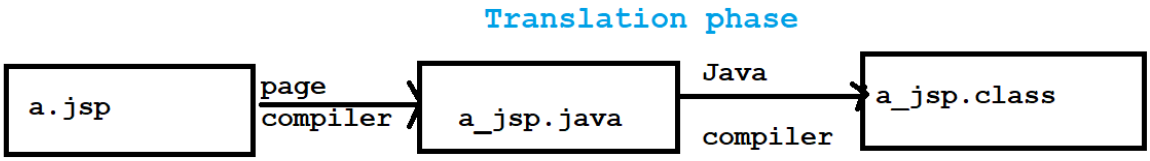
1. Servlet is a server side technology for developing web applications given by sun microsystem to overcome the drawbacks identified in CGI technology.
2. To develop web applications using a web technology programmer needs to write more Java programming code..
3. Programmers are not interested to develop applications with servlet technology due to large programming code.
4. Meanwhile Microsoft release another technology with name **ASP(Active Server pages)** for creating web applications using set of predefined tags and some basic VBSCRIPT code.
5. Developers are attracted to ASP due to less programming code.
6. In order to attract the developers to the sun microsystem technologies side also to reduce programming burden on the developers , sun microsystem released server side technology with the name of **JSP.**

Jsp Architecture:

Jsp architecture contains two phases.

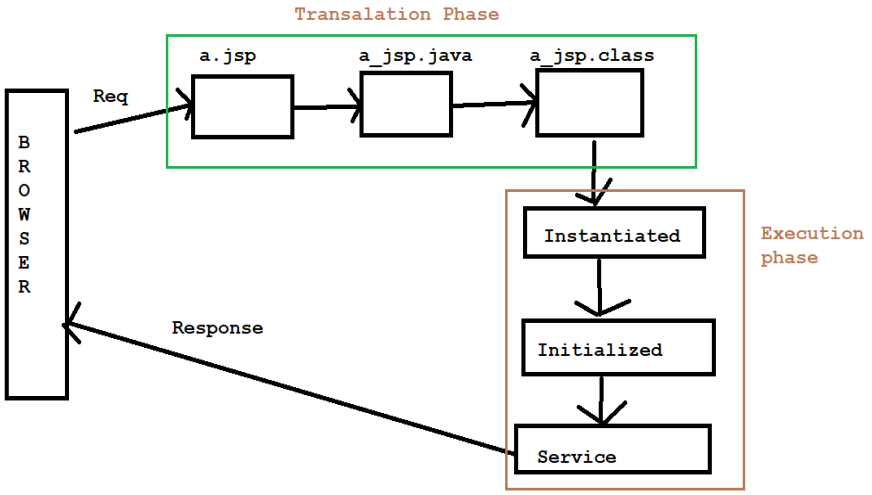
1. Translation Phase
2. Execution Phase

- Every JSP page is translated into an equivalent servlet at server.
- A JSP container has a page compiler .it will translate a jsp to equivalent servlet.
- A page compiler in Tomcat is JASPER and a Page compiler in weblogic in JSPC.
- For example, if JSP page is ‘a.jsp’ then jasper compiler translates a.jsp as a_jsp.java and JSPC compiler translates a.jsp as __a.java
- Translating a.jsp file as .java and then compiling it as .class.
- In a translation page two compilers are used
    1. Page compiler
    2. Java compiler

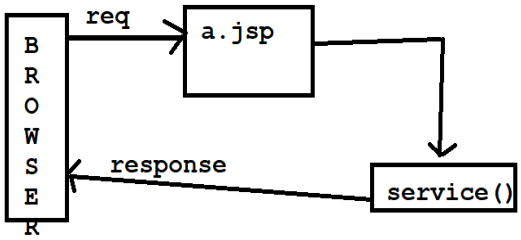


- In Execution phase, a servlet is instantiated then initialized then it's service is executed.
- When a first request comes to JSP page or when a first request comes after modification of jsp page the a jsp container(web container) ,first starts translation phase then start execution phase.
- For the remaining requests to a jsp page container only call the service.

First Request/First Request After Modification - Flow:



Remaining Requests - Flow



Q) What happens, if a server is restarted,after executing for JSP several times?

1. When a server is shut down that automatically a servlet object is destroyed but corresponding .java /.class files of jsp are not deleted from server.
2. After restarting the server when we send a request to the same jsp page then container creates an object for existing servlet class then initializes it ,then it calls service method.it means container again start **execution phase** but not **translation phase**.

Q) How does a jsp container recognise whether a jsp page is modified or not ?

A jsp container internally using some files comparison tool will verify the previous timestamp of a page and current timestamp of a page both are the same or not. If same then container directly executed service.if not same then container start translation phase phase following and execution phase.

Life cycle methods of JSP:

1. **Server side** ,a jsp translator converts a jsp page to an implementation class. This implementation class is not directly extending HttpServlet class but it extending indirectly.
2. At the time of translation,a jsp translator extends the implementation class from a superclass given by the jsp container.
3. A container given superclass extends HTtpServlet.
4. For example,in TOmcat server a jsp page implementation class extends HttpJspBase class given by tomcat server.
5. HttpJspBase class extends HttpServlet class.
6. Every Jsp page implementation class indirectly implements the HttpJspPage interface of javax.servlet.jsp package.
7. HttpJspPage interface has provided three lifecycle methods for JspPages.
    a. jspInit()
    b.jspDestroy()
    c._jspService(HttpServletRequest req ,HttpServletResponse res)
8. Among the three life cycle of jsp,at the time of writing a JSPPage a programmer can override jspInit() and jspDestroy() methods of jsp but a programer cannot override _jspService.
9. TO intimate the programmeers that jspService() is special method the name is started with _ .
10. At the of creating a JSP page implement class, a jsp translator override _jspService().
11. At the runtime,JSP lifecycle methods are not directly called.
12. Container calls servlet lifecycle methods and it is internally called jsp lifecycle methods.

13. The logic of calling jsp life cycles methods from servlet lifecycle methods will be defined in super class given by the server.

Different Categories of tags in JSP:

1. Elements of JSP
2. Custom tags
3. Expression language
4. JSTL (jsp Standard tag library)

Elements of JSP:

Jsp elements categorized into 3 types

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

Scripting Elements:

I. Scriptlet

- If we want to insert some java code in a jsp page and if we want to run that code for each request to the jsp page then we can use scriptlet tag of jsp.
- In a jsp page we can write the scriptlet tag in a html syntax of the tag or in xml syntax of the tag.
- We can write a jsp tags in two ways

  Html syntax

  <% java code %>

  xml syntax

  <jsp:script>
  Java code
  </jsp:script>
- At the time of translation the java code inserted in the scriptlet tag goes to the to _service() method. This method will be called for each request.

- In a jsp page we can write multiple scriplet tags.
  <% count++ %>
  <% int b=10; %>
- We can define variables in scriplet tag.these variables will become local variables to _jspService() at translation time.
- We cannot define methods in a scriptlet tag because a method cannot be defined as another method.

II. Expression

- This scripting elements is used for inserting an expression into a jsp page.
- The result of expression will be directly displayed an browser
- Every expression tag of jsp will be converted to out.print statement of java at translation time int _jspService()
- We can write expression tag in two ways called html and xml syntax.
- Expression tag will also be executed for each request
- In expression tag we should not use delimiters like (; , :) etc

a.jsp (html)
<% expression %>

a.jsp(xml)
<jsp:expression>

<jsp:expression>

III. Declaration

- This tag is used for creating the global variables and methods in the jsp page.
- A declaration tag can be written a jsp page using two ways (html,xml)
  <%! Variables; methods %>

<jsp:declaration>
Variables:
Methods:
</jsp:declaration>

- The code inserted in a declaration tag will be sent to directly in to the class at translation time.
- The methods that are created in declaration tag can be called from script tag are from expression tag.
- If a method of declaration tag is called from a script as expression , that method will be executed from each request.
- Q) can we use jsp implicit objects in a declaration tag?

  A)No, the jsp implicit objects are only available to the code that goes to the jsp service method.
  - Declaration tag code goes out of jsp service method.so, we cannot use jsp implicit objects in declaration tag.
  - EX:-

```
<%!
P void m1( )
{
out.println(“hello”);
%>
}
```

Q)Which scripting element is returned to override init and destroy methods of jsp?

A)Declaration tag

```
<%!
Pv jspinit( )
}
}
Pv jspDestroy( )
}
}
%>
```

Commands in jsp:-

In a jsp page we can define three types of comments.

1. Html comment
2. Jsp comment
3. Java comment

- In a js[page we can write html tags also. So html comments are allowed.
  <!-- comment→
- We can also write a comment using jsp given tag
  <%--comment–%>
- We can also write java comments declaration tag and scripted tag but we cannot write comments in expression tag.
- Often translation of js[, html comments and the java comments are visible in the class, but jsp comments are invisible. So jsp comments are called hidden comments.
  JSP configuration:
  ——----------

- Jsp files stored in a root folder of the web application so that are called public files
- A public file of a web application can be directly requested from the browser using the file name.
- If we want we can configure a jsp file in web.xml. If configured then we can sent a request with a url pattern.
- Jsp configuration is almost only , but in place of servlet class tag we write jsp-file tag.

```
    <servlet>
<s-n>XXXX</s-n>
<jsp-file>/a.jsp</jsp-file>
</servlet>
<servlet-mapping>
<s-n>XXXX</s-n>
<u-p>/j1</u-p>
</servlet-mapping>
```

- We can request to the jsp like the following
  1. http://LH:2015/root/j1

2. http://LH:2015/root/a.jsp

Example:

—---------

- ❖ We are creating a jsp page for counting the no of times a request is given to a jsp page.
- ❖ Deploy the jsp 1.folder in tomcat/w:ebapps directions and then start the service
- ❖ Send a request from the browser like the following http://localhost:/2015/jsp1/one.jsp

NOTE:

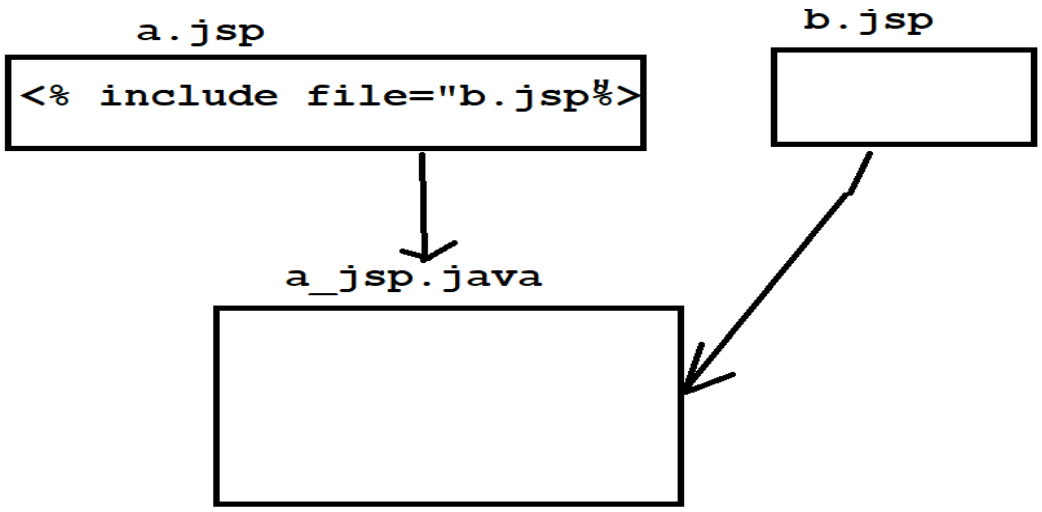
We can find the internal servlet class of a jsp page at C:\Program Files\Apache Software Foundation\Tomcat 9.0\work\Catalina\localhost\FirstJsp

Directives in JSP:

- ❖ Directive elements are given by the jsp specification to give some translation time instructions to the page translation of jsp.
- ❖ Jsp specification has given three types of directives
  1. Include
  2. Page
  3. taglib
- ❖ In a jsp we can write a directive tag in a html syntax an xml syntax
- ❖ <% directive nameattributes %>(html)
- ❖ <jsp:directivename attributes%>(xml)

include directive:

- ❖ We use this directive to include source code of another page into the same servlet when translation the current page.
- ❖ Include directive has only attribute called file.
- ❖ Include directive includes source code of another page,so it is called static including.



page directive:

import:

- This attribute is used to import one or more packages in jsp.
- At translation time ,page compiler adds on top of the servlet class.
- In jsp page, a package can be imported either on the top of the jsp or at the bottom or in the middle in the page.

A.jsp

<%@ page import=”java.util.*” %>

<%@ page import=”java.util.* , java,sql.*” %>

session:

- This attribute tells page compiler whether to add session object to the servlet at translation time or not.
- The default value of session attribute is true. It means session object is added to the servlet.
- If session = ”false” then at translation time, session object will not be added to the servlet.
- We can repeat session attribute in a JSP page, but each time it’s value must be same.
- Ex : 1 a.jsp

- <%@page session = “true”%>

NOTE:

- If session = “ false” then implicit object object session is not allowed to use in that JSP page.

EX: a.jsp

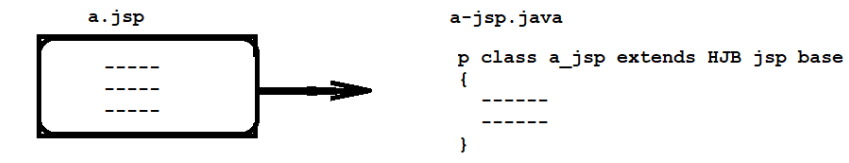
```
<%@page session = “False”%>
<%
   String str = session.getId();
%>
```

isThreadSafe:

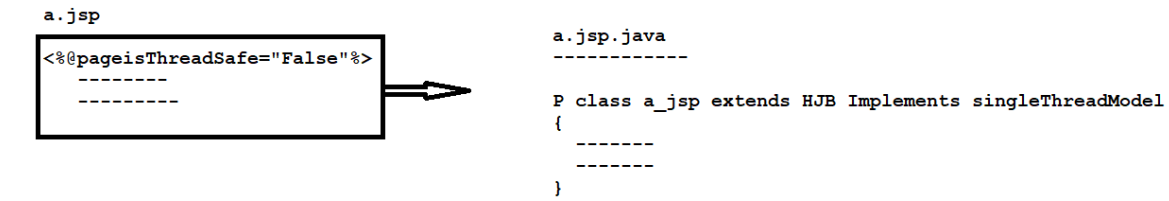
- This attribute tells page compiler, weather internal translated servlet should implement singleThreadModel interface or not.
- The default value of this attribute is true.
- If isThreadSafe = “ true “, then page compiler does not implement the servlet class from singleThreadModel interface.
- If isThreadSafe = “ false “, then page compiler implements servlet from singleThreadModel interface.
- If servlet implements singleThreadModel Then only one thread is allowed to access that servlet object at a time.

Ex:

Ex:1



Ex:2



Errorpage:

- This attributes tells page compiler to add the code for forwarding a request if an exception occurs in the current jsp page
- A request is forward to the given error page, only if exception is occurred, otherwise not forwarded.

Ex:

Ex:

a.jsp

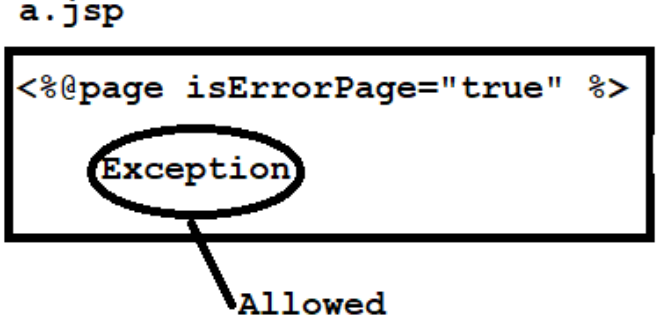
```
<%@page errorpage ="ex.jsp"%>
   ---------
   ---------
   ---------
```

isErrorPage:

- This attribute tells page compiler that weather a jsp can act as error page or not
- Default value of this attribute is false. It means a JSP page is not acting as an error page.
- We can make a JSP page as an error page by setting true to the is error page attribute.
- If isErrorPage = " true" then only exception object(Implement object) can be used in JSP

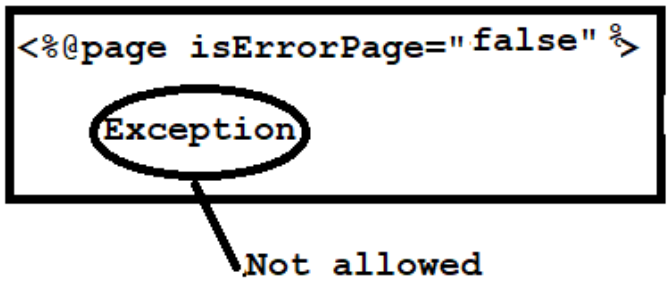
Ex:

a.jsp



Ex:

a.jsp



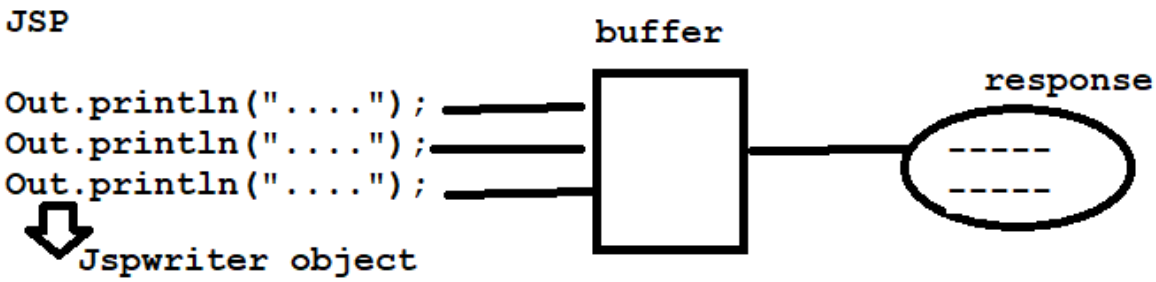
ContentType:

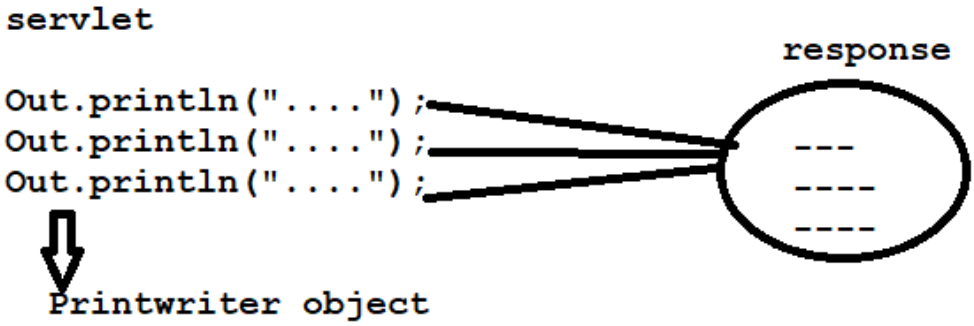
- In Jspm we can set MIME type of response in two ways
  1. By using contentType attribute
  2. By calling setContantType() method
- The default value of contentType attribute is Text/html
  1. <%@ page contantType = “text/xml” %>
  2. <%
     response.setcontentType(“text/xml);
     %>

Buffer:

- In jsp, out is an implicit object and out object is created for JSPwriter class.
- When out object is created in JSP then automatically a buffer of size 8 Kb is also allocated to that out object.
- When writing some response data to response object, out object of JSP will write the data to the buffer. Latter the data is transferred from buffer to response object.
- The buffer attribute of page directive is used to either increase or decrease a buffer size an buffer can be removed.
- In servlet technology , we can printwrite class object, to write the response data to the response object with [printwrite class object buffer is not created.
- A difference between JSP writer and printwriter classes is , JSPwriter class object manages buffer but printwriter class object does not manage buffer.

- JSPWriter = Printwriter+Buffer
- We can make JSP write class as equal to printwriter class by removing the buffer






```
<% @Page buffer = "12 kb" %>
<% @page buffer = "4 kb" %>
<% @page buffer = "none" %>
```

autoFlush:

- This attribute is used either to enable an to disable autoFlush mode of buffer.
- The default value of this attribute is true.It means autoflush mode is enabled.
- If we want to disable autoFlush mode then autoFlush = " false".
- If autoFlush mode enabled then container will take care about flushing the buffer at the appropriate times.
- If autoFlush mode diabled then we need to manually flush the buffer by writing out.flush();

- Flushing a buffer means transferring the data from buffer to response object ,
- If autoflush mode is disabled and we call flush() method after buffer reaches to full then " BufferOverFlowException" will be thrown so it is always recommended to enable autoflush mode.

language:

- This attribute tells page compiler, about language used in the scripting tags of JSP.
- As of now, we can write only java code in scripting tags. So default value and the only possible value is Java

  <% @ page language = "java" %>

Info:

- This attribute is used to write description of a page
- This description also looks like a comment only but it is a readable comment.
- If we add info attribute then at translation time, getServletInfo() method will be added to the servlet class.
- In a JSP page we can read the value of info attribute by calling getServletInfo() method .

a.jsp

```
<% @Page info= " This is a simple page"%)
<%
  String str = getServletInfo();
%>
```

a_jsp.java

```
p class a_jsp extends HJB
{
 P String getServletInfo()
{
retun "This is sample page"
{
P void jspService(Req, req) throws SE, IOE
}
}
}
```

extends:

- When a JSP page is translating, a page compiler extends a servlet class from a super class provided by the container.
- For Example in tomcat, the super class is HttpJspBase.
- SUppose if we want to create a super class and if we want to tell the translator to extends our super class the we need to use extends attribute.
- Creating a super class will be a burden in the programmer. So this extends attribute is very rarely used.

Ex:

```
Package com.chenchu.jsp;
public class MyOwnServlet extends HttpServlet
{

}
```

a.jsp

```
<@% page extends=”com.chenchu.jsp.MyOwnServlet” %>
```

a_jsp.java

```
public class a_jsp extends MyOwnServlet{
{
}
```

isELIgnored:

- This Attribute tells, page compiler whether the Expression language is enabled or disabled.
- The default value for this attribute is false.it means Expression language enabled.
- If we can set true then Expression Language is disabled.
- If EL is enabled then the translator translates EL statements to equivalent java code otherwise not translated.

Attribute	**Default Value**
import	NA
session	true
isThreadSafe	true
errorPage	NA
isErrorPage	false
contentType	text/html
buffer	8kb
autoflush	true
language	java
info	NA
extends	NA
isELIgnored	false

Action Tags in JSP:

- Action tags are used for the following communication
    1. Jsp to JSP/Servlet/html

2. Jsp to Applet
3. Jsp to Bean class

❖ There are mainly 6 action tags with 3 sub-action tags in JSP.

Action tags	Sub action tags
A. <jsp:forward> B. <jsp:include> C. <jsp:plugin> D. <jsp:useBean> E. <jsp:setProperty> F. <jsp:getProperty>	a. <jsp:param> b. <jsp:params> c. <jsp:fallback>

❖ Action tags of jsp can be written only in xml syntax.

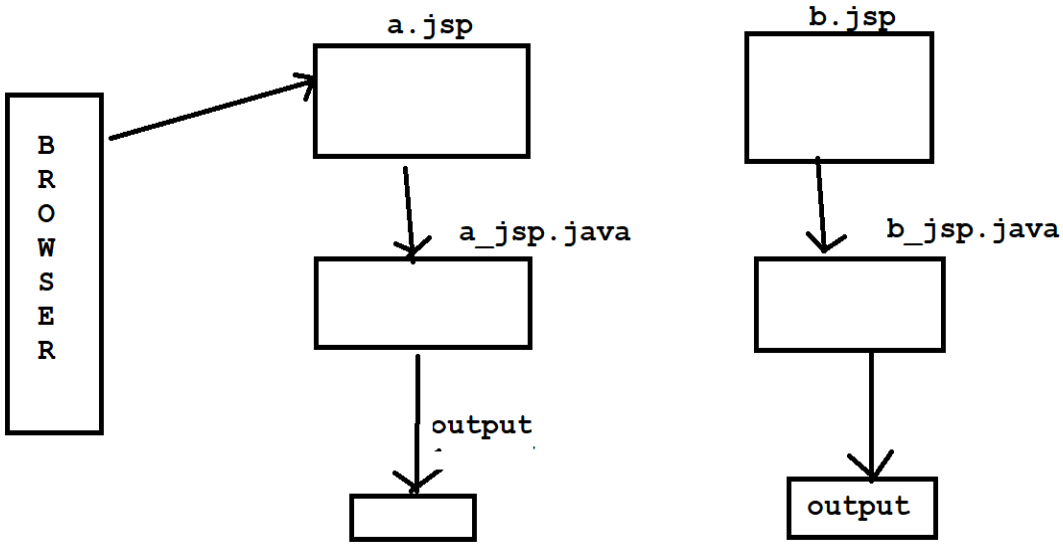
<jsp:forward>

1. This action tag is used for forwarding a request from Jsp page to another JSP page or Servlet or Html.
   <jsp:forward page="b.jsp"/>
   <jsp:forward page="srv1"/>
   <jsp:forward page="b.html/>
2. For a request,we can add one or more parameters to the request by adding sub action tag <jsp:param>
3. If any request parameters are sent from browser then they are automatically forwarded along with the request , to add additional parameters by forwarding the request ,we can use <jsp:param> sub action tag.

```
<jsp:forward page="b.jsp">
   <jsp:param  name="p1" value="100"/>
   <jsp:param  name="p2" value="200"/>
</jsp:forward>
```

<jsp:include>

1. This action tag is used for include the response of another jsp or servlet or html into response of current jsp.
2. This action tag includes the response at runtime, so it is dynamic including.
3. Suppose if a.jsp includes b.jsp then for both a and b servlets are created at translation time and only output is included at runtime.



4. In jsp,we have both include directive and include action,for including one page to another page,the Differences between these are like the following.

Include directive vs include action:

include directive	include action
1. In include detective source code of another page is included,so it is static including	1. In include action, o/p of another page is included at runtime, so it is dynamic including.
2. We cannot pass	2. We can pass expression

expression tag as a value to the file attribute of include directive.	tag as a value to the page attribute of include.
3.if any changes are made in source code in the include page,then the changes are not included into the source page servlet.	3. If the changes are made in included page then the changes are reflected in output
4.include directive can be added to JSP in html or xml syntax	4. Include action can be added to jsp in xml syntax only

Ex:

```
<%
    String str="b.jsp";
%>

<%@ include file="<%= str %>" %>    —- Invalid
<jsp:include page="<%= str%>" />    —–– valid
```

5. While including the output of another page into the current page, request is temporarily transferred to destination page, with request we can send some parameters to the included page by adding <jsp:param> sub action tag.

```
<jsp:include page="b.jsp">

    <jsp:param name="p1" value="100" />
    <jsp:param name="p2" value="200" />

</jsp:include>
```

6. Include action tag has page attribute and also autoflush attribute .

7. Flush attribute tells the container to clear the buffer,before including another jsp page.
8. default value of flush attribute is false.to flush the buffer,we can set flush=”true”

<jsp:plugin>

- This action tag integrates a jsp with an applet.
- When a jsp page wants to show the response an browser in graphics then jsp interact with an applet.
- <jsp:plugin> action has the following attributes.

```
<jsp:plugin type=”applet” code=”pack1.MyApplet.class”
                codebase=”applet” width=”300” height=”300” />
```

- the <jsp:plugin> action tag has two sub action tags.
  a. <jsp:params>
  b. <jsp:fallback>
- These two sub-actions tags can be used on;y with in <jsp:plugin>
- <jsp:param> contains at least one<jsp:param> to pass a parameter to an applet.
- <jsp:fallback> sub action tag is used to define an alternate message to display on the browser, if an applet is not display on the browser.

```
<jsp:plugin type=”applet” code=”Myapplet.class” codebase=”.”
                width=”300” height=”300”>

        <jsp:params>
            <jsp:param name=”p1”  value=”10” />
            <jsp:param name=”p2” value=”30” />
        <jsp:params>
    </jsp:plugin>
```

<jsp:useBean>

- We can create a java bean in a jsp Application, if any common java code exists in multiple jsp pages.
- In jsp pages has any common java code then to get the re-usability. We can seperate the java code into Java Bean class.
- A java bean class is java class ,which follows the following rules.
    1. Class must be public
    2. Class must contains public default constructor
    3. Private properties must contain either setter and getter or both methods.
    4. class can at most implement either serializable or Externalizable interface.
- <jsp:useBean> action tag either reads a java bean object from a given scope, if it exists or creates a new object for the java bean class and stores that in given scope.
- Jsp technology has define four scopes,where each scope is used to share the data across multiple pages.
    1. page
    2. request
    3. Session
    4. application

  <jsp:useBean id=”objname” class=”fullyqualifiedName” scope=”page” />

- When <jsp:useBean> tag finding bean object in given scope with id(id ack as key)
- When creating a new object for a bean class id acts as object name.
- To use any java class object in jsp page,that class must be store within a package.
- The default scope of a bean is page.

  <jsp:useBean id=”obj1” class=”pack1.Test” />

<jsp:setProperty>

- This action tag will set a value to a bean property by calling setter() method of a property.

- For example,we can have a bean called LoginBean with properties uname and pwd. A Jsp page will set the values to both the properties like the following.

```
<jsp:setProperty name="obj1"   property="uname" value="chenchu"/>
<jsp:setProperty name="obj1"   property="pwd" value="java"/>
```

- The value of the name attribute must be the same as the value of id attribute of <jsp:useBean> tag.
- A jsp can directly set a request parameter value to a property by using param attribute.

```
<jsp:setProperty name="obj1" property="uname" param="username"/>
```

- Name and property attributes are mandatory.we can use either param or value but not both at a time.
- If request parameters name are same as bean properties names then we can write a special value to the property called '*****'
- For example, request parameters names are uname,pwd and Bean properties names are aslo uname,pwd then we can set request parameters values to bean properties by property '*****"

```
<jsp:setProperty name="obj1" property="*" />
```

- A bean has three properties and there is only two request parameters whose names are matched with Bean Properties then we can set the value to the third property with value attribute.
- Suppose request parameter names are uname,pwd and Bean properties are uname,pwd,email then we can set the values like the following.

```
<jsp:setProperty name="obj1" property="*"/>
<jsp:setProperty name="obj1" property="email"
                                value="abc@gmail.com" />
```

<jsp:getProperty>

- This action tag read value of the bean property by calling getter method.
- This action tag will print a Bean property value on browser.
- This <jsp:setProperty> action has only two attributes called name and property.

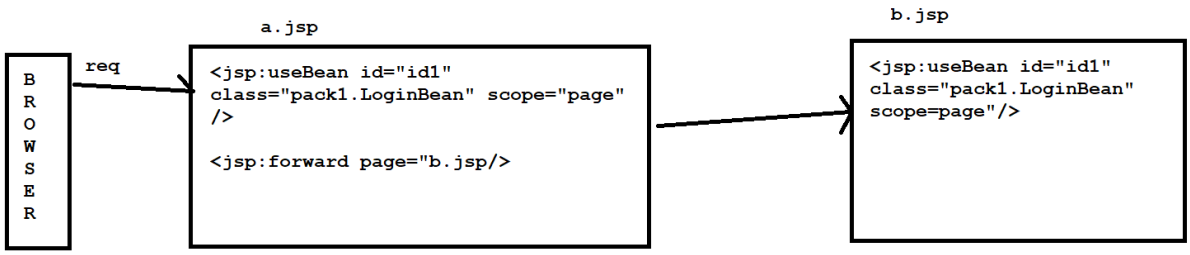
```
<jsp:getProperty name=”obj1” property=”uname”/>
```

- We cannot write property=* in <jsp:getProperty> action tag.

Bean Scopes:

pageScope:

- The bean object stored in page scope is not shareable with another pages.
- Jsp container creates separate page map for each jsp page.
- Page scope is sharable within the page scope so it is a local scope.

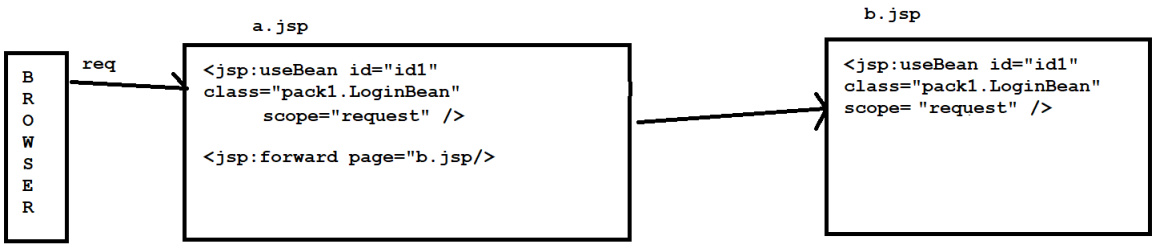


- In the above diagram, bean class object is created in a.jsp and also another object created in b.jsp because page scope of a.jsp is not shareable with b.jsp.

requestScope:

- A bean object stored in request scope is sharable with other pages participating in the same request.

- Jsp container creates a request scope map and the data stored in that map is sharable with other pages when a request is forwarded to the other pages.
- A jsp container creates one request scope map for each request separately.



- In ABove diagram,the Bean object is created in A.jsp and same object is shared in B.jsp also,because same request is forwarded to jsp.

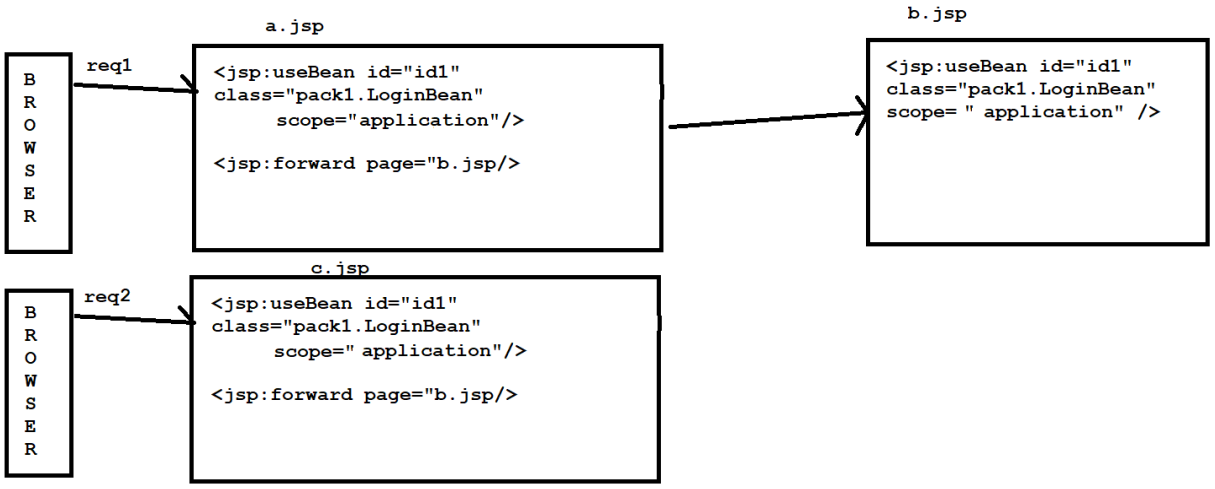
Session Scope:

- A bean object stored in session scope is shareable in multiple request of the same client.
- Jsp container creates a session scope map object for each client separately and the data stored in that map is sharable in multiple requests of same client.

- In above diagram,Ben Object is created in a.jsp and same is shared with b.jsp and c.jsp ,because same client has sent a second request to the c.jsp

Application Scope:

- A bean object stored in application scope is sharable to all the clients in all the pages across the entire application.
- Jsp container creates an application scope map object and makes its data shareable to all entire applications.



- In the above diagram, an bean object is created in a.jsp and the same object is shared in b.jsp and c.jsp also because application is global scope and its data is sharable in all jsp pages.

Implicit Objects in JSP:

- Implicit objects are also called implicit variables or implicit reference variable.
- There are 9 implicit objects in JSP.

Name	Type
request	HttpServletRequest
response	HttpServletResponse

session	HttpSession
out	JspWriter
exception	Throwble
config	ServletConfig
application	ServletContext
page	Object
pagecontext	PageContext

- In a jsp page, a session object is allowed to be used, if a session is enabled in the page.
- In jsp page session is enabled by default,because session attribute by default is true.
- Exception object is allowed,if a page is acting as an error.
- To act page error page , we need to set isErrorPage attribute as **true**.
- In Jsp technology, we call as **Application** and it is same as **ServletContext** object in servlet technology.
- Page object refers to the current object,page object same as this.
- We can find the following method in the _jspService method,after a jsp translated into servlet.

  Final java.lang.Object page=this;
- PageContext object is special implicit object in jsp and given for the following two purposes.
  a. To store remaining implicit objects in a single page context object.
  b. To store attributes in either page scope or request scope or session scope or application scope.

Attributes in JSP:

- A JSP can share it's data with other jsp pages through attributes.
- Attributes is a key-value pair of data .

- To share the data Attributes must be stored in an appropriate scope.
- In JSP we can store an attribute in one of the four scopes.
    1.Page
    2.Request
    3.Session
    4.Application
- An attribute can be stored in page scope only using a **PageContext** object.

    For Ex:

    1.pageContext.setAttribute(“K1”,”Java”);
    2.pageContext.setAttribute(“K1”,”Java”,1);
    3.pageContext.setAttribute(“K1”,”Java”
    ,PageContext.PAGE_SCOPE);

- In PageContext class has four Public Static Final Int variables are provided ,to indicate

Four Scopes in JSP

```
Public Static Final Int PAGE_SCOPE=1;
Public Static Final Int REQUEST_SCOPE=2;
Public Static Final Int SESSION_SCOPE=3;
Public Static Final Int APPLICATION_SCOPE=4;
```

- We can store an attribute in request scope,using request object an PageContext.

    For Ex:

```
request.setAttribute(“K2”,”JSP”);
    PageContext.setAttribute(“K2”,”JSP”,2);
    PageContext.setAttribute(“K2”,”JSP”,
            ,PageContext.REQUEST_SCOPE);
```

- TO store an attribute in Session Scope, using session an PageContext objects.

    For Ex:

```
session.setAttribute("Key","value");
pageConext.setAttribute("Key","Value",3);
pageConext.setAttribute("Key","Value",
                        pageContext.SESSION_SCOPE);
```

- To store an Attribute in application,use the application an PageContext object.

  For Ex:

```
application.setAttribute("Key","value");
pageContext.setAttribute("Key","value",4);
pageContext.setAttribute("Key","value",
                        PageContext.APPLICATION_SCOPE);
```

- We can read an a attribute from page scope,only by using **PagesContext** object

  For Ex:

```
Object o=PageContext.getAttribute("K1");
Object o=PageContext.getAttribute("K1",1);
Object o=PageContext.getAttribute("K1",
                        PageContext.PAGE_SCOPE);
```

- To read an attribute from request scope ,we can use either PageContext object an request object.

  Ex:

```
Object o=PageConetxt .getAtribute("k2",2);
Object o=request .getAttribute("k2");
```

- To read an attribute from session scope ,we can use either PageContext object an Session Object.

  Ex:

```
Object o=PageContext.getAttribute("K2",3);
Object o=request .getAttribute("k2");
```

- To read an attribute from application scope we can use either PagesContext object an application object.

  Ex:

```
Object o=PageContext.getAttribute("K2",4);
Object o=request .getAttribute("k2");
```

(Q) What is the different b/W getAttribute () and findAttribute() method of PageContext object?

- getAttribute() method will only search for the key under given scopo.if no exist then returns null.But findAttribute() method will search for the key in high level scope also .If not found in high level scopes also then reuters null.

  For Ex:

  Object o=pagesContext.getAttribute("K2",2);

  I

  It will search for the key only in request scope.

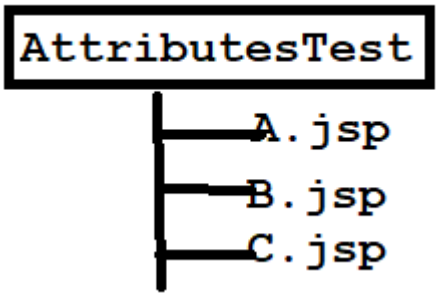
  Object o=pagesContext.findAttribute("K2",2);

  I

  It will search for the key only in request scope.if not found then Session scope then Application scope

Example:

- We are creating three JSP pages A.jsp, B.jsp, C.jsp. We forward the request from A.jsp to B.jsp and we send a separate request to C.jsp.
- We are finding which scope attribute are sharable between the two request and b/w the two client by using this example.



A.jsp

——---

- <! - - A.jsp - - >
  < %
  pageContent.setAttribute("K1", "A");

```
pageContent.setAttribute(“K2”, “B”,2);
pageContent.setAttribute(“K3”, “C”,3);
pageContent.setAttribute(“K4”, “D”,4);
%>
<jsp:forward page “ E.jsp”/>
```

B.jsp:

—----

```
<! - - b.jsp - - >
< %
    out.println(pageContent.findAttribute(“K1”);
    out.println(“<br>”);
    out.println(pageContent.findAttribute(“K2”);
    out.println(“<br>”);
    out.println(pageContent.findAttribute(“K3”);
    out.println(“<br>”);
    out.println(pageContent.findAttribute(“K4”);
    out.println(“<br>”);
%>
```

C.jsp:

—---

```
<! - - c.jsp - - >
< %
    out.println(pageContent.findAttribute(“K1”);
    out.println(“<br>”);
    out.println(pageContent.findAttribute(“K2”);
    out.println(“<br>”);
    out.println(pageContent.findAttribute(“K3”);
    out.println(“<br>”);
    out.println(pageContent.findAttribute(“K4”);
    out.println(“<br>”);
%>
```

Browser- 1:

- Req 1:http://Localhost:2024/AttributeTest/a.jsp

O/p : Null
B
C
D

- Req 2 : :http://Localhost:2024/AttributeTest/c.jsp

O/p: Null
Null
C
D

Browser- 2:

- Req : http://Localhost:2024/AttributeTest/c.jsp

O/p : Null
Null
Null
D

Custom tags in JSP:

- Custom tag is user defined tag.
- Already jsp technologies has provided pre-defined jsp tags as common to all the developers apart from predefined tags, If only new tags are required then developer can create own tags with login and developer can use then in multiple jsp pages of that project

  The own tags created are called Custom tags.
- To create a custom tag we need to follow the below steps:
  1) We need to define the logic of custom tags in java class. It is a tag handler.
  2) We need to configure one or more custom tags details in tld(tag lib descriptor) file.
  3) We need to configure the location and name if of each tld file with URI in web.xml
  4) Finally we need to import tag lib library URI in the js page.

Creating a tag handler:

- A tag handler is a java class and the jsp container will create an object of tag handler and calls its methods at runtime.

- We can create tag handler either by extending TagSupport or BodyTagSupport classes.
- TagSupport and BodyTagSupport are the classes given by JSP api in javax.servlet.jsp.Tagext.
- If we want to define logics for a custom tags starting and ending points then we need to extend a tag handler from TagSupport class.
- If we want to define logic for a custom tag startign point, for ending point,before enter into the body and after executing body then we need to extend a tag handler from BodyTagSupport.
- For Example, if we want to define logic for a custom tags startingPoint then we need to override doStartTag() method,similarly to define a logic for custom tag end point we need to override doEndTag() method.

```
public class HelloTagHandler extends agSupport
 {
      public int doStartTag() throws JSPException
       {
         // logic
       }
    public  int doEndTag() throws JSPExeption
       {
          // logic
       }
}
```

- doStartTag() method can return one of the following two variable values.

1. SKIP_BODY
2. EVAL_BODY_INCLUDE

- If a custom tag contains a body then we must return EVAL_BODY_INCLUDE.if THere is no body the return SKIP_BODY.
- doEndTag() method returns one of following two variables
    1. SKIP_PAGE
    2. EVAL_PAGE
- After the end of a custom tag if a remaining JSP page exists the must return EVAL_PAGE. If Not Exist then must return SKIP_PAGE.

creating a tld file:

- A tld file look like an xml file and we configure one or more custom tags in that file.
- If an application, we can divide custom tags into multiple groups, and we can configure each group of tag in a tld file.

one.tld:

```
<taglib>
        <tlib-version>1.0</tlib-version>
    <jsp-version>2.1</jsp-version>
    <short-name>Our first custom tag</short-name>

    <tag>
      <name>hello</name>
      <tag-class>pack1.HelloTagHandler</tag-class>
      <body-content>empty</body-content>
    </tag>
</taglib>
```

Configuring tld file:

- We need to configure location of tld file in web.xml

- With location, we also need to configure URI.
- In jsp container reads location of tld file from web.xml and then it will store the information of tags configured in tld under one unique name called URI.
- Each tld file created in web apllication must be configure in web.xml with <taglib> element.

web.xml

```
<web-app>
    <jsp-config>
        <taglib>
          <taglib-uri>http://super28.com/hello</taglib-uri>
          <taglib-location>/WEB-INF/one.tld</taglib-location>
        </taglib>
    </jsp-config>
</web-app>
```

Importing URI in JSP:

- If we want to write custom tags in jsp page then we need to import URI configured in web.xml
- Jsp technology has provided taglib directive for importing URI
- Each custom tag can be written in jsp page in xml syntax format,so a prefix is required for custom tags.

```
<%@ taglib uri="http://super28.com/hello" prefix="super" %>
```

Custom tags are divided into four types

1. Empty tag

```
<super:hello/>
```

2. With attribute

```
<super:hello user=”abcd”/>
```

3. With body

```
<super:hello>
    // logic
</super:hello>
```

4. With nested tag

```
<super:hello>
    <super:hii>
      // logic
    </super:hii>
</super:hello>
```

Adding attributes to custom tag:

If we want to define an attribute to a custom tag then there are two changes needed.

1. In tag handler ,create a variable with the same as attribute name..

```
private String course;
// setters and getters
```

2.Open one.tld file and add the following attribute tag after body-content.

```
<attribute>
    <name>course</name>
    <required>true</required>
    <rtexprvalue>true</rtexprvalue>
</attribute>
```

- The <required> value is true the attribute is mandatory for the tag,if it is false then attribute is optional.
- if <rtexprvalue> is true then runtime value are allowed to the attribute are allowed to the attribute,if it is false then only static value is allowed .

If jsp file define custom tag like the following.

```
<super:hello course="Java"/>
```

JSP Expression Language:

When working with JSP scripting elements, the JSP developers in industry are identified with the following problems.

1. When we write more no.of scripting elements in the jsp page then they are killing readability of the JSP page.

2. Scripting elements make debugging of JSP pages difficult.
3. With More number of scripting elements, java code also increased in a JSP page.

As a solution for the above problem sun microsystem responded with **Expression Language.**

- Expression Language is an internal part of JSP and sun microsystem called it as language because they given a new syntax to write the statement,operator and keywords.
- A statement of expression language starts with ${expression}
- We can use expression language statements as an alternate for scripting elements so we can reduce the java code within a jsp page.

Scriptlet:

```
<%
   Object o = request.getAttribute("key");
     String value=(String)o;
        out.println(value);
%>
```

EL statement:

```
${requestScope.key}
```

The above EL statement is equal to the scriptlet.

EL operators:

1. Arithmetic : + - * %
2. Relational : < > <= >= = !=
3. Logical: $$ || !

EL Implicit Objects:

1. pageScope
2. requestScope
3. sessionScope

4. applicationScope
5. applicationScope
6. Param
7. paramValues
8. Cookie
9. pageContext

- PageContext is a common implicit object in JSP and Expression.
- In Expression language,we use pageContext to call implicit Object of JSP in EL statement.
- We cannot call java methods and we cannot use java variables in EL statements.

EX1: ${pageScope.getAttribute("key") } —> wrong
Ex2: <% int number=100 %>
${number} ---------- wrong

- We can use key of an attribute in an EL statement because an attribute key is not a variable.

```
<% pageContext.setAttribute("k1","java",4);  %>
${application.k1}
```

- Suppose we want to call implicit object requests of JSP in EL statements then we can use pageContext objects like the following.
  ${pageContext.request.method}
- Suppose if we want to use an implicit session of JSP in EL statement then we can use a pageContext object like the following.

${pageContext.session.id}

- Param is an implicit object of EL used to read the value of the Request parameter.
- Suppose ${param.uname} statement reads the value of a request parameter uname.This EL statement is equals to the request.getParameter.

- ❖ paramValues is an implicit object used to read the multiple values of parameter.
- ❖ Suppose ${paramvalues.hobbies} reads the multiple values of a request parameters hobby,it is same as request.getParametersValues(“hobby”);
- ❖ Jsp container will process EL statements,based on isELIgnored attribute of page directive.
- ❖ The Default value of isELIgnored is false ,means Expression language is enabled. If we want to disable expression language statements we need to set isELIgnored=”true”.

Q) What is Difference between ${pageScope.k1} and ${k1} ?

- ➔ ${pageScope.K1} statement will only search under pagescope,if not found then it print a whitespace on browser.
- ➔ ${k1} statement will search for key from pageScope to ApplicationScope,it not found the its print a whitespace on browser

Note: in Expression language , null will be converted to white space.