Java Server Pages:

- JSP is a server side technology that enables web programmers to generate web pages dynamically in response to client requests.
- JSP is an as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

Shortcomings of Servlets:

- **Inability to use IDEs:** you cannot use IDEs to design HTML views. This increases the development time of web applications.
- **Inability to use HTML designer:** WE cannot use HTML designer services to prepare the presentation view. Instead we need to depend on the Java Programmer.
- **Depends on web configuration:** Servlets programs depend on the web configuration such as web.xml.
- **Recurring recompilation**: When you make changes in the code of a Servlet, we have to stop the server, recompile the source file, and restart the server.

Understanding JSP:

JSP is an extension to Servlets and is used to create dynamic web pages. JSP document consists of HTML and JSP tags. HTML is used to create static web pages and JSP tags are used to create dynamic web pages. JSP pages are translated into Java Servlets by translator. The Java servlet is the compiled and executed to generate an output for the client(browser).

Advantages of JSP:

- Allows tag based programming. So extensive java knowledge is not required.
- Suitable for both java and non-java programmer.
- Modification done in JSP program will be recognized by underlying server automatically without reloading of web application.
- Takes care of exception handling.
- Allows us to use separate presentation logic (html code) from Java code(business logic).
- Increases readability of code because of tags.
- Gives built-in JSP tags and also allows to develop custom JSP tags and to use third party supplied tags.
- JSPs provide session management by default.
- In JSPs, we have support for implicit objects.

Example: To print current date in Servlets and in JSP.

Servlet File

```
import java.io.*;
import javax.servlet.*;

public class DateSrv extends GenericServlet
{
    //implement service()
    public void service(ServletRequest req, ServletResponse res) throws IOException,
ServletException
    {
        //set response content type
        res.setContentType("text/html");
        //get stream obj
        PrintWriter pw = res.getWriter();
        //write req processing logic
        java.util.Date date = new java.util.Date();
        pw.println("<h2>"+"Current Date & Time: " +date.toString()+"</h2>");
        //close stream object
        pw.close();
    }
}
```

JSP File:

```
<%@ page import = java.util.*" %>
<html>
  <head>
    <title>Display Current Date & Time</title>
  </head>
  <body>

    <%
      Date date = new Date();
      out.print( date.toString());
    %>
  </body>
</html>
```

Advantages of tag based approach:

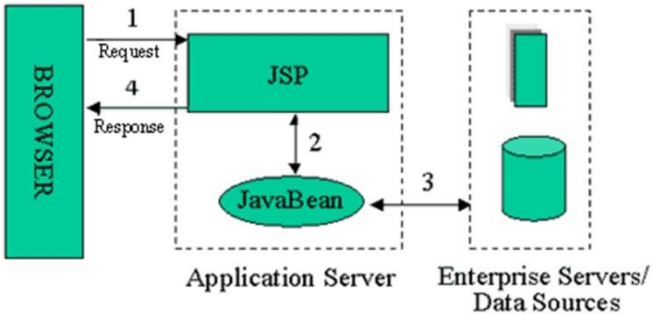
- JSP tag based approach helps reduces the Java code.
- Make the tasks of developing applications in easier way.
- Easy of working with beans in JSP.

Describing the JSP Architecture:

We can develop applications with JSP by using two approaches called JSP Model 1 and JSP Model 2 architectures.

Model 1 Architecture:

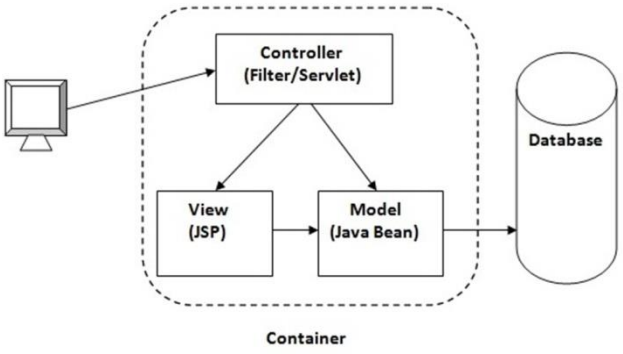
- In JSP Model 1 architecture, the web browser directly acces JSP page. The JSP page interacts with web containers JavaBean , which represents the application model then response is sent back to the browser.
- If the generated response requires database access, JSP uses JavaBean, which gets required data from the database.



Model II Architecture:

Model 2 is based on the MVC (Model View Controller) design pattern. The MVC design pattern consists of three modules model, view and controller.

- **Model** The model represents the state (data) and business logic of the application.
- **View** The view module is responsible to display data i.e. it represents the presentation.
- **Controller** The controller module acts as an interface between view and model. It intercepts all the requests i.e. receives input and commands to Model / View to change accordingly.

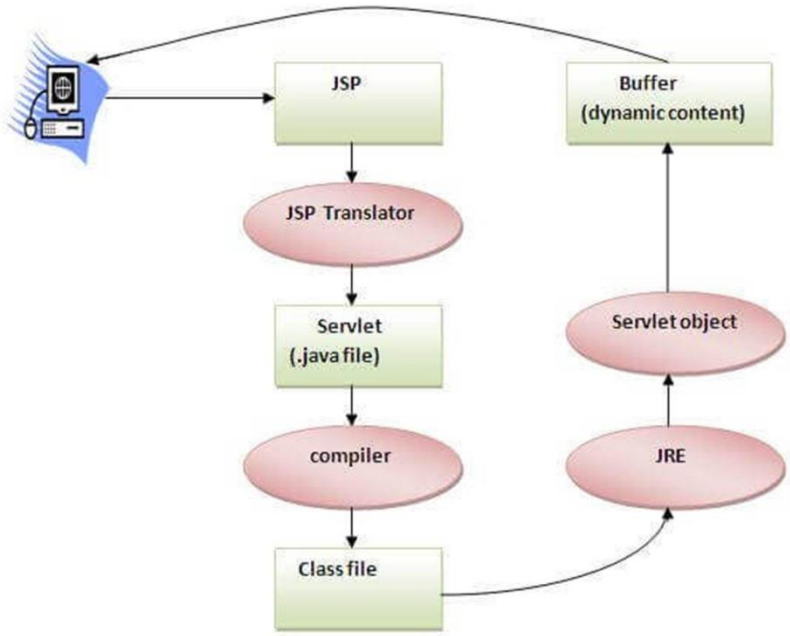


JSP Life Cycle:

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).

Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.



Translation of JSP Page:

- In this, the Web container translates JSP document into an equivalent Java code, i.e., a Servlet. Servlet contains Java code with some mark-up language tags such as HTML/XML.
- This is the first step of the JSP life cycle. This translation phase deals with the Syntactic correctness of JSP.
- Ex: Test.jsp file is translated into Tesp.java (servlet file)

Compilation of JSP Page:

- In this, the JSP container compiles the Java code for the corresponding servlet and converts into Java byte (class) code. After the compilation stage, the servlet is ready to be loaded and initialized.

Class loading :

- Servlet class which has been loaded (created) from the JSP source is now loaded into the container.

Instantiation:

- Here an instance of the class is generated. The container manages one or more instances by providing responses to requests.

Initialization:

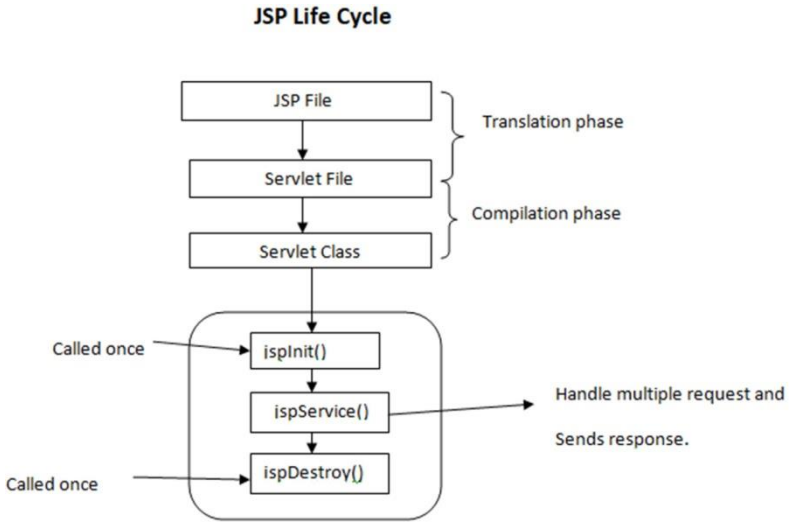
- jspInit() method is called only once during the life cycle immediately after the generation of Servlet instance from JSP.

Request processing:

- jspService() method is used to serve the raised requests by JSP. It takes request and response objects as parameters. This method cannot be overridden.

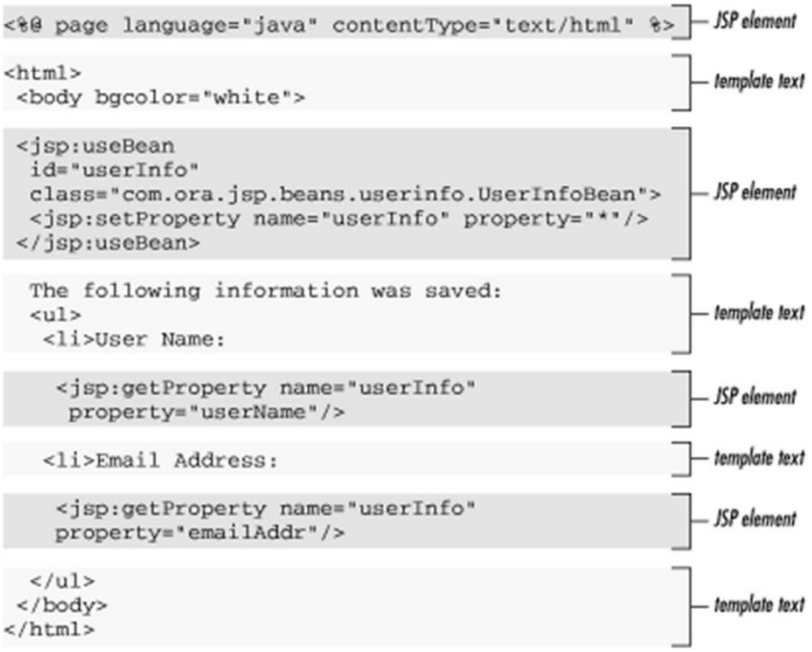
JSP Clean-up:

- In order to remove the JSP from the use by the container or to destroy the method for servlets jspDestroy()method is used. This method is called once, if you need to perform any cleanup task like closing open files, releasing database connections jspDestroy() can be overridden.



Anatomy of JSP Page:

A JSP page basically consists of two parts: HTML/XML markups and JSP elements. In many cases, large portion of JSP page mainly consists of HTML components, known as **Template Text.**



Basically three types of JSP elements are in JSP page:

a. JSP Scripting elements(tags)
   - JSP Scriptlet tag
   - JSP expression tag
   - JSP declaration tag

b. JSP Directives
   - JSP page directive
   - JSP include directive
   - JSP taglib directive

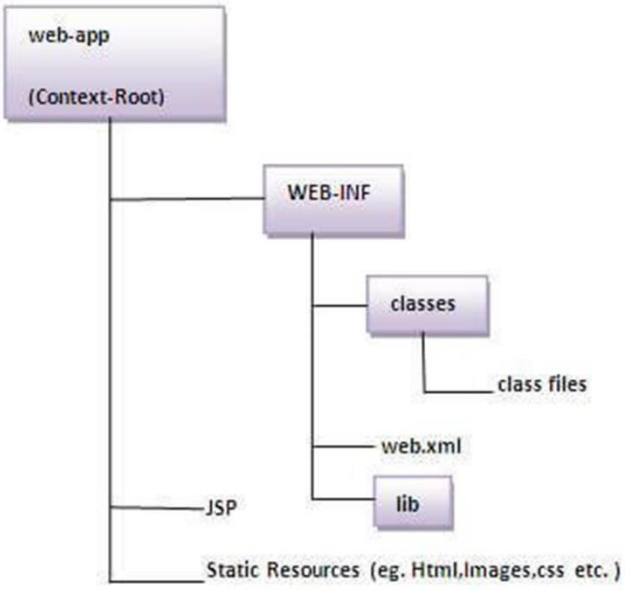
c. Implicit objects
   - out
   - request
   - response
   - config
   - application
   - session
   - pageContext
   - page
   - exception

d. JSP Action elements
   - jsp:forward
   - jsp:include
   - java bean class
   - jsp:useBean
   - jsp:setProperty and jsp:getProperty

e. JSP Comments
   - **Syntax:-**
     <% -- JSP Comments %>

The Directory structure of JSP:

The directory structure of JSP page is same as Servlet.



JSP Scripting Elements:

In JSP, java code can be written inside the jsp page using the scriptlet tag.

JSP Scripting elements

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

Scriptlets

- A scriptlet is a block of java code that is executed during the request processing time, and is enclosed between <% and %> tags.
- We can embed any amount of java code in the JSP Scriptlets. JSP Engine places these code in the *_jspService()* method.

Example:

```
<html>
    <body>
        <% out.print("Welcome to JSP"); %>
    </body>
</html>
```

Example of JSP scriptlet tag that prints the user name:

HTML File:

```
<html>
    <body>
        <form action="First.jsp">
            <input type="text" name="uname">
            <input type="submit" value="go"><br/>
        </form>
    </body>
</html>
```

First.jsp File:

```
<html>
    <body>
        <%
            String name=request.getParameter("uname");
            out.print("welcome "+name);
        %>
    </body>
</html>
```

JSP expression tag:

The most simple and basic of JSP scripting is expression. Expression is used to insert values directly to the output. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

- The syntax of the expression is as follows. (be noted that there is no space between <% and =)

  <%= expression %>

- For example, if you want to print out the current date and time you can use the expression as follows:

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

Example:

```
<html>
    <head>
        <title>JSP Expression Page</title>
    </head>
  <body>
        <h2>JSP Expression Page</h2>
        <p> Ex1: What is 3+3? <%= 3+3 %><br>
            Ex2: What is 2*3? <%= 2*3 %><br></p>
  </body>
</html>
```

Example: Prints the username with expression tag.

```
<html>
    <body>
        <%= "Welcome "+request.getParameter("uname") %>
    </body>
</html>
```

JSP Declaration Tag:

- The **JSP declaration tag** is used *to declare fields and methods.*
- The JSP declaration is surrounded by the sign **<%!** and **%>**. For example, if you want to declare a variable *x*, you can use JSP declaration as follows:

```
<%! int x = 10; %>
```

- The final semicolon is very important.
- The difference between a variable using declaration and a variable is declared using Scriptlet is that a variable declare using declaration tag is accessible by all the methods while a variable declared using Scriptlet is only accessible to the method *_jspservice* of the generated servlet from JSP page.
- The syntax of the declaration tag is as follows:

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

Example of JSP declaration tag that declares field

```
<html>
    <body>
        <%! int data=50; %>
        <%= "Value of the variable is:"+data %>
    </body>
</html>
```

Example of JSP declaration tag that declares method:

Second.jsp

```
<html>
<body>
    <%!
        int cube(int n){
            return n*n*n*;
        }
    %>
    <%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JSP Implicit Objects:

JSP implicit objects are predefined objects that are accessible to all JSP pages. 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. JSP Implicit Objects are also called pre-defined variables.

There are 9 implicit objects available in JSP:

Object	Type
request	HttpServletRequest
response	HttpServletResponse
out	JspWriter(similar to PrintWriter)
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	this (Object)
exception	Throwable

Advantages of implicit objects:

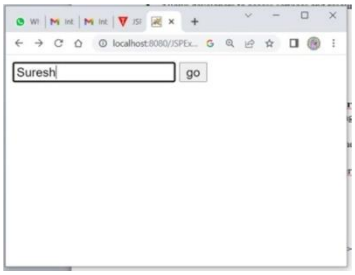
- Allows developers to access services and resources provided by the Web container.
- Are used to generate dynamic web content.
- No need to create instance explicitly.
- Implicitly available to all JSP pages.
- Helps in handling HTML parameters.

The request Object:

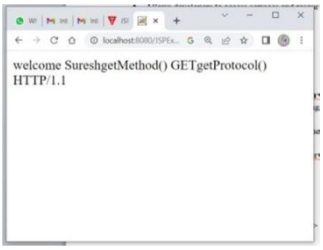
- The **request** object is an instance of a **javax.servlet.http.HttpServletRequest** object. Each time a client requests a page the JSP engine creates a new object to represent that request.
- The **request** object provides methods to get the HTTP header information including form data, cookies, HTTP methods etc.
- **request** object is passed as parameter to _jspService() method when a client request is made.

Ex:

```
<form action="RequestDemo.jsp">
        <input type="text" name="uname">
        <input type="submit" value="go"><br/>
</form>
```



RequestDemo.jsp




```
<%
        String name=request.getParameter("uname");
        out.print("welcome "+name);
        out.print("getMethod() "+request.getMethod());
        out.print("getProtocol() "+request.getProtocol());

%>
```

The response Object:

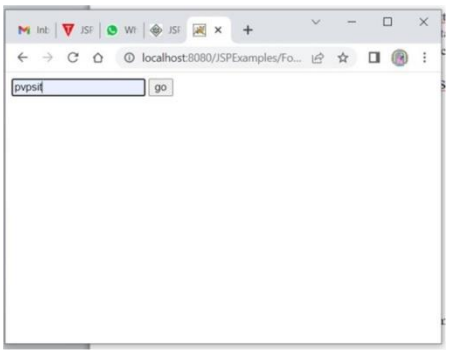
- In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each JSP request.
- It can be used to add or manipulate response such as redirect response to another resource, send error etc.
- **response** object is passed as parameter to _jspService() method when a client request is made.

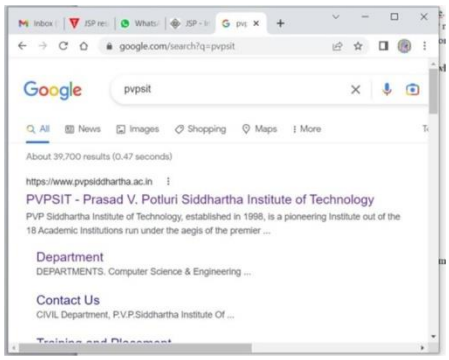
Form3.html

```
<form action="ResponseDemo.jsp">
    <input type="text" name="uname">
    <input type="submit" value="go"><br/>
</form>
```

ResponseDemo.jsp

```
<%
String pname=request.getParameter("uname");
response.sendRedirect("http://www.google.com/search?q="+pname);
%>
```





The out object:

- The out implicit object is an instance of a **javax.servlet.jsp.JspWriter** object and is used to send content in a response.
- The JspWriter object contains most of the same methods as the **java.io.PrintWriter** class.

Methods:

out.print(dataType dt) - Print a data type value

out.println(dataType dt) - Print a data type value then terminate the line with new line character.

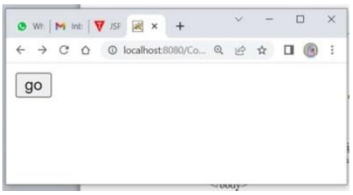
The config object:

- In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.
- Generally, it is used to get initialization parameter from the web.xml file.

Example:

index.html

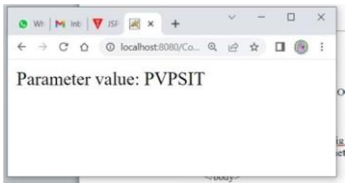
```
<form action="welcome">
        <input type="submit" value="go"><br/>
</form>
```



web.xml

```
<web-app>
        <servlet>
                <servlet-name>BeginnersBookServlet</servlet-name>
                <jsp-file>/Welcome.jsp</jsp-file>

                <init-param>
                        <param-name>parameter</param-name>
                        <param-value>PVPSIT</param-value>
                </init-param>
        </servlet>
        <servlet-mapping>
                <servlet-name>BeginnersBookServlet</servlet-name>
                <url-pattern>/welcome</url-pattern>
        </servlet-mapping>
</web-app>
```

welcome.jsp file:




```
<html>
        <head> <title> Config Implicit Object</title>
        </head>
        <body>
                <%
                        String p_v=config.getInitParameter("parameter");
                        out.print("Parameter value: "+p_v);
                %>
        </body>
</html>
```

The application object:

- In JSP, application is an implicit object of type *ServletContext*.
- The instance of ServletContext is created only once by the web container when application or project is deployed on the server.
- This object can be used to get initialization parameter from configuaration file (web.xml).

index.html

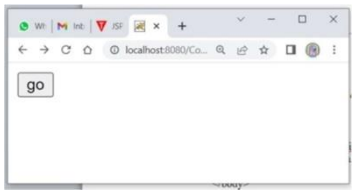
```
<form action="welcome">
        <input type="submit" value="go"><br/>
</form>
```

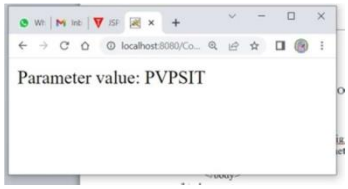
web.xml

```
<web-app>
<servlet>
 <servlet-name>BeginnersBookServlet</servlet-name>
 <jsp-file>/Welcome.jsp</jsp-file>
</servlet>
<servlet-mapping>
 <servlet-name>BeginnersBookServlet</servlet-name>
 <url-pattern>/welcome</url-pattern>
</servlet-mapping>

<context-param>
 <param-name>parameter</param-name>
 <param-value>PVPSIT</param-value>
</context-param>
</web-app>
```

Welcome.jsp

```
<html>
        <head> <title> Context Implicit Object</title>
        </head>
        <body>
         <%
                String p_v=application.getInitParameter("parameter");
                out.print("Parameter value: "+p_v);
         %>
        </body>
</html>
```





The session object:

- In JSP, session is an implicit object of type HttpSession.
- The Java developer can use this object to set,get or remove attribute or to get session information.

Example:

Index.html

```
<html>
        <body>
         <form action="welcome.jsp">
                <input type="text" name="uname">
                <input type="submit" value="go"><br/>
         </form>
        </body>
</html>
```

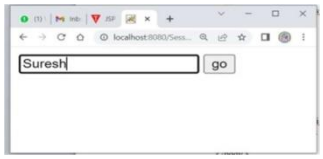
welcome.jsp

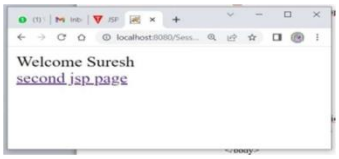
```
<html>
        <body>
         <%
                String name=request.getParameter("uname");
                out.print("Welcome "+name);

                session.setAttribute("user",name);
```

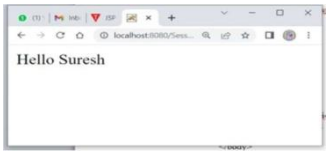
```
                %>
                <br>
                        <a href="second.jsp">second jsp page</a>

                </body>
</html>
```

second.jsp

```
<html>
                <body>
                <%
                 String name=(String)session.getAttribute("user");
                 out.print("Hello "+name);
                %>
                </body>
</html>
```







The exception object:

- In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception.

index.html

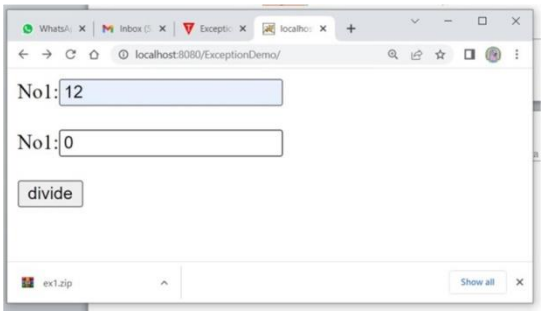
```
<form action="process.jsp">
                No1:<input type="text" name="n1" /><br/><br/>
                No1:<input type="text" name="n2" /><br/><br/>
<input type="submit" value="Divide"/></form>
```

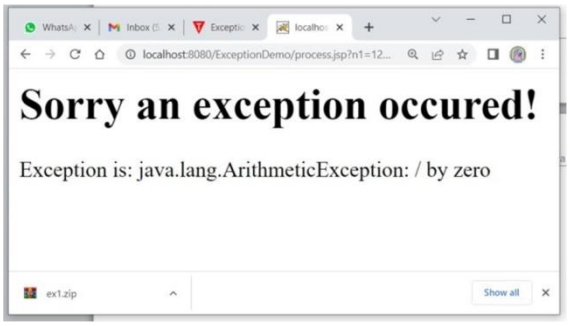
process.html

```
<%@ page errorPage="error.jsp" %>
<%
                String num1=request.getParameter("n1");
                String num2=request.getParameter("n2");

                int  a=Integer.parseInt(num1);
                int  b=Integer.parseInt(num2);
                int c=a/b;
                out.print("division of numbers is: "+c);
%>
```

error.jsp

```
<%@ page isErrorPage="true" %>
<h1>Sorry an exception occured!</h1>
Exception is: <%= exception %>
```





The pagecontext object:

- In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:
  - page
  - request
  - session
  - application

The page object:

- The *page* object is an instance of a JSP page. By using the *page* object, you can call any method of the page's servlet. The page object represents the generated servlet instance itself, i.e., it is same as the "this" keyword used in a Java file.

JSP Directives:

JSP directives are the elements of a JSP source code that guide the web container on how to translate the JSP page into its respective servlet.

Syntax:

<%@ directive attribute = "value"%>

Different types of JSP directives :

- page directive
- include directive
- taglib directive

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

a) import
b) contentType
c) extends
d) info
e) buffer
f) language
g) isELIgnored
h) isThreadSafe
i) autoFlush
j) session
k) pageEncoding
l) errorPage
m) isErrorPage

a. 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>
```

b. 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".

c. extends: The extends attribute defines the parent class that will be inherited by the generated servlet. It is rarely used.

d. 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="date printing" %>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 " date printing ";
}
```

e. 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>
```

f. language: The language attribute specifies the scripting language used in the JSP page. The default value is "java".

g. 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

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

h. 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{
.......
}
```

i. 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:

```
<html>
<body>

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

 <%= 100/0 %>

</body>
</html>
```

m. 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

```
<html>
<body>

<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>
The exception is: <%= exception %>

</body>
</html>
```

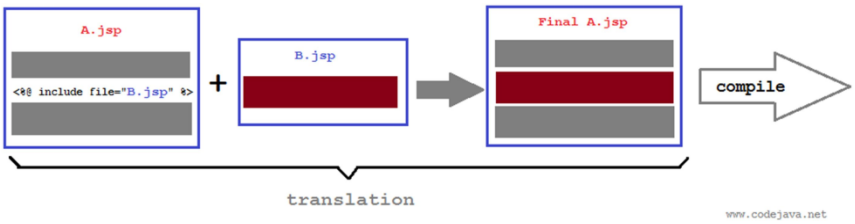
Include directive:

- The include directive tag is used to merge the contents of one or more files during the translation stage of the JSP lifecycle.
- The include directive adds the text of the included file to the JSP page, without processing or modification.
- The included file may be static or dynamic. If the included file is dynamic its JSP elements are translated and included. ‘

The syntax is <%@include file=”filepath” %>

XML based syntax:

<jsp:directive.include file=”file path” %>

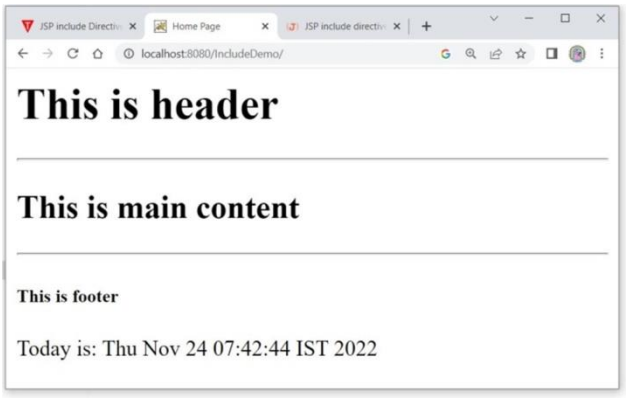


Example: index.jsp

```
<html>
<head>

<title>Home Page</title>
</head>
<body>

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

```
<hr/>
<h2>This is main content</h2>
<hr/>

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

        Today is: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

header. html -> <h1>This is header</h1>
footer.html -> <h5>This is footer</h5>



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.

Syntax JSP Taglib directive

```
<%@ taglib uri="URI" prefix="mytag" %>
```

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="URL Path" prefix="mytag" %>
        <mytag:currentDate/>
</body>
</html>
```

Using the JSP Standard Tag Library (JSTL):

- **The JSTL** is a collection of custom tag libraries, which provide core functionality used for JSP documents.
- JSP reduces the use of scriptlets in a JSP page.
- JSTL tags allow developers to use predefined tags instead of writing Java code.
- JSTL tags provide four types of libraries:
  - The Core JSTL
  - The XML Tag Library
  - The Format tag library
  - The SQL tag Library

The Core JSTL Tags:

- The core tags are used to perform iteration, conditional processing, and also provide support of expression language for the tags in JSP pages.
- The URL for the core tag is **http://java.sun.com/jsp/jstl/core**. The prefix of core tag is **c**.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

The core tags are divided into three categories:

- General-purpose tags
- Conditional and looping tags
- Network tags

General-Purpose Tags:

The general purpose tags contain four tags:

- <c:out> - for writing the value
- <c:set> - to set the value
- <c:remove> - to remove attribute
- <c:catch> - to handle exceptions

<c:out> tag: - similar to expression tag in JSP. It will display the result of an expression, similar to the way < %=...% > work.

Syntax:

<c:out attribute> [data content]</c:out>

Possible values of attribute are:

a. value – (Required)
b. default
c. escapeXml

<c:set> tag: allows us to set the value of a variable or property into the given scope.

Syntax:

<c:set attribute>[body content]</c:set>

The attributes are:

value – specifies the value being set

var – specifies the name of the variable

scope – scope of the variable such as{page, session, request, application}

<c:remove> tag: allows us to remove a variable from the specified scope.

Syntax:

<c:remove attribute/>

The attributes are:

var – name of the variable to be removed

scope – scope of the variable

<c:catch> tag: is used to handle the exceptions

Syntax:

<c:catch attributes>[bodycontent]</c:catch>

Attributes are:

var – name of the variable to store the exception thrown

Ex:

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title> JSTL Core Tags Example</title>
  </head>

  <body>

        <p><h1><c:out  value="Core Tags Examle"/></h1></p>

        <c:set var="Income" scope="session" value="${4000*4}" />

         <p>Before Remove Value is: <c:out value="${Income}"/></p>

        <c:remove var="Income"/>

        <p>After Remove Value is: <c:out value="${Income}"/></p>
```

```
        <c:catch var ="catchtheException">
                <% int x = 2/0;%>
        </c:catch>
        <p>Exception :<c:out value="${catchtheException}"/></p>

  </body>
</html>
```

Conditional and Looping Tags:

These are six types:

<c:if>
<c:choose>
<c:when>
<c:otherwise>
<c:forEach>
<c:forToken>

<c:if> tag: The < c:if > tag is used for testing the condition and it display the body content, if the expression evaluated is true.

- Attribute – test – Boolean variable

Ex:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
        <head>
                <title>Core Tag Example</title>
        </head>
        <body>
                <c:set var="income" scope="session" value="${4000*4}"/>
                <c:if test="${income > 8000}">
                        <p>My income is: <c:out value="${income}"/><p>
                </c:if>
        </body>
</html>
```

<c:choose> tag: This tag acts like switch statement. The <c:choose> tag encloses one or more <c:when> tags and a <c:otherwise> tag.

<c:choose> body </c:choose>

- The <c:when > is subtag of <choose > that will include its body if the condition evaluated be 'true'.
- The <c:when> tag accepts only one attribute named test(boolean value)
- The < c:otherwise > is also subtag of < choose > it follows <c:when > tags and runs only if all the prior condition evaluated is 'false'.

Ex:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>

<c:set var="income" scope="session" value="${4000*4}"/>

<p>Your income is : <c:out value="${income}"/></p>

<c:choose>

   <c:when test="${income <= 1000}">
     Income is not good.
   </c:when>

   <c:when test="${income > 10000}">
      Income is very good.
   </c:when>

   <c:otherwise>
     Income is undetermined...
   </c:otherwise>

</c:choose>

</body>
</html>
```

The <c:forEach> tag: The <c:forEach > is an iteration tag used for repeating the nested body content for fixed number of times or over the collection.

Syntax:

```
<c:forEach attributes> body</c:forEach>
```

Attributes are:

items – specifies the collection to iterate over
var – specifies the name of the variable (for(int i=0;i<10;i++)
begin – Starting index of the loop (0)
end – takes the ending index for the loop (10-1=9)
step – An optional increment for the loop. Default is 1. (++)

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
        <head>
                <title>Core Tag Example</title>
        </head>
        <body>
                <c:forEach var="j" begin="1" end="100" step="2">
                        Item <c:out value="${j}"/><p>
                </c:forEach>
        </body>
</html>
```

The <c:forTokens> tag: The <c:forTokens > is used for looping over tokenized elements of a string.

Syntax:

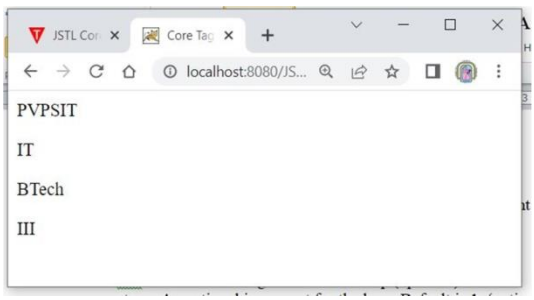
```
<c:forTokens attributes> body</c:forTokens>
```

Attributes are:

- items – specifies the string to be tokenized
- var – specifies the name of the variable into which the current iteration has to be set. (for(int i=0;i<10;i++)
- delims – specifies the delimiter
- begin – Starting index of the loop (optional)
- end – takes the ending index for the loop (optional)
- step – An optional increment for the loop. Default is 1. (optional)

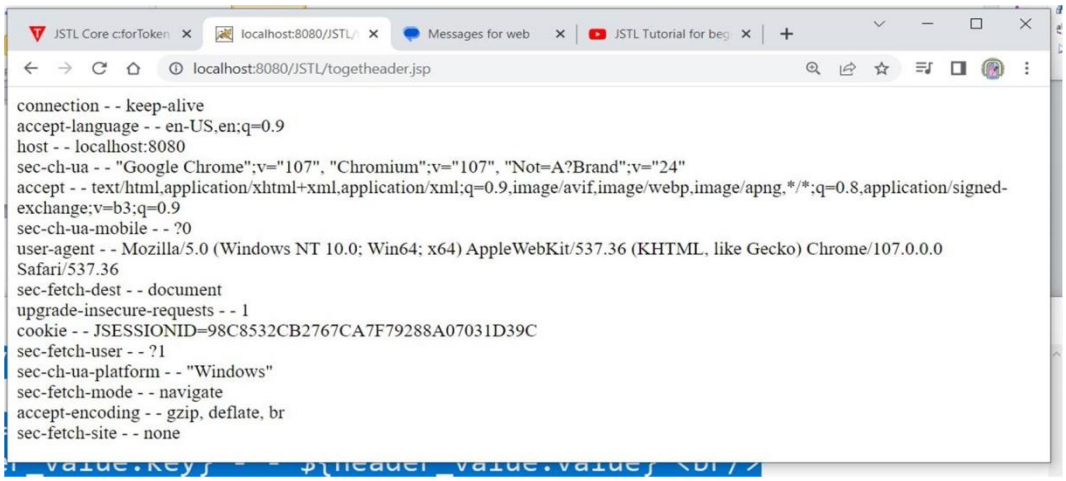
Ex:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
        <head>
                <title>Core Tag Example</title>
        </head>
        <body>
<c:forTokens items="VIJ;SAGTE;PVPSIT;IT;BTech;III;WT" delims=";"
        var="name" begin="2" end="5">
                        <c:out value="${name}"/><p>
                </c:forTokens>
        </body>
</html>
```



Ex: JSTL program to print header information.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<body>
        <c:forEach var="header_value" items="${header}">
                ${header_value.key} - - ${header_value.value} <br/>

        </c:forEach>
</body>
```



Networking Tags:

<c:import> tag:

- The <c:import> is similar to jsp 'include', with an additional feature of including the content of any resource either within server or outside the server.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
        <head>
                <title>Tag Example</title>
        </head>
        <body>
                <c:import var="data" url="togetheader.jsp"/>
                <c:out value="${data}"/>
        </body>
</html>
```

<c:redirect> tag: The < c:redirect > tag redirects the browser to a new URL.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
 <body>
    <c:redirect url="http://www.eenadu.net"/>
</body>
```

<c:url> tag: The < c:url > tag creates a URL with optional query parameter.

<c:param> tag: is used to add a request parameter to the URL. The tag can be used in <c:url> and <c:redirect> tags.

Syntax:

<c:param attributes/>

Attributes:

name - name of the parameter
value – value of the parameter

Ex:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<body>
  <c:url value="/index1.jsp" var="completeURL" >
        <c:param name="college" value="pvpsit"/>
        <c:param name="dept" value="it"/>
  </c:url>
  ${completeURL}
</body>
```