WEB TECHNOLOGIES

Unit-VI

INTRODUCTION TO JSP

Enrichment in server side programming is now a need of web application. We prefer component based, multithreaded client server application. A lots of server side technologies such as JSP, servlets, ASP, PHP are used.

Java Server Pages is a kind of scripting language in which we can embed Java code along with html elements.

Advantages of JSP:

- Jsp is useful for server side programming.
- Jsp can be used along with servlets .Hence business logic for any application can be developed using Jsp.
- Dynamic contents can be handled using Jsp because jsp allows scripting and element based programming.
- Jsp allows creating and using our own custom tag libraries. Hence any application specific requirements can be satisfied using custom tag libraries.
- Jsp is a specification and not a product. Hence any variety of applications can be developed.
- Jsp is essential component of J2ee.Hence using Jsp is possible to develop simple as well as complex applications.

Problem with Servlet:

In only one class, the servlet alone has to do various tasks such as:

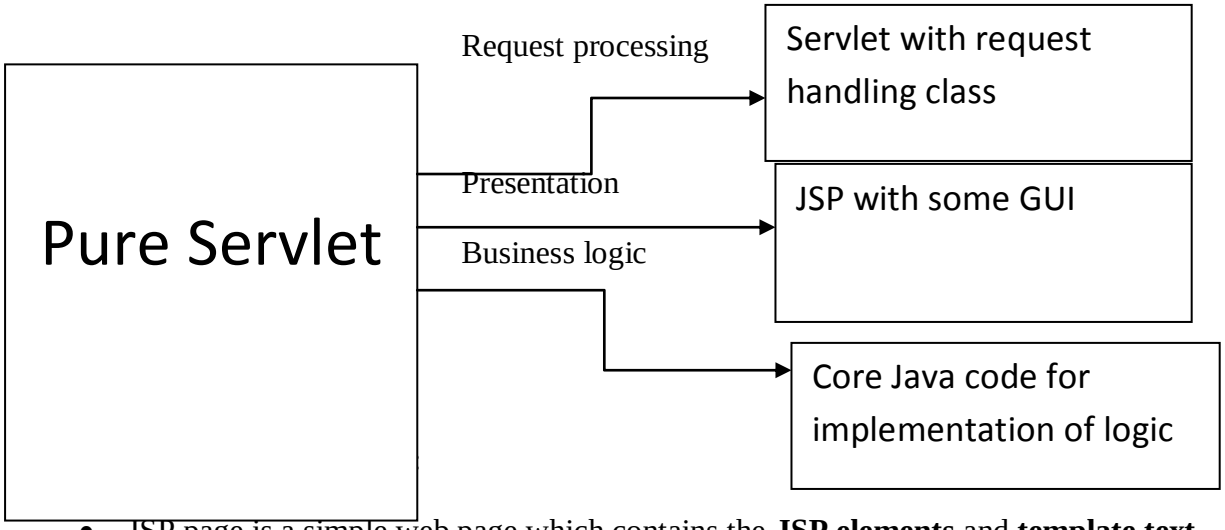
- Acceptance of request
- Processing of request
- Handling of business logic
- Generation of response

Hence there are some problems that are associated with servlets:

- For developing a servlet application, we need knowledge of Java as well as html code.
- While developing any web based application, look and feel of web based application needs to be changed then entire code needs to be changed and recompiled.
- There are some web page development tools available using which the developer can develop web based applications. But servlets don not support such tools. Even if such tools are used, we need to change embedded html code manually, which is time consuming and error prone.

These problems associated with servlets are due to one and only one reason that is servlet has to handle all tasks of request processing.JSP is a technology that came up to overcome these problems.

Jsp is a technology in which request processing, business logic and presentations are separated out.



- JSP page is a simple web page which contains the **JSP elements** and **template text**.
- The template text can be scripting code such as Html, Xml or a simple plain text.
- Various Jsp elements can be action tags, custom tags, JSTL library elements. These JSP elements are responsible for generating dynamic contents.

JSP CODE:

Anatomy of JSP

```
<%@ page import="java.util.Date;" %>
<html>
<head>
<title>
first jsp program</title>
</head>
<body>
<%
out.println("Hello BTech CSE  Students");
out.println("<br><br>");
out.println("Welcome to Jsp Programming");
out.println(new Date().toString());
%>
<center>Have a nice Day</center>
<br><br>
</body>
</html>
```

JSP Element

Template Text

JSP Element

Tempalte Text

```
<%@ page language="java" contentType="text/html" %>          JSP element
<html>
 <body bgcolor="white">                                      template text
<jsp:useBean
 id="userInfo"
 class="com.ora.jsp.beans.userinfo.UserInfoBean">            JSP element
 <jsp:setProperty name="userInfo" property="*"/>
</jsp:useBean>
  The following information was saved:
  <ul>                                                       template text
   <li>User Name:
    <jsp:getProperty name="userInfo"                         JSP element
     property="userName"/>
   <li>Email Address:                                        template text
    <jsp:getProperty name="userInfo"                         JSP element
    property="emailAddr"/>
  </ul>
 </body>                                                     template text
</html>
```

When JSP request gets processed template text and JSP elements are merged together and sent to the browser as response.

JSP Processing:

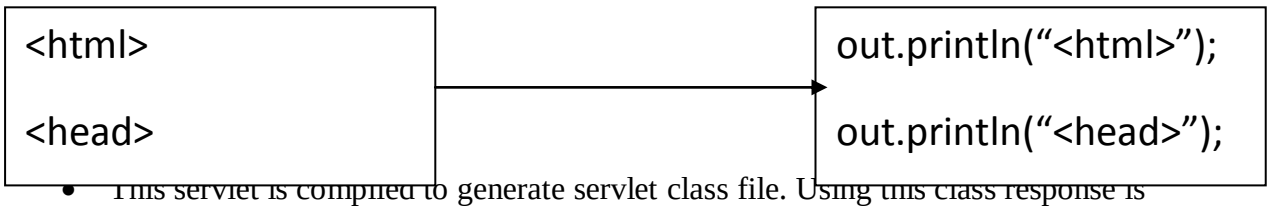
JSP pages can be processed using JSP Container only. Following are the steps that need to be followed while processing the request for JSP page:

- Client makes a request for required JSP page to server. The server must have JSP container so that JSP request can be processed. For instance: let the client makes request for xyz.jsp page.
- On receiving this request the JSP container searches and then reads the desired JSP page. Then this JSP page is converted to corresponding servlet.
- Basically any JSP page is a combination of template text and JSP elements.

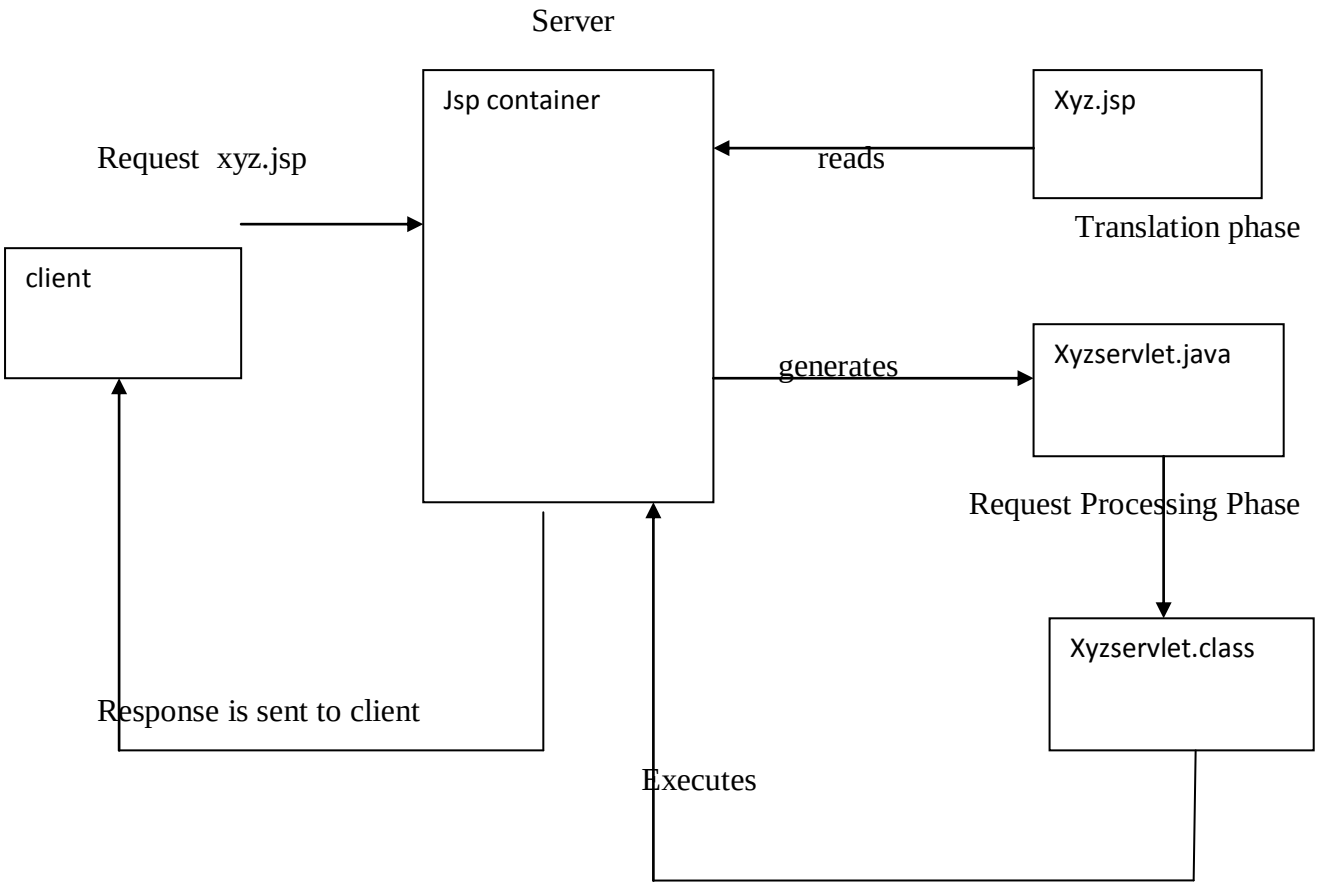
Every template text is converted to corresponding println statement.

Every JSP element is converted into corresponding Java code.

This phase is called Translation phase, output of it is a servlet.for instance:xyz.jsp is converted to xyzservlet.java



- This servlet is compiled to generate servlet class file. Using this class response is generated. This phase is called request processing phase.
- The Jsp container thus executes servlet class file.
- A requested page is then returned to client as response.



JSP Application design with MVC:

The design model of JSP application is called MVC model.MVC stands for Model-View-Controller. The basic idea in MVC design model is to separate out design logic into 3 parts-modelling,viewing,controlling.

Any server application is classified in 3 parts such as business logic,presentation and request processing.

The business logic means coding logic applied for manipulation of application data.

Presentation refers to code written for look and feels of web page like background color, font size etc.

Request processing is combination of business logic and presentation.

According to MVC model,

- Model corresponds to business logic
- View corresponds to presentation
- Controller corresponds to request processing

Advantages of using MVC design model:

The use of MVC architecture allows developer to keep the separation between business logic and request processing. Due to this separation it becomes easy to make changes in presentation without disturbing business logic. The changes in presentation are often required for accommodating new presentation interfaces.

Setting up JSP environment:

For executing any JSP we require:

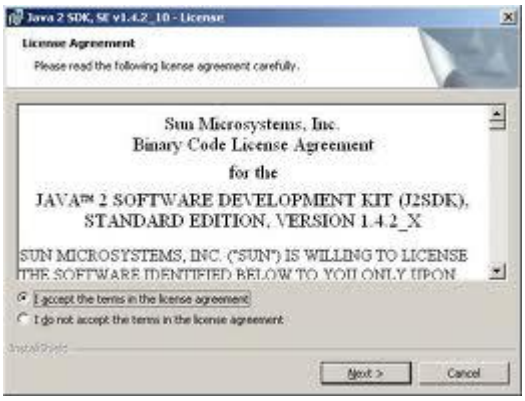
- Java Development Kit
- Any web server such as Apache Tomcat

Installing JDK:

JDK can be downloaded from oracle website on to our machine. After downloading we can install jdk as:

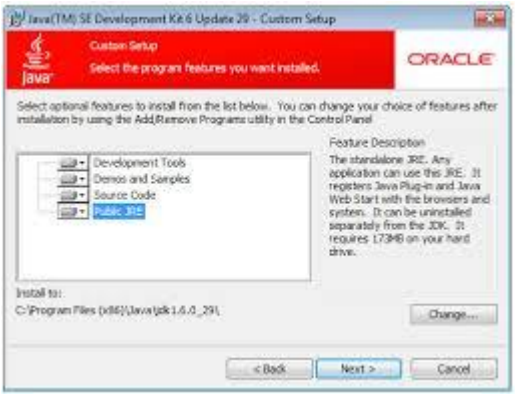
Step-1:

Double click on file download option. And you will get license agreement window. Click on "I Accept" and then "Next".

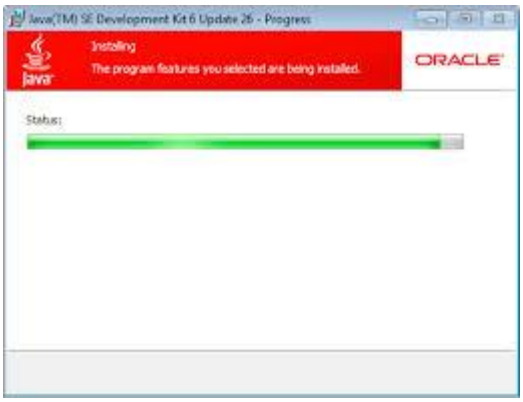


Step-2:

Then the setup screen will appear as follows:



Here we can change default downloading directory to required location. Click on "Next"



Step-3:

After installing JDK further Java Runtime Environment will be downloaded.

After this installation will be completed and following screen will appear.

Step-4:

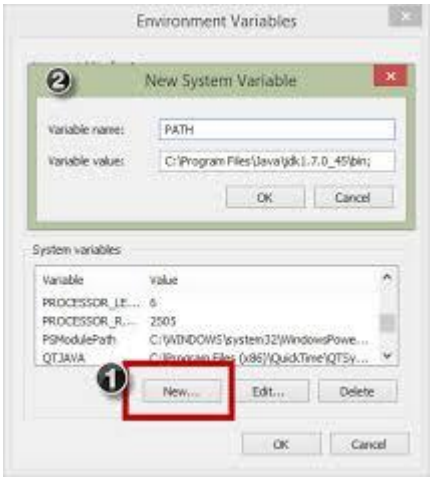
Now we have to set up environment variables after java is installed in our PC.

Go to

Control panel->System Properties->Advanced->

Environment variables

Create a new variable path with its value as location where bin directory of JDK is located.



Now JDK is successfully installed.

INSTALLATION OF TOMCAT:

We can download tomcat from tomcat.apache.org

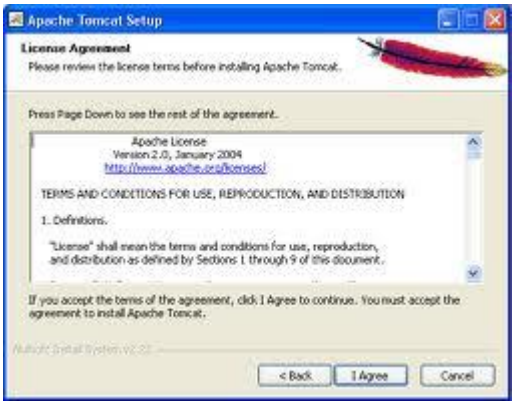
Installation process is as:

Step-1:

When we download tomcat and click on installer following screen appears. Click on “Next”

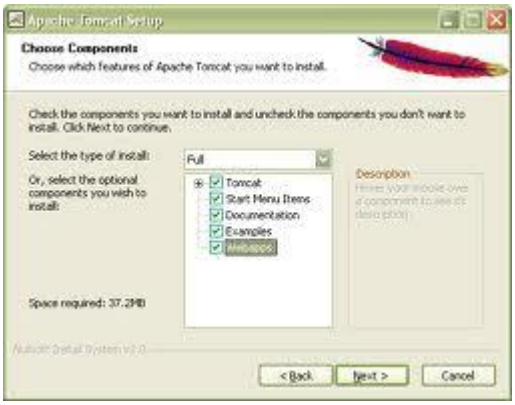
Step-2:

Accept terms by clicking “I Agree”



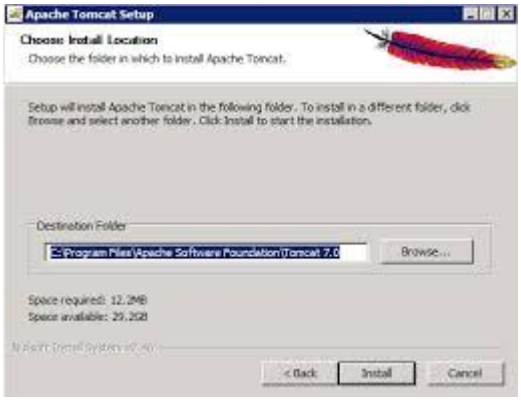
Step-3:

We can choose components to be installed and click on “Next”.



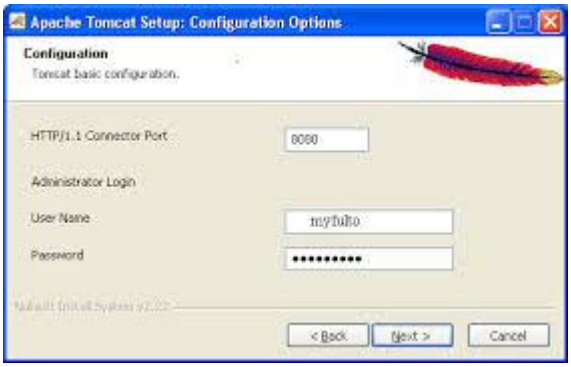
Step-4:

Now installation directory can be chosen. We can change default path by clicking on “Browse” and selecting location.



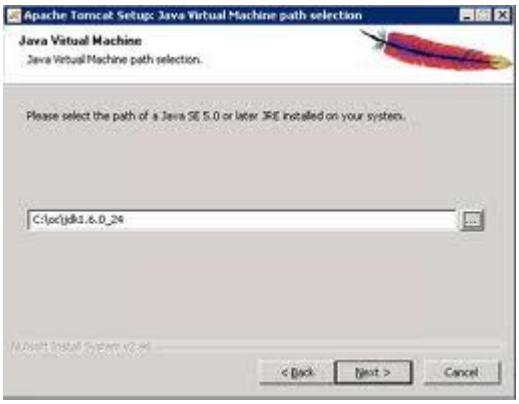
Step-5:

Now by clicking on “Next” we get configuration window. Here we can set connector port. Default port is 8080 but we can set other values also excluding first 1024 values.

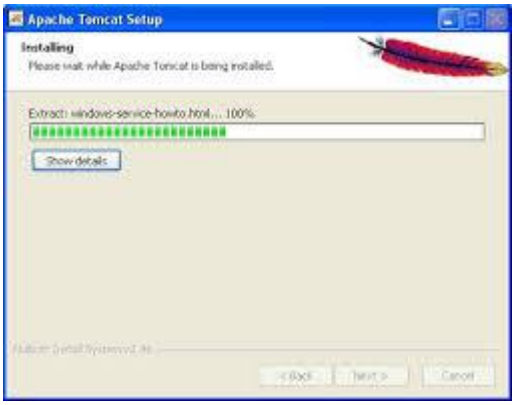


Step-6:

We can also set username and password for administrator login. Then click “Next” button.



Then installation process will start



Click on Finish and installation procedure gets completed.

Step-7:

Setting up JAVA_HOME variable:

Go to Control panel-> System-> Advanced tab->

Environment Variables. Create a new variable “JAVA_HOME” with value as the location where jdk is installed.

Now click on startup.bat batch file and start Tomcat server.

UNIT-VI

Java Server pages

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc.

A JSP page consists of HTML tags 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. They are as follows:

1) Extension to Servlet

JSP technology is the extension to 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) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

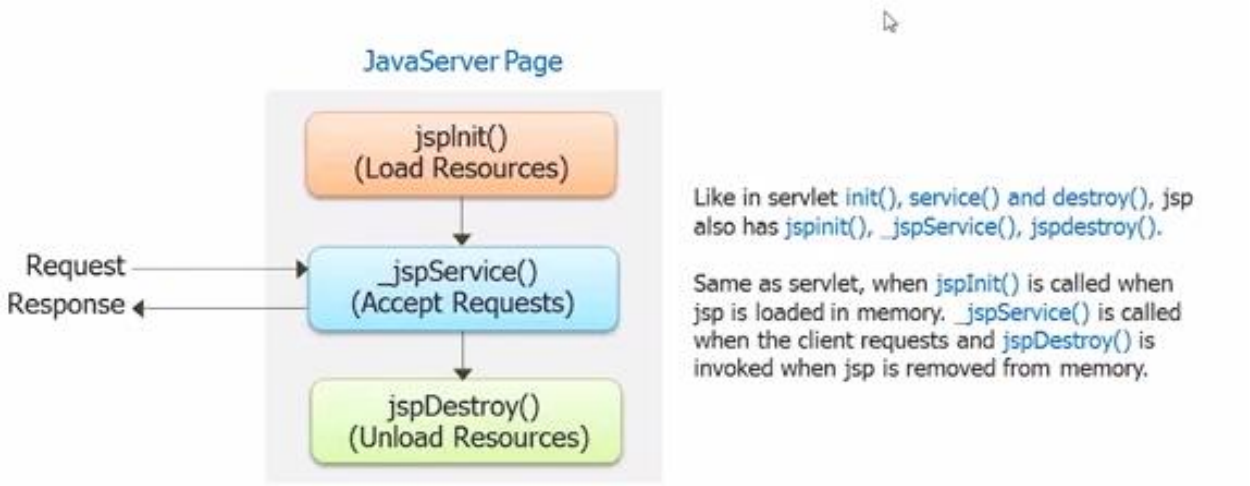
4) Less code than Servlet

In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Life cycle of a JSP Page

The JSP pages follows these phases:

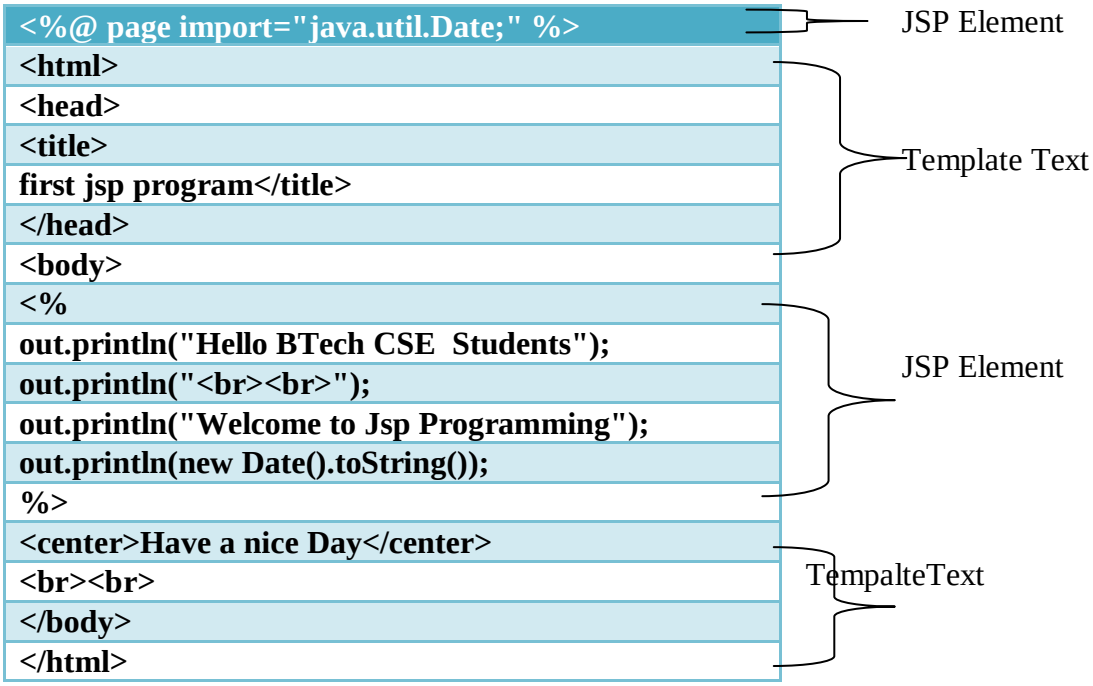
- Translation of JSP Page
- Compilation of JSP Page
- Classloading (class file is loaded by the classloader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Reqeust processing (_jspService() method is invoked by the container).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



ANATOMY OF JSP PAGE:

- JSP page is a simple web page which contains the **JSP elements** and **template text**.
- The template text can be scripting code such as Html, Xml or a simple plain text.
- Various Jsp elements can be action tags, custom tags, JSTL library elements. These JSP elements are responsible for generating dynamic contents.

Anatomy of JSP



When JSP request gets processed template text and JSP elements are merged together and sent to the browser as response.

JSP Processing:

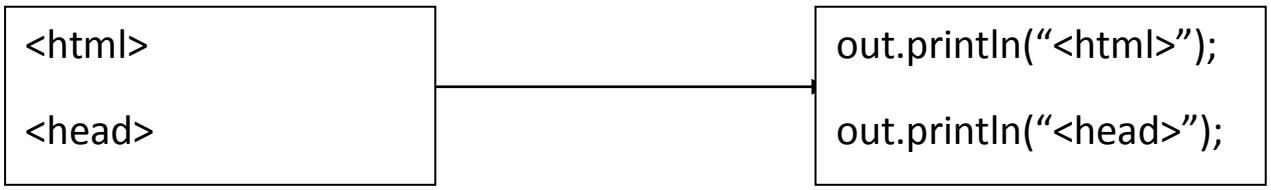
JSP pages can be processed using JSP Container only. Following are the steps that need to be followed while processing the request for JSP page:

- Client makes a request for required JSP page to server. The server must have JSP container so that JSP request can be processed. For instance: let the client makes request for hello.jsp page.
- On receiving this request the JSP container searches and then reads the desired JSP page. Then this JSP page is converted to corresponding servlet.
- Basically any JSP page is a combination of template text and JSP elements.

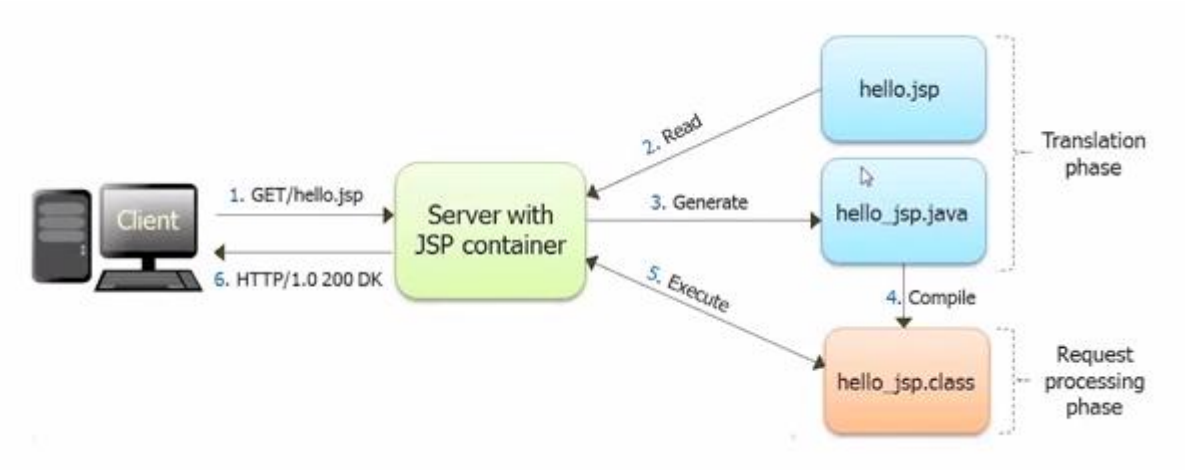
Every template text is converted to corresponding println statement.

Every JSP element is converted into corresponding Java code.

This phase is called Translation phase, output of it is a servlet.for instance:hello.jsp is converted to hello_jsp.java



- This servlet is compiled to generate servlet class file. Using this class response is generated. This phase is called request processing phase.
- The Jsp container thus executes servlet class file.
- A requested page is then returned to client as response.



Model-View-Controller Architecture:

Model:is used for Buisness Logic

Example:Javabean,EJB

View:is used for Presentation Logic

Example:HTML,JSP

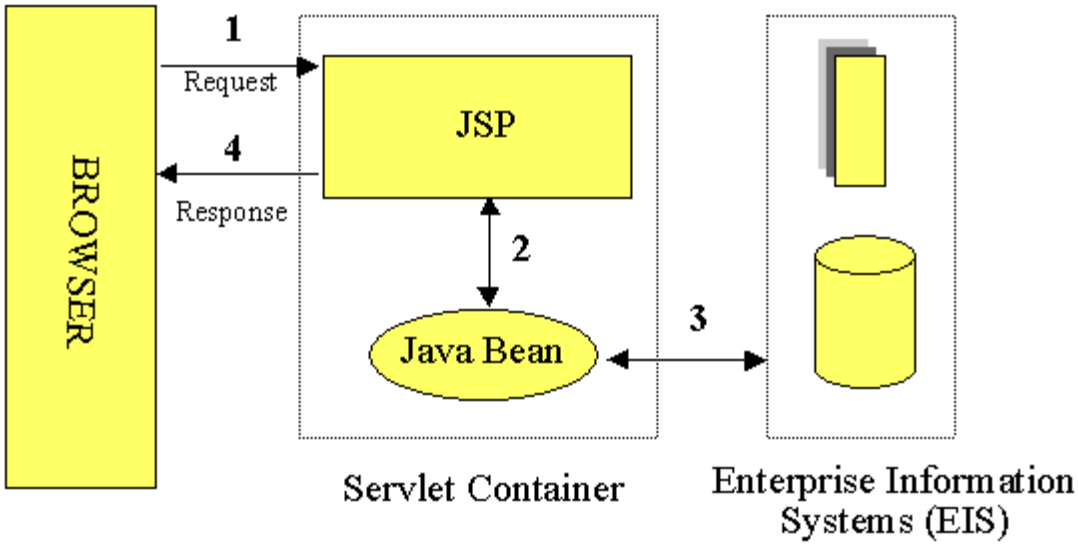
Controller: is used for RequestProcessing

Example:Servlet

JSP Access Models

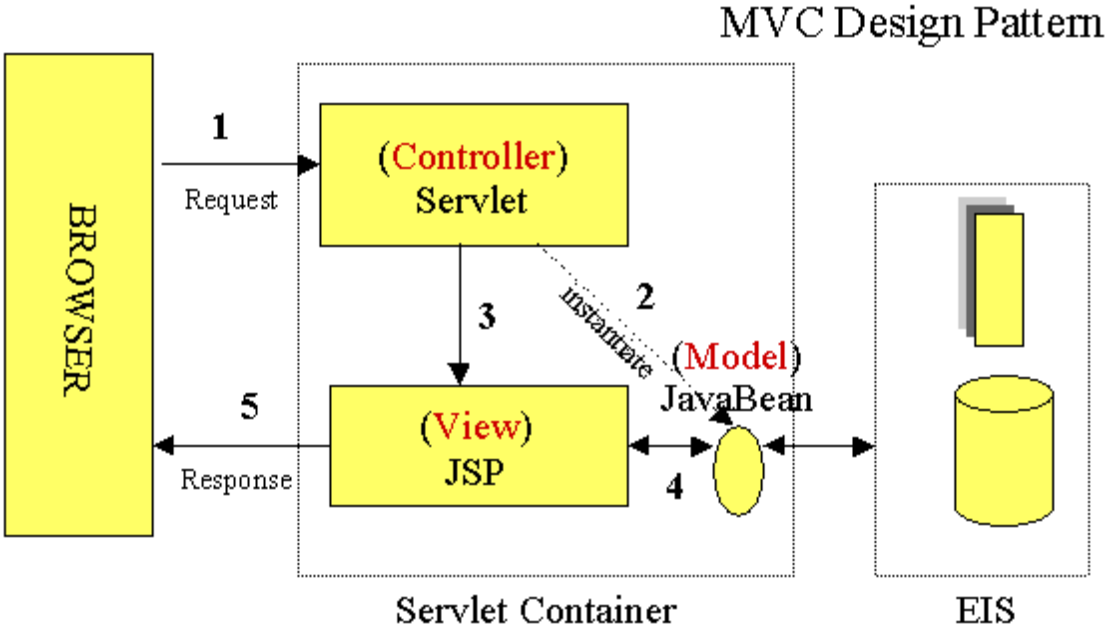
The early JSP specifications advocated two philosophical approaches, popularly known as Model 1 and Model 2 architectures, for applying JSP technology. These approaches differ essentially in the location at which the bulk of the request processing was performed, and offer a useful paradigm for building applications using JSP technology.

Consider the Model 1 architecture, shown below:



In the Model 1 architecture, the incoming request from a web browser is sent directly to the JSP page, which is responsible for processing it and replying back to the client. There is still separation of presentation from content, because all data access is performed using beans.

Although the Model 1 architecture is suitable for simple applications, it may not be desirable for complex implementations. Indiscriminate usage of this architecture usually leads to a significant amount of scriptlets or Java code embedded within the JSP page, especially if there is a significant amount of request processing to be performed. While this may not seem to be much of a problem for Java developers, it is certainly an issue if your JSP pages are created and maintained by designers--which is usually the norm on large projects. Another downside of this architecture is that each of the JSP pages must be individually responsible for managing application state and verifying authentication and security.



The Model 2 architecture, shown above, is a server-side implementation of the popular Model/View/Controller design pattern. Here, the processing is divided between presentation and front components. Presentation components are JSP pages that generate the HTML/XML response that determines the user interface when rendered by the browser. Front components (also known as controllers) do not handle any presentation issues, but rather, process all the HTTP requests.

Here, they are responsible for creating any beans or objects used by the presentation components, as well as deciding, depending on the user's actions, which presentation component to forward the request to. Front components can be implemented as either a servlet or JSP page.

The advantage of this architecture is that there is no processing logic within the presentation component itself; it is simply responsible for retrieving any objects or beans that may have been previously created by the controller, and extracting the dynamic content within for insertion within its static templates.

Consequently, this clean separation of presentation from content leads to a clear delineation of the roles and responsibilities of the developers and page designers n the programming team. Another benefit of this approach is that the front components present a single point of entry into the application, thus making the management of application state, security, and presentation uniform and easier to maintain.

JSP ELEMENTS/Components

3 Types

1. Directive Elements
2. Action Elements
3. Scripting Elements

Directive Elements:

Directive elements are used to specify information about the page

Syntax:

<%@ directivename attr1="value1" attr2="value2" %>

The directive name and attribute names are case sensitive.

Examples of Some Directives:

page

inlcude

taglib

attribute

tag

variable

Page Directive:

This directive can only be used in jsp pages,not tag files.It defines page dependent attributes,such as scripting language,errorpage,buffer requirements

Syntax:

<%@page[autoFlush="true|false"][buffer="8kb|NNkb|none"][contentType="MIMEType"][errorPage="pageorContextRelativepath"][extends="classname"][import="packagelist"][info="info"][isErrorPage="true|false"][isThreadSafe="true|false"][language="java|language"][pageEncoding="encoding"][session="true|false"]%>

Example:

<%@ page language="java" contentType="text/html" %>

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

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

Include Directive:

includes a static file,merging its content with the including page before the combined results is converted to jsp page implementation class.

Syntax:

<%@ include file="page or contextrelativepath"%>

Example:

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

Taglib Directive

Declares a tag library,containing custom actions that is used in the page.

Syntax:

<%@ taglib prefix="prefix" [uri="tagliburi|tagdir="contextrealtivepath"]%>

Example:

<%@ taglib prefix="ora" uri="orataglib" %>

<%@ taglib prefix="mylib" tagdir="/WEB-INF/tags/mylib" %>

Attribute Directive:

This directive can only be used in tag files.It declares the attributes the tag file supports.

Syntax:

<%@attributename="attname"
[description="desc"][required="true|false"][fragment="true|false" |type="attrDatatype"]%>

Example:

<%@ attribute name="date" type="java.util.Date"%>

Tag Directive:

This directive can only be used in tag files.

Syntax:

<%@ tag [body-content="empty|scriptless|tagdependent"][description="desc"][display-name="displayName"][dynamicattributes="attrColVar"][import="packagelist"][language="java|language"][page-encoding="encoding"]

Example:

<%@ tag body-content="empty"%>

Variable Directive:

<%@ variable name-given="attrName"|name-from-attribute="attrname" alias="varName"%>

Action Elements:

Action elements are XML element syntax and represent components that are invoked when a client request the jsp page.

Standard Action Elements:

<jsp:useBean>

<jsp:getProperty>

<jsp:setProperty>

<jsp:include>

<jsp:forward>

<jsp:param>

<jsp:plugin>

<jsp:useBean>:

action associtates a javabean with a name in one of the jsp scopes and also makes it available as a scripting variable

Syntax:

<jsp:useBeanid="beanvariablename"class="classname"[scope="page|request|session|application]/>

<jsp:getProperty>:

action adds the value of a bean property converted to a string to the response generated by the page.

Syntax:

<jsp:getProperty name="beanVariableName" property="PropertyName"/>

<jsp:setProperty>

action sets the value of one or more bean properties.

Syntax:

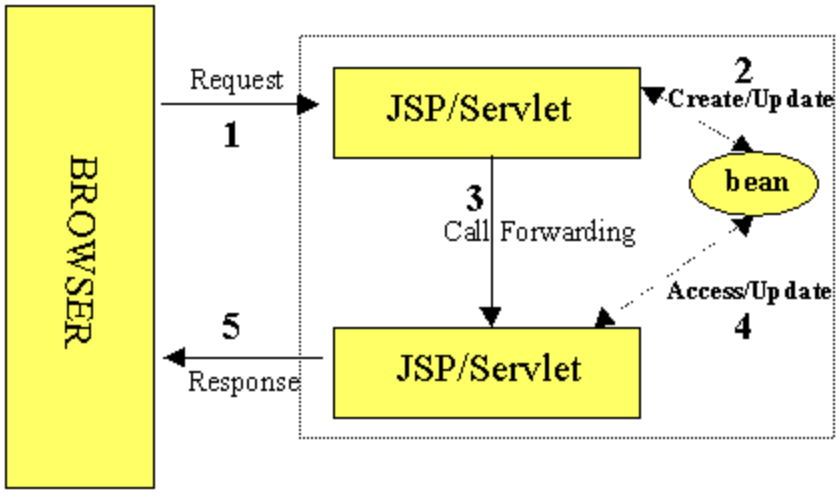
<jsp:setPropertyname="beanVariableName"property="PropertyName"[param="parameterName"|value="value"]/>

Forwarding Requests

With the <jsp:forward> tag, you can redirect the request to any JSP, servlet, or static HTML page within the same context as the invoking page. This effectively
halts processing of the current page at the point where the redirection occurs,
although all processing up to that point still takes place:

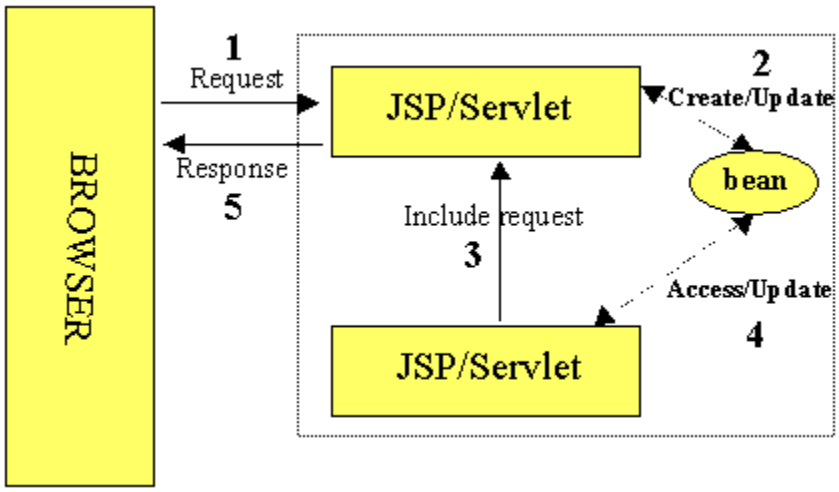
```
<jsp:forward page="somePage.jsp" />
```

The invoking page can also pass the target resource bean parameters by placing
them into the request, as shown in the diagram:



Including Requests

The <jsp:include> tag can be used to redirect the request to any static or dynamic
resource that is in the same context as the calling JSP page. The calling page can also pass the target resource bean parameters by placing them into the request, as shown in the diagram:



For example:

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

Exception Handling

JSP provides a rather elegant mechanism for handling runtime exceptions. Although you can provide your own exception handling within JSP pages, it may not be possible to anticipate all situations. By making use of the page directive's errorPage attribute, it is possible to forward an uncaught exception to an error handling JSP page for processing.

For example,

```
<%@ page isErrorPage="false" errorPage="errorHandler.jsp" %>
```

informs the JSP engine to forward any uncaught exception to the JSP page errorHandler.jsp. It is then necessary for errorHandler.jsp to flag itself as a error processing page using the directive:

```
<%@ page isErrorPage="true" %>
```

This allows the Throwable object describing the exception to be accessed within a scriptlet through the implicit exception object.

K.Yellaswamy ,AssistantProfessor|CMR College of Engineering & Technology
E-mail:toyellaswamy@gmail.com