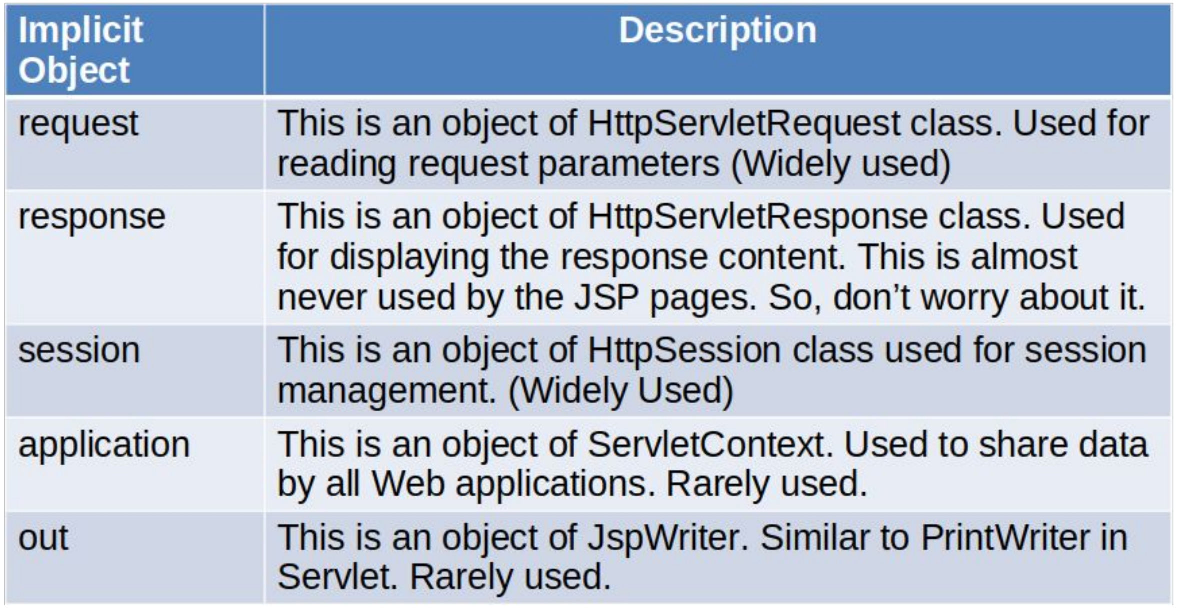
The taglib directive

- This directive allows the JSP page to use custom tags written in Java.
- Custom tag definitions are usually defined in a separate file called as Tag Library Descriptor.
- For the current JSP to use a custom tag, it needs to import the tag library file which is why this directive is used. Following is how taglib directive is used.
    - <%@ taglib uri=”location of definition file” prefix=”prefix name” %>

```
<html>
<body>

<%@ taglib uri = “./location/tags” prefix=”mytag”%>

<mytag:currentDate/>

</body>
</html>
```

Implicit Objects

As the name suggests every JSP page has some implicit objects that it can use without even declaring them.



Implicit Object	Description
request	This is an object of HttpServletRequest class. Used for reading request parameters (Widely used)
response	This is an object of HttpServletResponse class. Used for displaying the response content. This is almost never used by the JSP pages. So, don't worry about it.
session	This is an object of HttpSession class used for session management. (Widely Used)
application	This is an object of ServletContext. Used to share data by all Web applications. Rarely used.
out	This is an object of JspWriter. Similar to PrintWriter in Servlet. Rarely used.

Java Bean

- A Java Bean is a java class that should follow following conventions:
    - It should have a no-arg constructor.
    - It should be Serializable.
    - It should provide methods to set and get the values of the properties, known as getter and setter methods.
- Why use Java Bean?
    - It is a reusable software component.
    - A bean encapsulates many objects into one object, so we can access this object from multiple places.
    - Moreover, it provides the easy maintenance.

JavaBeans are **classes that encapsulate many objects into a single object** (the bean).

Java Beans in JSP

- One of the good practices while writing JSP is to isolate the presentation logic (HTML) from business logic (Java code).
- A typical JSP page should have as minimum business logic as possibly can. If this is the case, where should the business logic be?
- The business logic will be moved into external entities which will then be accessed from within JSP page. These external entities are nothing but Java beans.
- A Java bean is a simple class with getters and setters.
- Using Java beans in JSP offers whole lot of flexibility and avoids duplicate business logic.
- JSP technology uses standard actions for working with beans. So, let’s see how we can use beans in JSP and simplify our life.

1. <jsp:useBean>
2. <jsp:setProperty>
3. <jsp:getProperty>

jsp:useBean

- This action is used by the web container to instantiate a Java Bean or locate an existing bean.
- The web container then assigns the bean to an id which the JSP can use to work with it.
- The Java Bean is usually stored or retrieved in and from the specified scope.
- The syntax for this action is shown below:
- <jsp:useBean id=”bean name” class=”class name” scope=”scope name”/> where,
  - id - A unique identifier that references the instance of the bean
  - class - Fully qualified name of the bean class
  - scope – The attribute that defines where the bean will be stored or retrieved from; can be request or session (widely used)

- Consider the following declaration.
    - <jsp:useBean id = “myCustomer” class=”beans.Customer” scope=”session” />
- With the above declaration, following is what the web container does.
    - Tries to locate the bean with the name myCustomer in session scope. If it finds one, it returns the bean to the JSP.
    - If the bean is not found, then the container instantiates a new bean, stores it in the session and returns it to the JSP.
- If the scope is session, then the bean will be available to all the requests made by the client in the same session.
- If the scope is request, then the bean will only be available for the current request only.

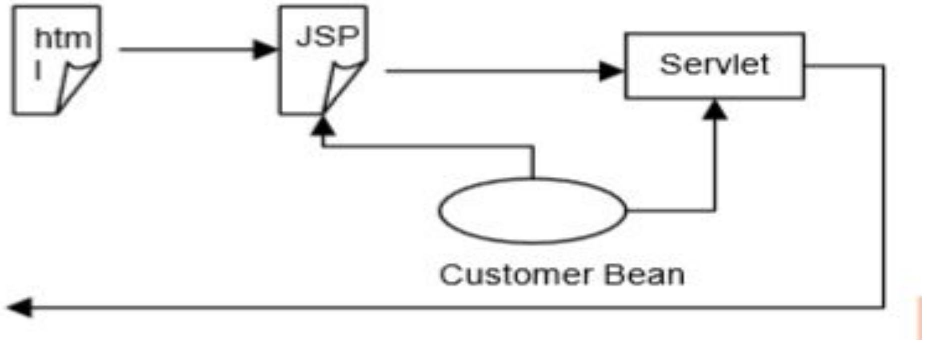
jsp:setProperty

- This action as the name suggests is used to populate the bean properties in the specified scope. Following is the syntax for this action.
- <jsp:setProperty name ="bean name" property ="property name" value= "data" />
- For instance, if we need to populate a bean whose property is firstName with a value John we use this action as shown below:
- <jsp:setProperty name = "myCustomer" property ="firstName" value= "John" />

jsp:getProperty

- This standard action is used to retrieve a specific property from a bean in a specified scope.
- Following is how we can use this action to retrieve the value of firstName property in a bean identified by myCustomer in the session scope.
- <jsp:getProperty name=”myCustomer” property=”firstName” scope=”session” />
- There are two common scenarios with using JavaBeans in JSP.
  - Scenario 1:
    - JSP collects the data from the client, populate the bean's properties and stores the bean in request or session scope. This bean will then be used by another server side component to process the data.
  - Scenario 2:
    - A Server side component loads a Java Bean with all the information and stores it in the request or session. The JSP will then retrieve the bean and displays the information to the client.

- In scenario 1, JSP uses the bean to collect the data into it, while in scenario 2, it uses the bean to read the data from it to display.
- The following example demonstrates scenario 1 in which a html form posts the data to a JSP page. The JSP page will then collect the data and store it in a Java Bean named Customer in session scope.


html
JSP
Servlet
Customer Bean