Module 2 - JSP and Controlling the Structure of generated servlets

Invoking java code with JSP scripting elements, creating Template Text, Invoking java code from JSP, Limiting java code in JSP, using jsp expressions, comparing servlets and jsp, writing scriptlets. For example Using Scriptlets to make parts of jsp conditional, using declarations, declaration example.
Controlling the Structure of generated servlets: The JSP page directive, import attribute, session attribute,isElignore attribute, buffer and autoflush attributes, info attribute, errorPage and iserrorPage attributes, isThreadSafe Attribute, extends attribute, language attribute, Including files and applets in jsp Pages, using java beans components in JSPdocuments

2.1 Invoking java code with JSP scripting elements

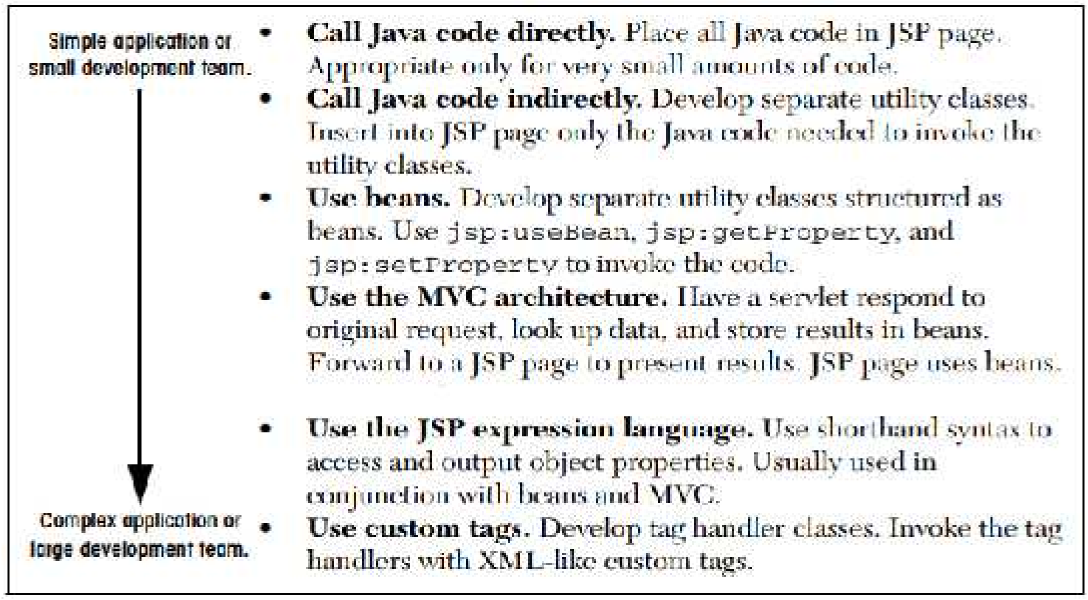
- JSP Scripting element are written inside <% %> tags. These code inside <% %> tags are processed by the JSP engine during translation of the JSP page. Any other text in the JSP page is considered as HTML code or plain text.

2.2 Creating Template Text

- JSP document consists of static text (usually HTML), known as template text.
- There are two minor exceptions to the "template text is passed straight through" rule. First, if you want to have in the output, you need to put in the template text. Second, if you want a comment to appear in the JSP page but not in the resultant document, use
  - **<%--** *JSP COMMENTS* **--%>**
- HTML comments of the form are passed through to the client normally
  - **<!--** *HTML COMMENTS* **-->**

2.3 Invoking java code from JSP

- There are a number of different ways to generate dynamic content from JSP, as illustrated in the below figure.


Simple application or small development team.
Complex application or large development team.

- **Call Java code directly.** Place all Java code in JSP page. Appropriate only for very small amounts of code.
- **Call Java code indirectly.** Develop separate utility classes. Insert into JSP page only the Java code needed to invoke the utility classes.
- **Use beans.** Develop separate utility classes structured as beans. Use `jsp:useBean`, `jsp:getProperty`, and `jsp:setProperty` to invoke the code.
- **Use the MVC architecture.** Have a servlet respond to original request, look up data, and store results in beans. Forward to a JSP page to present results. JSP page uses beans.
- **Use the JSP expression language.** Use shorthand syntax to access and output object properties. Usually used in conjunction with beans and MVC.
- **Use custom tags.** Develop tag handler classes. Invoke the tag handlers with XML-like custom tags.

2.3.1 Types of JSP Scripting Elements

- JSP scripting elements allows to insert Java code into the servlet that will be generated from the JSP page. There are three forms:

  1. ***Expressions*** of the form, which are evaluated and inserted into the servlet's output.

  2. ***Scriptlets*** of the form, which are inserted into the servlet's _jspService method (called by service).

  3. ***Declarations*** of the form, which are inserted into the body of the servlet class, outside any existing methods.

  4. ***Directive*** is used to provide instructions to the container.

2.4 Limiting java code in JSP

- There are two options
  - Put 25 lines of Java code directly in the JSP page
  - Put those 25 lines in a separate Java class and put 1 line in the JSP page that invokes it
- Why is the second option much better?
  - *Development*- Write the separate class in a Java environment (editor or IDE), not an HTML environment
  - *Debugging*- If there are syntax errors, it is immediately seen at compile time.
  - *Testing*. You can write a test routine with a loop that does 10,000 tests and reapply it after each change.
  - *Reuse*. You can use the same class from multiple pages.

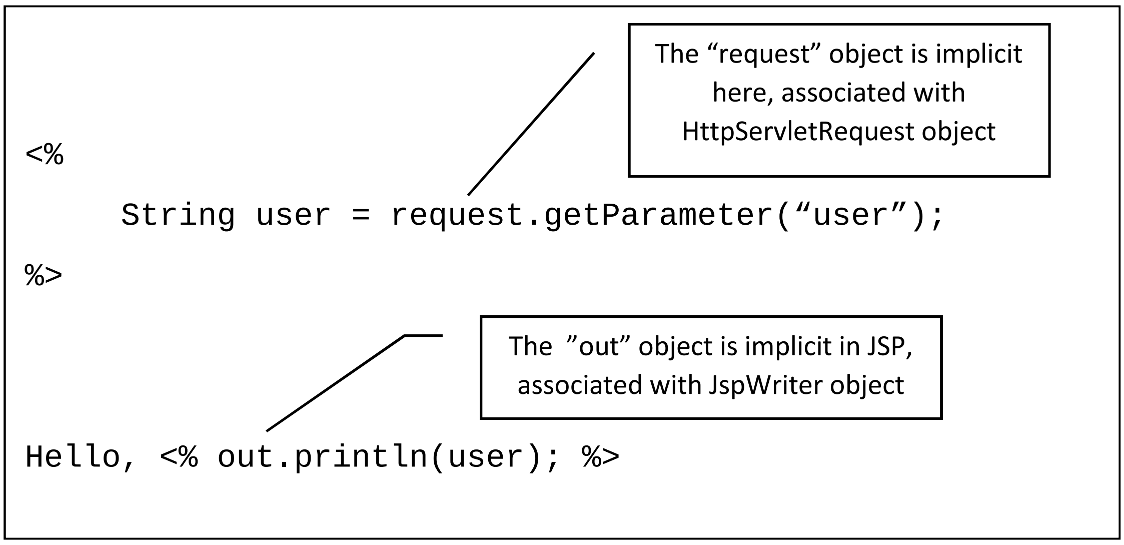
2.5 Using jsp expressions

- A JSP expression is used to insert values directly into the output. It has the following form.
  - **<%=** *Java Expression* **%>**
- The expressions are
  - evaluated,
  - converted to a string, and
  - inserted in the page.
- This evaluation is performed at runtime (when the page is requested) and thus has full access to information about the request.
- For example, the following shows the usage of expression.
  - Current time: **<%= *new java.util.Date( )* %>**
  - To display user name :<%= *"welcome"* + *request.getParameter("uname)"*"%>
  - To evaluate the expression <%= *2*10* %>

2.5.1 Predefined Variables (or *implicit objects*)

- To simplify these expressions, a number of predefined variables (or "implicit objects"). are used.

- These Objects are the Java objects that the JSP Container makes available to the developers in each page and the developer can call them directly without being explicitly declared.
- For example you can retrieve HTML form parameter data by using request variable, which represent the HttpServletRequest object.



```
<%
     String user = request.getParameter("user");
%>

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

The "request" object is implicit here, associated with HttpServletRequest object
The "out" object is implicit in JSP, associated with JspWriter 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
Session	The **HttpSession** object associated with the session for the given user of request.
Out	The **JspWriter** object associated with the output stream of the response.
Application	The **ServletContext** object for the web application.
Exception	The **exception** object represents the **Throwable** object that was thrown by some other JSP page.

2.5.1.1 Jsp/Servlet Correspondence

- It is actually quite simple: JSP expressions basically become print (or write) statements in the servlet that results from the JSP page.
- Whereas regular HTML becomes print statements with double quotes around the text, JSP expressions become print statements with no double quotes.
- Instead of being placed in the doGet method, these print statements are placed in a new method called _jspService that is called by service for both GET and POST requests.
- out in a JSP page is a JspWriter, so you have to modify the slightly simpler PrintWriter that directly results from a call to getWriter. So, don't expect the code your server generates to look *exactly* like this.

Example: Sample JSP Expression: Random Number

```
<H1>A Random Number</H1>
<%= Math.random() %>
```

Representative Resulting Servlet Code: Random Number

```
public void _jspService(HttpServletRequest request,
HttpServletResponse response)
}throws ServletException, IOException {

HttpSession session = request.getSession();
JspWriter out = response.getWriter();

out.println("<H1>A Random Number</H1>");
out.println(Math.random());
...
```

2.5.2 Example: JSP Expression

```
<!DOCTYPE html>
<html>
    <head><title>JSP Page</title></head>
    <body>
        <h1>JSP Expression</h1>
        <UL>
           <li>Current time: <%= new java.util.Date() %></li>
           <li>Server: <%= application.getServerInfo() %></li>
           <li>Session ID: <%= session.getId() %></li>
          <li> Expression Evaluation: <%= 2 * 6% ></li>
        </UL>
    </body>
</html>
```

2.6 Comparing servlets and jsp

JSP	Servlet
JSP is a web page scripting language that can generate dynamic content	Servlets are already compiled which also creates dynamic web content
JSP runs slower compared to servlet as it takes compilation time to convert into java Servlets	Servlets run faster compared to JSP.

It’s easier to code in JSP than in Java Servlets.	Its little much code to write here.
In MVC, jsp act as a view.	In MVC, servlet act as a controller.
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.
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.

Example in Servlet: ThreeParams.java

```
public class ThreeParams extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse
response)
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    String title = "Reading Three Request Parameters";
    out.println("<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>param1</B>: "
                + request.getParameter("param1") + "\n" +

                "  <LI><B>param2</B>: "

                + request.getParameter("param2") + "\n" +

                "  <LI><B>param3</B>: "
                + request.getParameter("param3") + "\n" +

                "</UL>\n" +
                "</BODY></HTML>"); } }
```

Example in JSP: ThreeParams.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Reading Three Request Parameters</TITLE> </HEAD>
<BODY>
      <H1>Reading Three Request Parameters</H1>
      <UL>
        <LI><B>param1</B>: <%= request.getParameter("param1") %>
        <LI><B>param2</B>: <%= request.getParameter("param2") %>
        <LI><B>param3</B>: <%= request.getParameter("param3") %>
      </UL>
</BODY></HTML>
```

2.7 Writing scriptlets.

- Scriptlet Tag allows to write java code inside JSP page
- Scriptlet tag implements the `_jspService` method functionality by writing script/java code.
- Scriptlets have the following form:

```
<% Java Code %>
```

- Scriptlets have access to the same automatically defined variables as do expressions (request response, session, out, etc.). So, for example, if you want to explicitly send output to the resultant page, you could use the out variable, as in the following example.

```
<%
String queryData = request.getQueryString();
out.println("Attached GET data: " + queryData);
%>

---------------------------------------------------------
```

Or combination of a scriptlet and a JSP expression, as below

```
<% String queryData = request.getQueryString(); %>
Attached GET data: <%= queryData %>

---------------------------------------------------------
```

Or single JSP expression, as here

```
Attached GET data: <%= request.getQueryString() %>
```

Scriptlets Example: Printing even number

```
<html>
<head><title>Even number program in JSP</title></head>
<body>
     <%
       for(int i=0;i<=10;i++)
       {
        if((i%2)==0)
        {
         out.print("Even number  :"+i);
         out.print("<br>");
        }
       }
      out.println(“<br><br><br>First 10 odd numbers are”);
      for(i=0;i<=10;i++)
      {
            if(i%2!=0)
            {
                  out.println(i);
            }
      }
     %>
</body>
</html>
```

2.8 For example, Using Scriptlets to make parts of jsp conditional

- Another use of scriptlets is to conditionally output HTML or other content that is *not* within any JSP tag.
- Key to this approach are the facts that
  (a) code inside a scriptlet gets inserted into the resultant servlet's _jspService method (called by service) *exactly* as written and
  (b) that any static HTML (template text) before or after a scriptlet gets converted to print statements.
- This behavior means that scriptlets need not contain complete Java statements and that code blocks left open can affect the static HTML or JSP outside the scriptlets. For example, consider the JSP fragment that contains mixed template text and scriptlets.

```
<body>
     <% if (Math.random() < 0.5) { %>
     <H1>Have a <I>nice</I> day!</H1>
     <% } else { %>
     <H1>Have a <I>lousy</I> day!</H1>
     <% } %>
</body>
```

- You probably find the bold part a bit confusing, Simply follow the rules for how JSP code gets converted to servlet code. Once you think about how this example will be converted to servlet code by the JSP engine, you get the following easily understandable result.

```
if (Math.random() < 0.5) {
  out.println("<H1>Have a <I>nice</I> day!</H1>");
} else {
  out.println("<H1>Have a <I>lousy</I> day!</H1>");
}
```

- Overuse of this approach can lead to JSP code that is hard to understand and maintain. Avoid using it to conditionalize large sections of HTML, and try to keep your JSP pages as focused on presentation (HTML output) tasks as possible.

2.9 Using declarations

- A JSP declaration lets you define methods or fields that get inserted into the main body of the servlet class (*outside* the _jspService method that is called by service to process the request).
- Declaration is made inside the servlet class but outside the service method.
- A declaration has the following form:

```
<%! Field or Method Definition %>
```

- Since declarations do not generate output, they are normally used in conjunction with JSP expressions or scriptlets.
- In principle, JSP declarations can contain field (instance variable) definitions, method definitions, inner class definitions, or even static initializer blocks: anything that is legal to put inside a class definition but outside any existing methods.
- In practice, however, declarations almost always contain field or method definitions.
- *Example:*

```
<H1>Some Heading</H1>
<%!
  private String randomHeading() {
    return("<H2>" + Math.random() + "</H2>");
  }
%>
<%= randomHeading() %>
```

2.10 Declaration Example:

- JSP snippet prints the number of times the current page has been requested.

```
<%! private int accessCount = 0; %>

Accesses to page since server reboot:
<%= ++accessCount %>
```

2.10 Controlling the Structure of generated servlets: The JSP page directive

- JSP *directive* affects the overall structure of the servlet that results from the JSP page.
- The **jsp directives** are messages that tells the web container how to translate a JSP page into the corresponding servlet.
- Following are the two forms to use directives in Jsp

i) **<%@ directive attribute="value" %>**
ii) **<%@ directive attribute1="value1"**
attribute2="value2"
....
attributeN="valueN" %>

- In JSP, there are three types of directives:
  - `page`,
  - `include`, and
  - `taglib`

- The **`page`** directive
  - controls the structure of the servlet by importing classes,
  - customizing the servlet superclass,
  - setting the content type
  - `page` directive can be placed anywhere within the document
  - The `page` directive defines following attributes:
    - `import`, `contentType`, `isThreadSafe`, `session`, `buffer`, `autoflush`, `extends`, `info`, `errorPage`, `isErrorPage`, and `language`.

```
Syntax:
<%@page [attribute_name] =[value]%>
```

- The **`include`** directive
  - Inserts a file into the servlet class at the time the JSP file is translated into a servlet.
  - An `include` directive should be placed in the document at the point at which you want the file to be inserted;
  - The *include directive tells the Web Container to copy everything in the included file and* paste it into current JSP file.

```
Syntax:
<%@ include file="filename.jsp" %>
```

Example:

```
<html>
     <body>
          <%@ include file= “header.jsp” %> <br>
               Contact us at: we@studytonight.com
          <br/>
          <%@ include file= “footer.jsp” %>
          <br/>
     </body>
</html>
```

- The **`taglib`** directive
    - which can be used to define custom markup tags;
    - 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 many commonly used core functionalities.
    - Syntax:

```
<%@taglib prefix=”mine” uri=”randomName” %>
```

-
    - **prefix** is prepended to the custom tag name. Each library used in a page needs its own taglib directive with unique prefix.
    - **uri** is a unique identifier in the Tag Library Descriptor (TLD). It’s a unique name for the tag library the TLD describe.

2.11 import attribute

- The `import` attribute of the `page` directive lets you specify the packages that should be imported by the servlet into which the JSP page gets translated.
- By default, a container automatically imports `java.lang.*`, `javax.servlet.*`, `javax.servlet.jsp.*`, `javax.servlet.http.*`, and possibly some number of server-specific entries.
- Use of the `import` attribute takes one of the following two forms:

```
<%@ page import="package.class" %>
<%@ page import="package.class1,...,package.classN" %>
```

- Example: the following directive signifies that all classes in the `java.util` package should be available to use without explicit package identifiers.

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

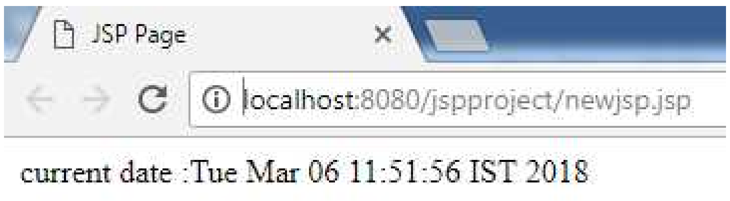
- The `import` attribute is the only `page` attribute that is allowed to appear multiple times within the same document.

Example : import attribute

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

<html>
      <head>
            <title>import page directive example</title>
      </head>
      <body>
            Current date: <%= new Date() %>
      </body>
</html>
```

Output



JSP Page

localhost:8080/jspproject/newjsp.jsp

```
current date :Tue Mar 06 11:51:56 IST 2018
```

2.12 contentType

- The contentType attribute sets the character encoding for the JSP page and for the generated response page. The default content type is **text/html**, which is the standard content type for HTML pages.

```
<%@ page contentType=text/HTML %>
```

2.13 session attribute

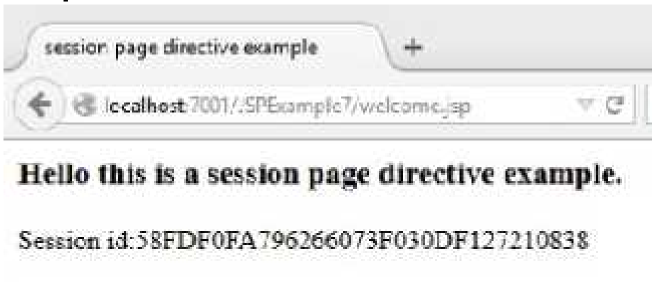
- The `session` attribute controls whether or not the page participates in HTTP sessions.
- Use of this attribute takes one of the following two forms:

```
<%@ page session="true" %> <%-- Default --%>
<%@ page session="false" %>
```

- A value of `true` (the default) indicates that the predefined variable `session` (of type `HttpSession`) should be bound to the existing session if one exists;
- a new session should be created and bound to `session`. A value of `false` means that no sessions will be used automatically and attempts to access the variable `session` will result in errors at the time the JSP page is translated into a servlet.

```
Example: session attribute
<%@ page session="true" %>
<html><head><title>session </title></head>
<body>
   <h3>Hello this is a session page directive example.</h3>
      <%
            out.print("Session id:" + session.getId());
      %> </body></html>
```

Output:


session page directive example
localhost:7001/JSPExample7/welcome.jsp

Hello this is a session page directive example.

Session id:58FDF0FA796266073F030DF127210838

2.14 isElignore attribute

- The isELIgnored attribute takes boolean value of true or false as input.
- If isELIgnored is true, then any Expression Language in JSP is ignored.
- The default value of isELIgnored is false.
- The example below illustrates the usage of isELIgnored attribute of page directive.

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

2.15 buffer attribute

- The `buffer` attribute specifies the size of the buffer used by the `out` variable, which is of type `JspWriter` (a subclass of `PrintWriter`). Use of this attribute takes one of two forms:

```
<%@ page buffer="sizekb" %>
<%@ page buffer="none" %>
```

- Servers can use a larger buffer than you specify, but not a smaller one.
- For example, `<%@ page buffer="32kb" %>` means the document content should be buffered and not sent to the client until at least 32 kilobytes have been accumulated or the page is completed.
- The default buffer size is server specific, but must be at least 8 kilobytes.
- Be cautious about turning off buffering; doing so requires JSP entries that set headers or status codes to appear at the top of the file, before any HTML content.

2.16 autoflush attribute

- The `autoflush` attribute controls whether the output buffer should be automatically flushed when it is full or whether an exception should be raised when the buffer overflows.
- Use of this attribute takes one of the following two forms:

```
<%@ page autoflush="true" %> <%-- Default --%>
<%@ page autoflush="false" %>
```

- A value of `false` is illegal when also using `buffer="none"`.

2.17 info attribute

- The `info` attribute defines a string that can be retrieved from the servlet by means of the `getServletInfo` method. Use of `info` takes the following form:

```
<%@ page info="Some Message" %>
```

2.18 errorPage attributes

- The `errorPage` attribute specifies a JSP page that should process any exceptions (i.e., something of type `Throwable`) thrown but not caught in the current page.
- It is used as follows:

```
<%@ page errorPage="Relative URL" %>
```

- The exception thrown will be automatically available to the designated error page by means of the `exception` variable.

2.19 iserrorPage attribute

- The `isErrorPage` attribute indicates whether or not the current page can act as the error page for another JSP page.
- Use of `isErrorPage` takes one of the following two forms:

```
<%@ page isErrorPage="true" %>
<%@ page isErrorPage="false" %> <%!-- Default --%>
```

- JSP page to compute speed based upon distance and time parameters.
- The page neglects to check if the input parameters are missing or malformed, so an error could easily occur at run time.

2.20 isThreadSafe Attribute

- The `isThreadSafe` attribute controls whether or not the servlet that results from the JSP page will implement the `SingleThreadModel` interface.
- Use of the `isThreadSafe` attribute takes one of the following two forms:

```
<%@ page isThreadSafe="true" %> <%!-- Default --%>
<%@ page isThreadSafe="false" %>
```

- **isThreadSafe="true"**, creates multiple objects for the same JSP file when requested by multiple clients. Each client is served with a separate **_jspService()** method (with only one JSP file loaded).
- **isThreadSafe="false"**, allows the container to create one Servlet object for each client requesting the same JSP.

2.21 extends attribute

- The **extends** attribute specifies a superclass that the generated servlet must extend.
- For example, the following directive directs the JSP translator to generate the servlet such that the servlet extends somePackage.SomeClass
- General form:

```
<%@ page extends="package.class" %>
```

2.22 language attribute

- The language attribute is intended to specify the scripting language being used, as below:

```
<%@ page language="java" %>
```

- since `java` is both the default and the only legal choice.

2.23 Including files in JSP Pages

- There are two ways to include a file
  1. Including files at Page Translation Time
  2. Including files at request Time

- **Including files at Page Translation Time**
  - **Include directive** is used for merging external files to the current JSP page during translation phase (The phase where JSP gets converted into the equivalent Servlet)
  - Syntax:

```
<%@ include file="Relative URL" %>
```

  -
- **Including files at request Time**
  - The **jsp:include action tag** is used to include the content of another resource it may be jsp, html or servlet.
  - The jsp include action tag includes the resource at request time so it is **better for dynamic pages**

- **Syntax**

```
<jsp:include page="header.jsp" flush="true" />
```

Action tags

- The action tags are used to control the flow between pages and to use Java Bean.
- These tags are used to remove or eliminate scriptlet code from JSP page because scriplet code are technically not recommended.
- Standard tags begin with the jsp: prefix
- Following are the JSP standard Action tag which are used to perform some specific task.

JSP Action Tags	Description	Syntax
jsp:forward	forwards the request and response to another resource.	`<jsp:forward page="relativeURL  />`
jsp:include	includes another resource.	`<jsp:include page ="{ page_to _include}" flush="true" />`
jsp:useBean	creates or locates bean object.	`<jsp:useBean id="bean name"  class="qualified path of bean" scope="page | request | application |session">`
jsp:setProperty	sets the value of property in bean object.	`<jsp:setProperty name="beanName" property="propertyName" value="propertyValue" />`
jsp:getProperty	prints the value of property of the bean.	`<jsp:getProperty name="beanName" property="propertyName" />`
jsp:plugin	embeds another component such as applet.	`<jsp:plugin type="applet" code="MyApplet.class" width="450" height="350">` `</jsp:plugin>`
jsp:param	sets the parameter value. It is used in forward and include mostly.	`<jsp:params>` `<jsp:param name="MESSAGE" value="Your Message Here" />` `</jsp:params>`
jsp:fallback	Can be used to print the message if plugin is working. It is used in jsp:plugin.	`<jsp:fallback>` `<B>Error: this example requires java</B>` `</jsp:fallback>`

Difference between include directive and include Action

Include Directive	Include Action
Resources included during jsp translation time	Resources included during request time
Attribute to specify resource is *file*	Attribute to specify resource is *page*
Copies the included file	References to the included file
For static content	For dynamic content
Cannot pass parameters	Can pass parameter through <jsp:param> tag

2.24 Including applets in JSP pages

- Java programs embeded in web pages and executed by web browsers.
- The jsp:plugin element is used to insert applets that use the Java Plug-in into JSP pages.
- Applets will be deployed to the general public, because that option does not require users to install any special software
- Its main advantage is that it saves you from writing long, tedious, and error-prone OBJECT and EMBED tags in your HTML.
- Its main disadvantage is that it applies to applets, and applets are relatively infrequently used.
- The jsp:plugin element instructs the server to build a tag appropriate for applets that use the plug-in.
- The jsp:plugin Element
    - The simplest way to use jsp:plugin is to supply four attributes
        - type
        - code
        - width
        - height
- The attribute names are case sensitive, and single or double quotes are always required around the attribute values.
- Example:

```
    <jsp:plugin type="applet"
         code="MyApplet.class"
         width="475" height="350">
    </jsp:plugin>
```

2.24.1 Attributes of jsp:plugin

- The attributes of the JSP **plugin** tag provide configuration data for the presentation of the tag

Attribute	Description
type	Identifies the type of the component. This attribute is required.
code	The full name of the class including package notation and the .class extention. This attribute is required.
codebase	The directory names where to find the class file.

[align]	Positioning of the object
[archive]	Specifies a space-separated list of URLs indicating resources needed by the object.
[name]	Name of the object when submitted as part of a html form.
[height]	Indicates the maximum height in CSS pixels.
[width]	Indicates the maximum width in CSS pixels.
[hspace]	Specifies the whitespace on left and right side of an object in CSS pixels.
[vspace]	Specifies the whitespace on top and bottom of an object in CSS pixels.
[jreversion]	Identifies the spec version number of the JRE the component requires in order to operate.
[nspluginurl]	URLs where the Java plug-in can be downloaded for Netscape Navigator.
[iepluginurl]	URLs where the Java plug-in can be downloaded for Internet Explorer.

2.25 Using java beans components in JSP documents

2.25.1 Why use Beans?

- Using separate Java classes instead of embedding large amounts of code directly in JSP pages.
- Beans are regular Java classes that follow some simple conventions defined by the JavaBeans specification; beans extend no particular class, are in no particular package, and use no particular interface.
- Use of JavaBeans components provides three advantages over scriptlets and JSP expressions that refer to normal Java classes.
    - No Java syntax
    - Simpler object sharing
    - Convenient correspondence between request parameters and object properties.

2.25.2 What are Beans?

- 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.

2.25.3 Using Beans: Basic Task

- There are three main constructs to build and manipulate javabeans component in jsp.
  - **jsp:useBean**
      - The jsp:usebean action loads a bean to be used in the jsp page.

- This element builds a new bean. It is normally used as follows:

```
<jsp:useBean id= "instanceName"
  scope= "page | request | session | application"
  class= "packageName.className"
  type= "packageName.className"
  beanName="packageName.className >
</jsp:useBean>
```

- This statement usually means "instantiate an object of the class specified by class, and bind it to a variable in _jspService with the name specified by id.
- The `class` attribute, you are permitted to use `beanName` instead.
- If you supply a `scope` attribute the `jsp:useBean` element can either build a new bean or access a preexisting one.
- **Attributes and usage of jsp:useBean action tag**
  - **id:** is used to identify the bean in the specified scope.
  - **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.
    - **page:** specifies that you can use this bean within the JSP page. The default scope is page.
    - **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
    - **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
    - **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
  - **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-argument or no constructor and must not be abstract.
  - **type:** provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
  - **beanName:** instantiates the bean using the java.beans.Beans.instantiate() method.

- **jsp:getProperty(Accessing Bean Properties)**
  - This element reads and outputs the value of a bean property. Reading a property is a shorthand notation for calling a method of the form get***Xxx***.
  - This element is used as follows:

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

  - `jsp:getProperty`, which takes a `name` attribute that should match the `id` given in `jsp:useBean`

- **jsp:setProperty(Setting Bean Property)**
  - This element modifies a bean property (i.e., calls a method of the form `set`***`Xxx`***). It is normally used as follows:

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

  - It has three attributes:
    - `name, (`which should match the `id` given by `jsp:useBean)`
    - `value, (`the new value`)`
    - `property (`the name of the property to change`)`
  - Associate all properties with identically named input parameters.
  - Supply `"*"` for the `property` parameter.

```
<jsp:setProperty name="entry" property="*" />
```