JSP

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to the servlet because it provides more functionality than servlet.

A JSP page contains HTML code and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

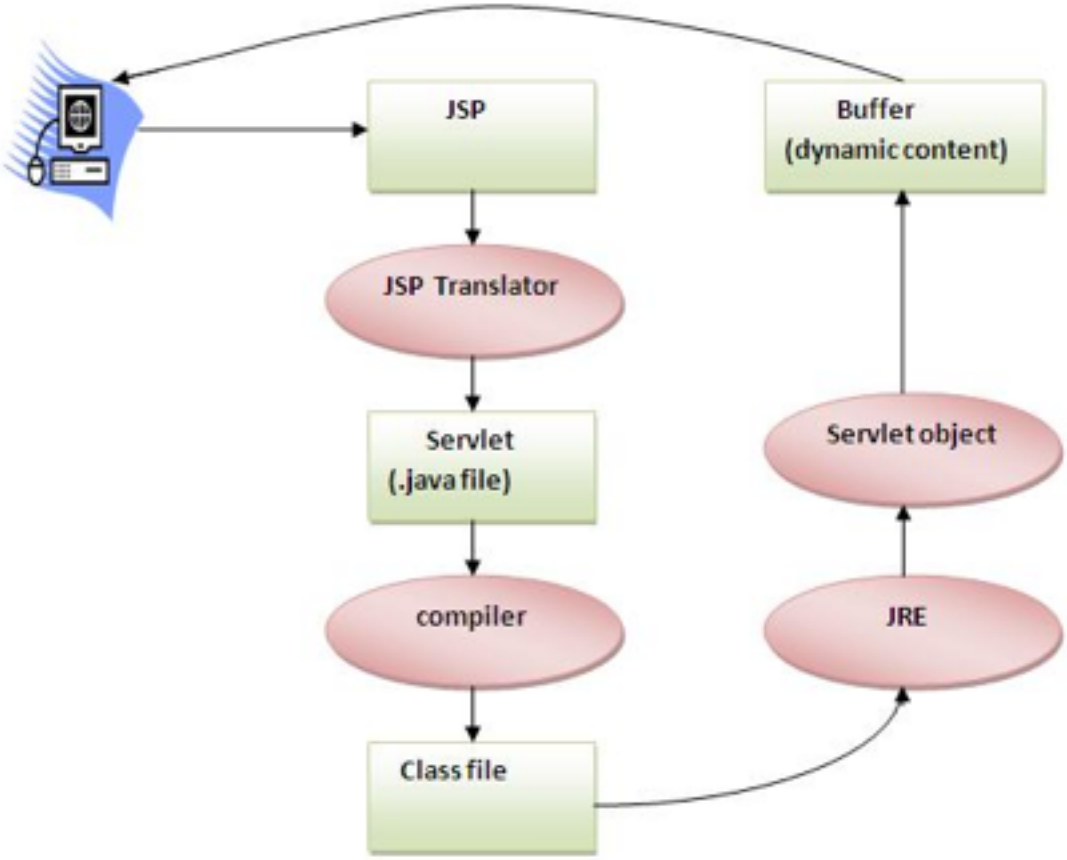
Advantage of JSP over Servlet
There are many advantages of JSP over servlet. These are as follows:

1. JSP is the extension to the servlet technology. We can use all the features of Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
2. JSP can be easily managed because we can easily separate our business logic with presentation logic. In servet, we mix our business logic with the presentation logic.
3. If JSP page is modified, we don't need to redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

Life cycle of a JSP Page:
The JSP pages follows these phases:

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

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



As depicted in the above diagram, JSP page is translated into servlet by the help of JSP translator. The JSP translator is a part of webserver that is responsible to translate the JSP page into servlet. Afterthat Servlet page is compiled by the compiler and gets converted into the class file.
Moreover, all the processes that happens in servlet is performed on JSP later like initialization, commiting response to the browser and destroy.

Creating a Sample JSP Page

Create a JSP page, write some HTML code and save it with ".jsp" extension. Copy this .jsp file to your Tomcat's webapps\ROOT folder and you are ready to access the JSP page on the browser.

How to run a JSP Page?

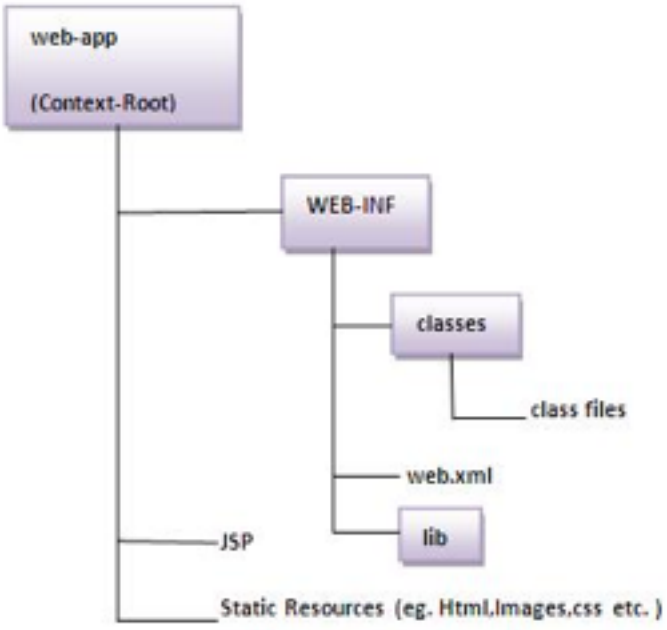
- Start the server
- put the jsp file in a folder and deploy on the server
- visit the browser by the url http://localhost:portno/contextRoot/jspfile e.g.
- http://localhost:8089/myapplication/index.jsp

Do I need to follow a directory structure to run a JSP page?

No, there is no need of directory structure if you don't have class files or tld files. For example, put jsp files in a folder directly and deploy that folder.It will be running fine.But if you are using bean class, Servlet or tld file then directory structure is required.

Directory structure of JSP

The directory structure of JSP page is same as servlet. We contains the jsp page outside the WEB-INF folder or in any directory.



JSP API

The JSP API consists of two packages:

- javax.servlet.jsp
- javax.servlet.jsp.tagext

javax.servlet.jsp package

The javax.servlet.jsp package has two interfaces and classes.

The two interfaces are as follows:

- JspPage
- HttpJspPage

The classes are as follows:

- JspWriter
- PageContext
- JspFactory

- JspEngineInfo
- JspException
- JspError

The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.

Methods of JspPage interface

public void jspInit(): It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
public void jspDestroy(): It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

The HttpJspPage interface

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

Method of HttpJspPage interface:

public void _jspService(): It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.

Scripting Elements

In JSP, there are three types of scripting elements:

Scriptlet Tag
Expression Tag
Declaration Tag

Scriptlet Tag is to contain java code within JSP with the following syntax:

```
<%
        // Java code goes here
%>
```

Expression Tag is to print the content of an object or variable within HTML code with the following syntax:

<%= <the variable OR expression> %>

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.

JSP declaration tag is to define methods like jspInit() and other user defined methods. This declaration section executes only once when the JSP gets loaded first time. The syntax is:

```
<%!
        // Methods goes here
%>
```

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.

What is the difference between the jsp scriptlet tag and jsp declaration tag ?

The jsp scriptlet tag can only declare variables not methods whereas jsp declaration tag can declare variables as well as methods.

The declaration of scriptlet tag is placed inside the _jspService() method whereas the declaration of jsp declaration tag is placed outside the _jspService() method.

JSP Implicit Objects

There are 9 implicit objects available for the JSP page. The Auto Generated Servlet contains many objects like out, request, config, session, application etc.

The 9 implicit objects are as follows:

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

1. **out - Implicit Object**

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

PrintWriter out=response.getWriter();

But, you don't need to write this in JSP.

2. **request – Implicit Object**

In JSP, request is an implicit object of type HttpServletRequest.

3. **response – Implicit Object**

It is an object of type HttpServletResponse.

4. **config – Implicit Object**

In JSP, config is an implicit object of type ServletConfig. This object can be used to get configuration information for a particular JSP page.This variable information can be used for one jsp page only.

5. **application – Implicit Object**

In JSP, application is an implicit object of type ServletContext.This object can be used to get configuration information from configuaration file(web.xml). This variable information can be used for all jsp pages.

```
<context-param>
<param-name>dbname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>
```

In JSP, access this param using the following syntax:

application.getInitParameter("dbname");

6. **session – Implicit Object**

This is an object of type HttpSession. It is used to set or get the current session information.

7. **pageContext – Implicit 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

In JSP, page scope is the default scope.

pageContext.setAttribute(“user”, name, PageContext.SESSION_SCOPE);
String name = (String) pageContext.getAttribute(“user”,PageContext.SESSION_SCOPE);

8. **page – Implicit Object**

In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class.

It is written as:

Object page=this;

For using this object it must be cast to Servlet type.

For example:
<% (HttpServlet) page.log("message"); %>

Since, it is of type Object, it is less used because you can use this object directly in JSP.

For example:

<% this.log("message"); %>

9. **exception – Implicit Object**

In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.

JSP Directives

Page directive
Include Directive
Taglib Directive

Page directive

import
contentType
extends
info
buffer
language

isELIgnored
isThreadSafe
autoFlush
session
pageEncoding
errorPage
isErrorPage

Include directive

%@include file=”test.jsp” %

Taglib Directive

<%@ taglib uri=”URI of the TAGLIB” prefix=”mytag” %>

<mytag:currentdate />

JSP Action Tags

jsp:forward
jsp:include
jsp:useBean
jsp:setProperty
jsp:getProperty
jsp:plugin
jsp:param
jsp:fallback

<jsp:forward page=”” />

<jsp:include page=”’ />

JSP Standard Tag Library

Core tags

SQL Tags

XML Tags

Internationalization tags

Functions tags

Core Tags – c:catch, c:out, c:import, c:forEach, c:if

```
<c:catch>
        Int I = 10 / 0;
</c:catch>


<c:out value=”${param.fname}”> </c:out>

<c:import url=”http://www.google.com” />

<c:import var=”data” url=http://www.google.com></c:import>
<c:out value=”${data}” />

<c:forEach var=”number” begin=”5” end=”10”>
<c:out value=”${number}”></c:out>
</c:forEach>
```

The core group of tags are the most frequently used JSTL tags. Following is the syntax to include JSTL Core library in your JSP:

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

There are following Core JSTL Tags:

Tag	Description
<c:out >	Like <%= ... >, but for expressions.
<c:set >	Sets the result of an expression evaluation in a 'scope'
<c:remove >	Removes a scoped variable (from a particular scope, if specified).
<c:catch>	Catches any Throwable that occurs in its body and optionally exposes it.
<c:if>	Simple conditional tag which evalutes its body if the supplied condition is true.
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<c:when>	Subtag of <choose> that includes its body if its condition evalutes to 'true'.
<c:otherwise >	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'.
<c:import>	Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.

<c:forEach >	The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality .
<c:forTokens>	Iterates over tokens, separated by the supplied delimeters.
<c:param>	Adds a parameter to a containing 'import' tag's URL.
<c:redirect >	Redirects to a new URL.
<c:url>	Creates a URL with optional query parameters

Formatting tags:

The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Web sites. Following is the syntax to include Formatting library in your JSP:

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

Following is the list of Formatting JSTL Tags:

Tag	Description
<fmt:formatNumber>	To render numerical value with specific precision or format.
<fmt:parseNumber>	Parses the string representation of a number, currency, or percentage.
<fmt:formatDate>	Formats a date and/or time using the supplied styles and pattern
<fmt:parseDate>	Parses the string representation of a date and/or time
<fmt:bundle>	Loads a resource bundle to be used by its tag body.
<fmt:setLocale>	Stores the given locale in the locale configuration variable.
<fmt:setBundle>	Loads a resource bundle and stores it in the named scoped variable or the bundle configuration variable.
<fmt:timeZone>	Specifies the time zone for any time formatting or parsing actions nested in its body.
<fmt:setTimeZone>	Stores the given time zone in the time zone configuration variable
<fmt:message>	To display an internationalized message.
<fmt:requestEncoding>	Sets the request character encoding

SQL tags:

The JSTL SQL tag library provides tags for interacting with relational databases (RDBMSs) such as Oracle, mySQL, or Microsoft SQL Server.

Following is the syntax to include JSTL SQL library in your JSP:

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

Following is the list of SQL JSTL Tags:

Tag	Description
<sql:setDataSource>	Creates a simple DataSource suitable only for prototyping
<sql:query>	Executes the SQL query defined in its body or through the sql attribute.
<sql:update>	Executes the SQL update defined in its body or through the sql attribute.
<sql:param>	Sets a parameter in an SQL statement to the specified value.
<sql:dateParam>	Sets a parameter in an SQL statement to the specified java.util.Date value.
<sql:transaction >	Provides nested database action elements with a shared Connection, set up to execute all statements as one transaction.

XML tags:

The JSTL XML tags provide a JSP-centric way of creating and manipulating XML documents. Following is the syntax to include JSTL XML library in your JSP.

The JSTL XML tag library has custom tags for interacting with XML data. This includes parsing XML, transforming XML data, and flow control based on XPath expressions.

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

Before you proceed with the examples, you would need to copy following two XML and XPath related libraries into your <Tomcat Installation Directory>\lib:

- **XercesImpl.jar:** Download it from http://www.apache.org/dist/xerces/j/
- **xalan.jar:** Download it from http://xml.apache.org/xalan-j/index.html

Following is the list of XML JSTL Tags:

Tag	Description
<x:out>	Like <%= ... >, but for XPath expressions.
<x:parse>	Use to parse XML data specified either via an attribute or in the tag body.
<x:set >	Sets a variable to the value of an XPath expression.
<x:if >	Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored.
<x:forEach>	To loop over nodes in an XML document.

<x:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>
<x:when >	Subtag of <choose> that includes its body if its expression evalutes to 'true'
<x:otherwise >	Subtag of <choose> that follows <when> tags and runs only if all of the prior conditions evaluated to 'false'
<x:transform >	Applies an XSL transformation on a XML document
<x:param >	Use along with the transform tag to set a parameter in the XSLT stylesheet

JSTL Functions:

JSTL includes a number of standard functions, most of which are common string manipulation functions. Following is the syntax to include JSTL Functions library in your JSP:

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

Following is the list of JSTL Functions:

Function	Description
fn:contains()	Tests if an input string contains the specified substring.
fn:containsIgnoreCase()	Tests if an input string contains the specified substring in a case insensitive way.
fn:endsWith()	Tests if an input string ends with the specified suffix.
fn:escapeXml()	Escapes characters that could be interpreted as XML markup.
fn:indexOf()	Returns the index withing a string of the first occurrence of a specified substring.
fn:join()	Joins all elements of an array into a string.
fn:length()	Returns the number of items in a collection, or the number of characters in a string.
fn:replace()	Returns a string resulting from replacing in an input string all occurrences with a given string.
fn:split()	Splits a string into an array of substrings.
fn:startsWith()	Tests if an input string starts with the specified prefix.
fn:substring()	Returns a subset of a string.
fn:substringAfter()	Returns a subset of a string following a specific substring.
fn:substringBefore()	Returns a subset of a string before a specific substring.
fn:toLowerCase()	Converts all of the characters of a string to lower case.

fn:toUpperCase()	Converts all of the characters of a string to upper case.
fn:trim()	Removes white spaces from both ends of a string.