Chapter 9
JSP - I

9.1 Introduction

Instead of *static* contents that are indifferent, *Java Servlet* was introduced to generate *dynamic* web contents that are customized according to users' requests (e.g. in response to queries and search requests). However, it is a pain to use a Servlet to produce a presentable HTML page (via the out.prinltn() programming statements). It is even worse to maintain or modify that HTML page produced. Programmers, who wrote the servlet, may not be a good graphic designer, while a graphic designer does not understand Java programming.

JavaServer Pages (JSP) is a *complimentary* technology to Java Servlet which facilitates the mixing of dynamic and static web contents. JSP is Java's answer to the popular Microsoft's *Active Server Pages*(ASP). JSP, like ASP, provides a elegant way to mix static and dynamic contents. The main page is written in regular HTML, while special tags are provided to insert pieces of Java programming codes. The *business programming logic* and the *presentation* are cleanly separated. This allows the programmers to focus on the business logic, while the web designer to concentrate on the presentation.

Write Once Run Anywhere : JSP technology brings the "**Write Once, Run Anywhere**" paradigm to interactive Web pages. JSP pages can be moved easily across platforms, and across web servers, without any changes.

JSP Request Model

Now let's take a look at how HTTP requests are processed under the JSP model. In the basic request model, a request is sent directly to a JSP page. Figure 1 illustrates the flow of information in this model. JSP code controls interactions with JavaBeans components for business and data logic processing, and then displays the results in dynamically generated HTML mixed with static HTML code.

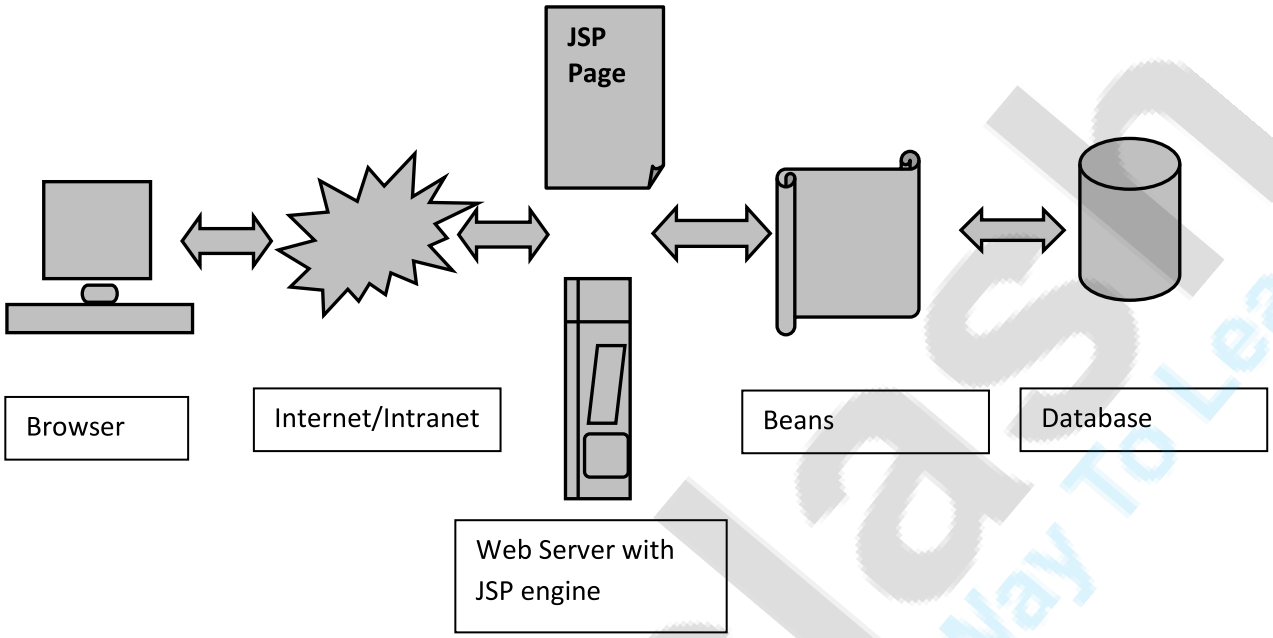


Figure 1. Basic JSP request model

The beans depicted can be JavaBeans or EJB components. Other, more complicated request models include calling out to other JSP pages or Java servlets from the requested JSP page.

9.2 Advantages and Disadvantages of JSP

How Java Server Pages works?

A page using Java Server Pages is executed during the query, by a JSP engine (generally running with a Web server or an application server). The JSP model is derived from the one used for Java servlets (JSP are indeed a way to write servlets). It is a Java class derived from HttpServlet class, making use of using `doGet()` and `doPost()` to return an HTTP response.

When a user calls a JSP page, the Web server calls the JSP engine which creates a Java source code from the JSP script and compile the class to provide a compiled file (with the *.class* extension).

Note that: the JSP engine checks if the date of the *.jsp* file corresponds to the .class file. The JSP engine will convert and compile the class, only if the JSP script has been updated. Thus, the fact that the compilation only takes place when the JSP script is updated, makes JSP, one of the fastest technologies to create dynamic pages.

Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offer several advantages in comparison with the CGI.

Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having a separate CGI files.

JSP are always compiled before it's processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.

JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including JDBC, JNDI, EJB, JAXP etc.

JSP pages can be used in combination with servlets that handle the business logic, the model supported by Java servlet template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

Advantages and Disadvantages of JSP TECHNOLOGY

ADVANTAGES OF JSP TECHNOLOGY

The main advantage of the JSP method is that the output is standard HTML and is therefore compact and universally readable in any browser. HTML requires little from the client except a compatible browser. There is no JVM running on the client, so there is no requirement for a set of Java runtime files or Java application files on the local PC.

The presentation look-and-feel of a page is embedded in HTML tags and cascading style sheets (an HTML facility for changing the appearance and formatting of common tags in a standardized way). Since the HTML tags are directly embedded in the JSP source file, you can split the development work. A web graphics designer can create the template look-and-feel for a page, while the dynamic JSP-specific sections would be developed by a Java programmer. Merging the work is just a matter of embedding one into the other. JDeveloper provides the tools to create and test the JSP code. The web designer would work in another tool such as Microsoft FrontPage or Macromedia Dreamweaver.

DISADVANTAGES OF JSP TECHNOLOGY

The main advantage of JSP pages—that they output lightweight HTML—is also the main disadvantage. You do not use the feature-rich Swing (or AWT) controls to construct the user interface. The HTML language is not a programming language as such and has fewer features than the Swing controls for creating a user interface. In addition, simple functions such as scrolling down in a list of records, deleting a record, or changing the way information is sorted requires a refresh of the page. You can embed JavaScript in the HTML page to enhance functionality, but this solution requires that the JavaScript that you use be supported by the ability of your users' browsers to interpret it correctly.

Advantages of JSP

1. HTML friendly simple and easy language and tags.
2. Supports Java Code.
3. Supports standard Web site development tools.

Disadvantages of JSP

1. As JSP pages are translated to servlets and compiled, it is difficult to trace errors occurred in JSP pages.
2. JSP pages require double the disk space to hold the JSP page.
3. JSP pages require more time when accessed for the first time as they are to be compiled on the server.

9.3 JSP v/s Servlets

JSPs are compiled to servlets, effectively allowing you to produce a servlet by just writing the HTML page, without knowing Java.

If you write servlets, you need to know Java to write the servlet body, and HTML to write the output.

The real advantage of JSP is the use of special tags that allow you to call Java beans directly, and the ability to produce your own custom tags (in Java) to do stuff that you can't do with a single call to a Java bean (e.g. inserting a list of items into the HTML, which needs a loop construct).

The idea of JSP is that a development house can put a team of specialists onto developing their web applications. A HTML web page writer/designer can write the JSP without knowing Java, using beans and custom tags written by Java experts.

Unfortunately, custom tags are a bit tedious to produce, and for very simple stuff, like simple loops, or stuff that's unlikely to get re-used, they can seem like overkill.

So JSP allows you to put real Java directly into the HTML inside special tags.

All this Java code will be executed before the HTML is sent to the client, and is generally used to modify the HTML page.

Ironically, individuals producing JSP applications on their own often end up with a JSP containing HTML, JavaScript, and Java all in the one file, which can be a little confusing, especially with all the JSP tags scattered around!

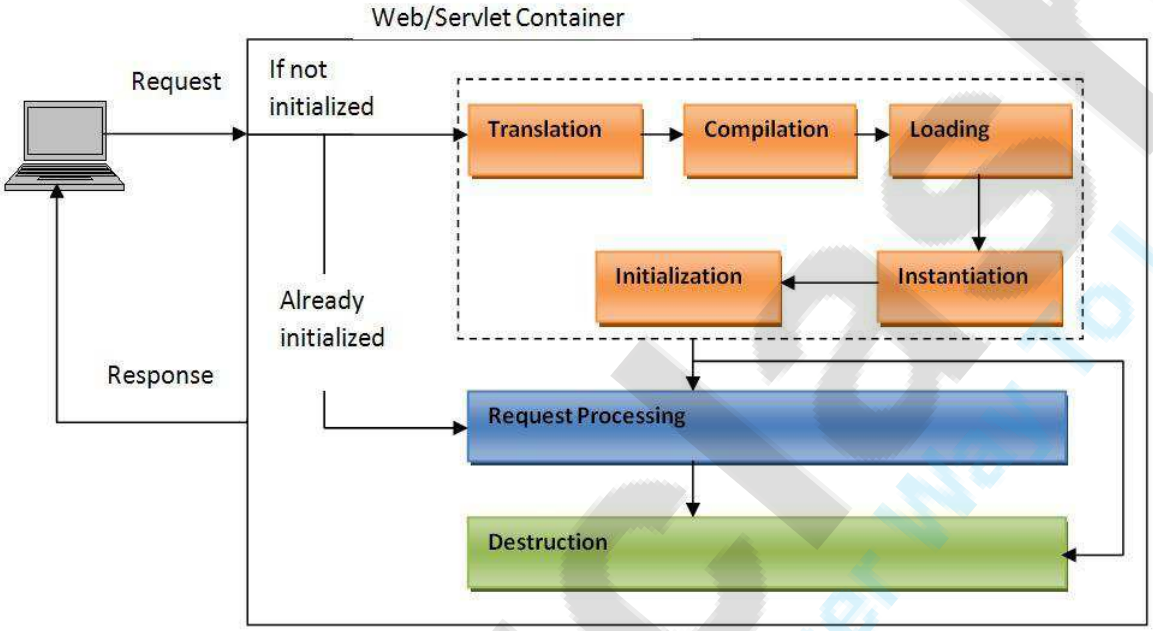
JSP	Servlets
JSP is a webpage scripting language that can generate dynamic content.	Servlets are Java programs that are already compiled which also creates dynamic web content.
In MVC(Model View Controller), jsp act as a view.	In MVC(Model View Controller), servlet act as a controller.
It's easier to code in JSP than in Java Servlets.	Its little much code to write here.
JSP are generally preferred when there is not much processing of data required.	servlets are best for use when there is more processing and manipulation involved.
JSP run slower compared to Servlet as it takes compilation time to convert into Java Servlets.	Servlets run faster compared to JSP.
The advantage of JSP programming over servlets is that we can build custom tags which can directly call Java beans.	There is no such custom tag facility in servlets.
We can achieve functionality of JSP at client side by running JavaScript at client side.	There are no such methods for servlets.

9.4 Lifecycle of JSP

When a web container or servlet container receives a request from client for a jsp page, it takes the jsp through its various life cycle phases, and then returns the response to the client. What all things these containers should support, is defined by the jsp and servlet specifications. The web containers can be a part of web servers, e.g. tomcat, and application servers.

Following diagram shows the different stages of life cycle of jsp. Broadly, these stages can be classified into three.

- Instantiation
- Request Processing
- Destruction



9.4.1 Instantiation:

When a web container receives a jsp request (may be first or subsequent), it checks for the jsp's servlet instance. If no servlet instance is available or if it is older than the jsp, then, the web container creates the servlet instance using following stages.

a) Translation
b) Compilation
c) Loading
d) Instantiation
e) Initialization

a) Translation:

Web container translates (converts) the jsp code into a servlet code. This means that jsp is actually a servlet. After this stage, there is no jsp, everything is a servlet. This task will create a complete jsp page, by considering all included components. Here on, the static content and dynamic contents are treated differently. The resultant is a java class instead of an html page (which we wrote). This is how the structure of a jsp compiled into a java class will be.

```
        package org.apache.jsp.WEB_002dINF.jsp;

        import javax.servlet.*;

        import javax.servlet.http.*;

        import javax.servlet.jsp.*;

        public final class firstJsp_jsp extends
org.apache.jasper.runtime.HttpJspBase

        implements org.apache.jasper.runtime.JspSourceDependent {

        private static final JspFactory _jspxFactory =
JspFactory.getDefaultFactory();

        .............

        .............

        public Object getDependants() {

        return _jspx_dependants;

        }

        public void _jspInit() {

        .............

        .............

        }

        public void _jspDestroy() {

        .............

        .............

        }

        public void _jspService(HttpServletRequest request,
```

```
HttpServletResponse response)

 throws java.io.IOException, ServletException {

.....

.....

}

.....

.....

}
```

b)Compilation:

The generated servlet is compiled to validate the syntax. As it is a java class, the compilation is done using javac command. This will generate the byte code to be run on JVM.

c)Loading:

The compiled byte code is loaded by the class loader used by web container. This is a standard process of using any java class.

d)Instantiation:

In this step, instance of the servlet class is created so that it can serve the request.

e)Initialization:

Initialization is done by calling the `jspInit()` method. This is one time activity at the start of the initialization process. Initialization will make the ServletContext and ServletConfig objects available. One can access many attributes related to the web container and the servlet itself. After initialization the servlet is ready to process requests.

9.4.2 Request Processing:

Entire initialization process is done to make the servlet available in order to process the incoming request. `jspService()` is the method that actually processes the request. It prints the response in html (any other) format, using 'out' object.

9.4.3 Destroy:

Whenever the server is shutting down or when the server needs memory, the server removes the instance of the servlet. The destroy method `jspDestroy()` can be called by the server after initialization and before or after request processing. Once destroyed the jsp needs to be initialized again.

Just to summarize, web container handles incoming requests to a jsp by converting it into a servlet and then by using this servlet to generate the response. Also when the server shuts down, the container needs to clear the instances.

Controlling Life Cycle:

We understood that the jsp initialization phase happens when the first request hits the web container. This might take more time to present the response to the first user of jsp. It should be possible to keep initialized servlets ready to receive first request and immediately return response. This can be achieved by using the attributes provided by web and application server. The attribute can be "load-on-startup", "pre compile" etc.

9.5 Comments

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out" part of your JSP page.

Following is the syntax of JSP comments:

```
<%-- This is JSP comment --%>
```

Following is the simple example for JSP Comments:

```
<html>

<head><title>A Comment Test</title></head>

<body>
```

```
<h2>A Test of Comments</h2>

<%-- This comment will not be visible in the page source --%>

</body>

</html>
```

This would generate following result:

There are a small number of special constructs you can use in various cases to insert comments or characters that would otherwise be treated specially. Here's a summary:

Syntax	Purpose
<%-- comment --%>	A JSP comment. Ignored by the JSP engine.
<!-- comment -->	An HTML comment. Ignored by the browser.
<\%	Represents static <% literal.
%\>	Represents static %> literal.
\'	A single quote in an attribute that uses single quotes.
\"	A double quote in an attribute that uses double quotes.

9.6 JSP documents

These are JSP pages written as namespace-aware XML documents.

The container needs to distinguish classic JSP pages and JSP documents, so that it can apply XML well-formedness tests to the latter. Identification of JSP documents can occur in three ways:

- via the `<is-xml>` element within a `<jsp-property-group>` element in web.xml
- via the `.jspx` extension
- the top element is `<jsp:root>`

9.7 JSP elements

If you want to put any java code in that file then you can put it by the use of jsp tags(elements).

There are mainly three groups of jsp tags(elements) available in java server page:

1. JSP scripting elements

    1.1 JSP scriptlet tag

    1.2 JSP expression tag

    1.3 JSP declaration tag

    1.4 Comment tag

2. JSP directive elements

    2.1 page directive

    2.2 include directive

    2.3 taglib directive

3. JSP standard action elements

9.7.1 JSP scripting elements

The scripting elements provides the ability to insert java code inside the jsp. 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 four types of scripting elements:

- scriptlet tag
- expression tag
- declaration tag
- Comment tag

9.7.1.1 scriptlet tag

A scriptlet is a valid code in java and is placed in the `jspService()` method of the JSP engine at the time of running. The scriplet syntax is -

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

There are variables available exclusively for the scriplets. They are request, response, out,session and pageContext, application, config and exception.

Simple Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>

<body>

<% out.print("welcome to jsp"); %>

</body>

</html>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

index.html

```
<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname">

<input type="submit" value="go"><br/>

</form>

</body>

</html>
```

welcome.jsp

```
<html>

<body>

<%

String name=request.getParameter("uname");

out.print("welcome "+name);

%>

</form>

</body>

</html>
```

9.7.1.2 JSP expression tag

The code placed within expression tag is written to the output stream of the response. So you need not write `out.print()` to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

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

Example of JSP expression tag

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

```
<html>

<body>

<%= "welcome to jsp" %>

</body>

</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the `getTime()` method of Calendar class. The `getTime()` is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the `getInstance()` method.

index.jsp

```
<html>

<body>

Current Time: <%= java.util.Calendar.getInstance().getTime()
%>

</body>

</html>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The `index.html` file gets the username and sends the request to the `welcome.jsp` file, which displays the username.

index.html

```
<html>

<body>

<form action="welcome.jsp">

<input type="text" name="uname"><br/>

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

</form>

</body>
```

```
</html>
```

welcome.jsp

```
<html>

<body>

<%= "Welcome "+request.getParameter("uname") %>

</form>

</body>

</html>
```

9.7.1.3 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:

```
<%!  field or method declaration %>
```

Difference between the jsp scriptlet tag and jsp 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

```
<html>

<body>

<%! int data=50; %>

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

</body>

</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

```
<html>

<body>

<%!

int cube(int n){

return n*n*n*;

}

%>

<%= "Cube of 3 is:"+cube(3) %>

</body>

</html>
```

9.7.1.4 JSP Comment

There is only one type of JSP comment available by JSP specification.

JSP Comment Syntax:

```
<%-- comment --%>
```

This JSP comment tag tells the JSP container to ignore the comment part from compilation. That is, the commented part of source code is not considered for the content parsed for 'response'.

Example:

```
<%-- This JSP comment part will not be included in the response object --%>
```

9.7.2 JSP directive elements

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

```
<%@ directive attribute="value" %>
```

9.7.2.1 JSP page directive

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

Syntax of JSP page directive

```
<%@ 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

```
<html>

<body>

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

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

</body>

</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

```
<html>

<body>

<%@ page contentType=application/msword %>

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

</body>
```

```
</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

```
<html>

<body>

<%@ page info="composed by ABC" %>

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

</body>

</html>
```

The web container will create a method `getServletInfo()` in the resulting servlet.

For example:

```
public String getServletInfo() {

return "composed by ABC";

}
```

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

```
<html>

<body>

<%@ page buffer="16kb" %>

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

</body>

</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.

```
<%@ 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:

```
public class SimplePage_jsp extends HttpJspBase

implements SingleThreadModel{

.......
```

```
}
```

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

```
//index.jsp

<html>

<body>

<%@ page errorPage="myerrorpage.jsp" %>

<%= 100/0 %>

</body>

</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

```
//myerrorpage.jsp

<html> <body>

<%@ page isErrorPage="true" %>

 Sorry an exception occured!

<br/>

The exception is: <%= exception %>
```

```
</body> </html>
```

9.7.2.2 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

```
<%@ 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.

```
<html> <body>

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

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body> </html>
```

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

9.7.2.3 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

```
<%@ 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.

```
<html> <body>

<%@ taglib uri="http://www.javatpoint.com/tags"
prefix="mytag" %>

<mytag:currentDate/>

</body> </html>
```