Adv. Java means DURGA SIR..

ADV.JAVA

With

SCWCD / OCWCD

JSP Material

1. The Java Server pages (JSP) Technology model

DURGA M.Tech

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students
for last 14 years across INDIA

JSP's 2.1v

1) The Java Server pages (JSP) Technology model
2) Building JSP Pages Using Standard Actions
3) Building JSP Pages Using the Expression Language(EL)
4) Building JSP Pages using Tag Libraries(JSTL)
5) Building a Custom Tag Library

JSP Technology Model

1) JSP API and life cycle
   - Translation Phase
   - JSP-API
   - javax.servlet.jsp.JspPage(I)
   - javax.servlet.jsp.HttpJspPage(I)
   - Internal implementation of HttpJspBase class ?
   - LifeCycle of JSP
   - PreCompilation of JSP

2) JSP ELEMENTS :
   - Template Text
   - JSP Directives :
     1) page directive
        - import
        - session
        - isELIgnored
        - isThreadSafe
        - contentType
        - language
        - buffer
        - autoFlush
        - extends
        - info
        - isErrorPage
        - errorPage
        - pageEncoding
          - Summary of the page directive attributes
          - Is Servlet is ThreadSafe ?
          - Is Jsp is ThreadSafe by default ?
     2) include directive

        Difference between include directive and include action :

3) taglib directive

3) Scripting Elements
- ❖ Traditional Scripting elements
  - Scriptlets
  - Declarations
  - Expressions
  - Comments
    - Jsp comments
    - Html comments
    - Java comments
    - VISIBILITY of comments
    - Comparision of Scripting Elements
- ❖ Modern Scripting elements
  - EL
  - JSTL

4) JSP Actions

5) JSP Implicit Objects/elements :
- ❖ request
- ❖ response
- ❖ config
- ❖ application
- ❖ session
- ❖ out :
  - How to write response in Jsp ?
  - What is the difference between PrintWriter and JspWriter ?
  - What is the difference between out.write(100) and out.print(100) ?
- ❖ page
- ❖ pageContext
  - To get all other Jsp implicit objects :
  - To perform RequestDispatcher Mechanism :
  - To perform Attribute Management in any scope (Jsp scopes)
- ❖ exception
  - Declarative Approach
  - Programmatic Approach
  - Which approach is best approach to configure error pages ?

6) JSP Scopes
- request scope
- session scope
- application scope (or) context scope
- page scope

JSP API and life cycle

Describes the purpose and event sequence of JSP page life cycle.

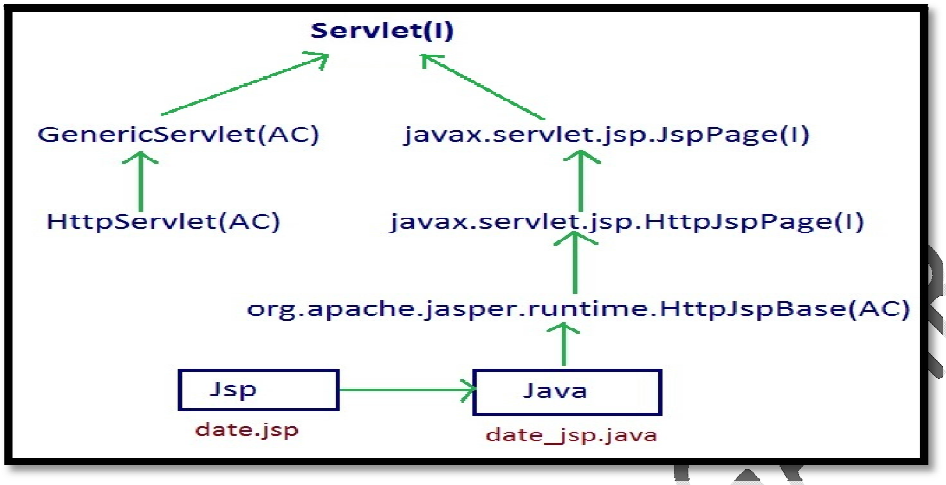
1. Translation phase
2. Jsp page compilation
3. Load the class
4. Create an Instance (Instantiation)
5. Call jspInit( )
6. Call _jspService( )
7. Call jspDestroy( )

Translation Phase :

1. The process of translating Jsp page (.jsp file) into corresponding Servlet (.java file) is called translation phase.
2. This can be done by Jsp container(engine) , In tomcat this engine is called "JASPER"
3. The generated .java file is available in the following location(Tomcat6.0\work\catalina\contextrootnamejsp1\org\apache\jsp)
   In weblogic : user-projects\domain\mydomain\myserver\.....

JSP-API :

1. Every class which is generated for the jsp , must implementsjavax.servlet.jsp.JspPage(I) Or javax.servlet.jsp.HttpJspPage either directly or indirectly
2. Tomcat people provided a base class org.apache.jasper.runtime.HttpJspBase for implementing the above 2 interfaces. Hence any Servlet which is generated by tomcat is extending this HttpJspBase class.



HttpJspBase class name will be valid from vendor to vendor (server to server)

javax.servlet.jsp.JspPage(I) :

This interface defines the following 2 methods

1. **jspInit( )**
2. **jspDestroy( )**

jspInit( ) :

public void jspInit( )

1. **This method will be executed only once to perform initialization activities , whenever we are sending first request to the JSP**
2. **Web-container always calls init(ServletConfig sc) presents in HttpJspBase class , which intern calls jspInit( )**
3. **If we have any initialization activities we have to override jspInit( ) in our Jsp**

```
public class HttpJspBase --- {
      public final void init(ServletConfig config)throws ServletException {
            jspInit();
      }
}
```

Various possibilities of overriding jspInit( ) in our JSP

date.jsp

```
<%!
      public void jspInit() {
          sysem.out.println("jsp Init");
```

```
        //our own initialization
    }
%>
The server time is : <%=new java.util.Date()%>
```

This method will be executed at the time of first request. We can't override init(ServletConfig sc) in jsp, because it is declared as final in HttpJspBase class

jspDestroy( )

public void jspDestroy()

1. This method will be executed only once to perform cleanup activities just before taking jsp from out of service.
   public void jspDestroy( )
2. Web-container always calls destroy( ) available in HttpJspBase which intern calls jspDestroy()

```
class HttpJspBase {
      final  destroy() {
            jspDestroy() ;
      }
      jspDestroy() {}
}
```

If we have any cleanUp activity we should override jspDestroy( ) & we can't override destroy( ) , because it is declared as final in HttpJspBase class.

javax.servlet.jsp.HttpJspPage(I) :

1. It is the child interface of JspPage(I),it contains only one method i.e., _jspservice( )
   public void _jspService(HttpServletRequest req , HttpServletResponse res) throws ServletException, IOException
2. This method will be executed for every request.
3. Web-container always calls service(HSR,HSR) of HttpJspBase class, which intern calls our _jspService(HSR,HSR)
4. _jspService( ) should be generated automatically by the web-container at translation phase and we can't override.
5. If we write _jspService( ) in the jsp explicitly then generated Servlet class contains two _jspService( ) , which causes compile time error.(with in the same class 2 methods with same signature is not possible)
6. In our jsp we can't override service( ) , because these are declared as final in base class.

Internal implementation of HttpJspBase class ?

```
public  abstract class HttpJspBase extends HttpServlet implements HttpJspPage {
```

```
    public final void init(ServletConfig config) throws ServletException {
        super.init(config);
        jspInit();
    }
    public final void destroy() {
        jspDestroy();
    }
    public void jspInit() {}
    public void jspDestroy() {}
    public final void service(HttpServletRequest req, HttpServletResponse res) throws
                                                                    ServletException,IOException {
        _ jspService(req, res);
    }
    public abstract void _jspService(HttpServletRequest req, HttpServletResponse res) throws
                                                                    ServletException,IOException;
            public String getServletInfo() {
                    return "Jsp information";
            }
}
```

What is the signature of Underscore in _jspService( ) ?

We can't write this method explicitly and it should be generated automatically by the web-container.

In our Jsp , which of the following methods we can write explicitly ?

1. **init(ServletConfig sc) //X-- final so wrong**
2. **jspInit( ) //right**
3. **destroy() //wrong**
4. **jspDestroy() //right**
5. **service() //wrong**
6. **_jspService() //wrong**

LifeCycle of JSP:

1. **jspInit()**
2. **_jspService(-,-)**
3. **jspDestroy()**

These methods are called life cycle methods of a Jsp.

.jsp ----------> .java ----------> .class ----------> load .class file ----------> instantiation ----------> jspInit() ---------> _jspService(-,-) ----------->jspDestroy()

1. **jspInit() and jspDestroy() methods will be executed only once, But _jspService() will be executed for every request.**

2. If any problem occurs at the time of translation Or at the time of compilation we will get 500 status code response.
3. JspPage will be participated into translated in the following cases :
    1. At the time of 1st request
    2. When compared with earlier request , Source code of the jsp got modified . For this web-container will use ARAXIS tool to compare time stamps of .jsp and .class file

date.jsp (date_jsp.java)

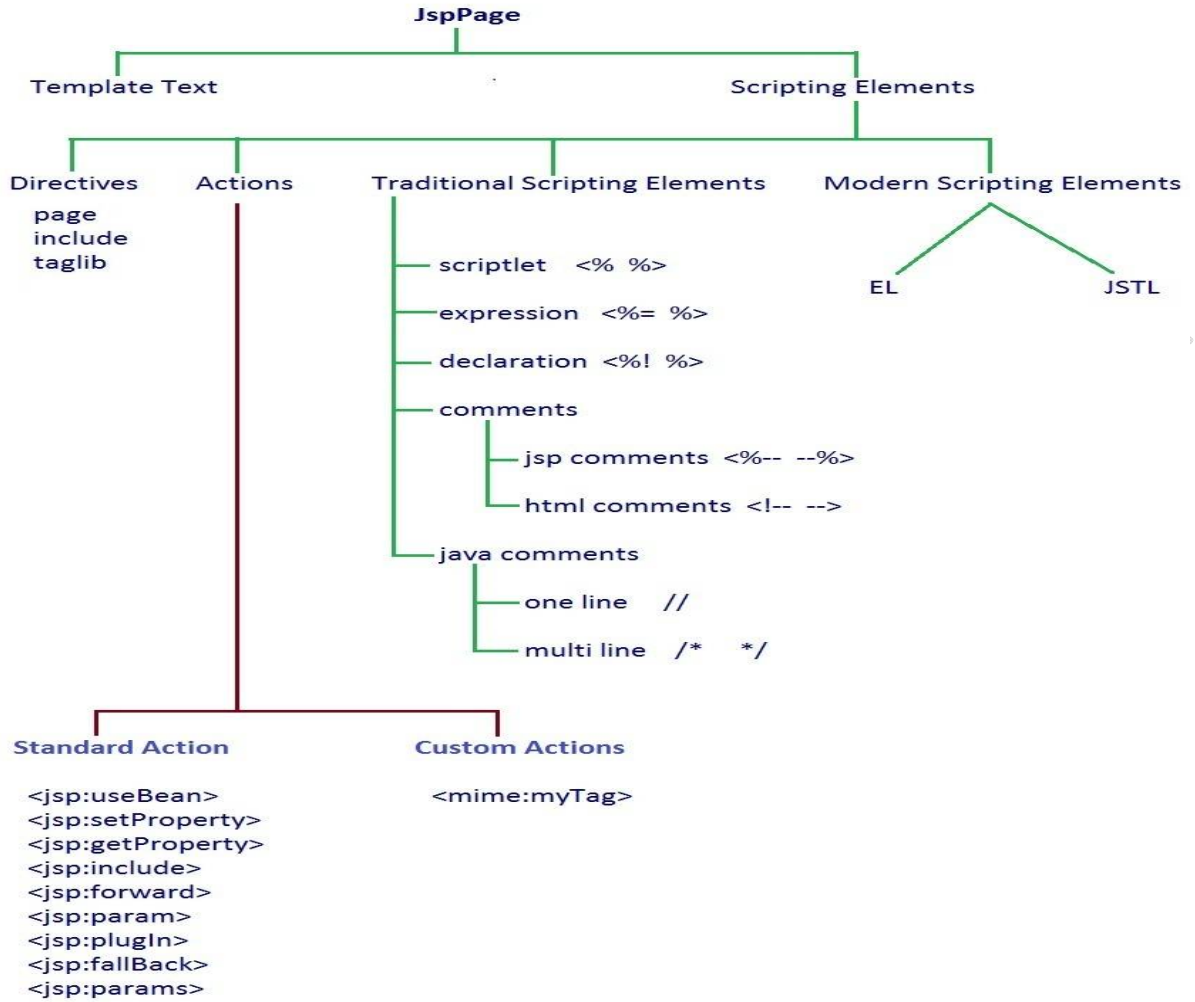
```
<%!
        static {
                System.out.println("class is loading");
        }
        public date_jsp() {
                System.out.println("The generated java class object is created");
        }
        public void jspInit() {
                System.out.println("jspInit() will be called");
        }
        public void jspDestroy() {
                System.out.println("jspDestroy() will be called");
        }
%>
The server time is : <%= new java.util.Date() %>
```

PreCompilation of JSP :

1. We can initiate translation phase explicitly by using pre-compilation process(upto jspInit()).
2. Jsp specification provides a special request parameter "jsp-precompile" to perform precompilation . We can use this parameter to hit the jsp for compilation without processing any request.
3. We can invoke pre-compilation as follows : http://localhost:8080/jsp1/date.jsp?jsp_precompile=true [upto jspInit( ) will be executed]
4. The main advantages of pre-compilation are
    1. All requests will be processed with uniform response time
    2. We can capture translation time errors and compilation time errors before processing first request.

While translation and compilation any problem or exception occurs then we will get http status code 500 response.

JSP ELEMENTS :



Objective : **Identify , describe and write Jsp code for the following elements**

1. **Template Text**
2. **Scripting elements**
3. **Standard and Custom Actions**
4. **Expression Language Elements**

Template Text :

1. **It consists of html and xml tags and raw data.**
2. **For the template text no translation is required , And it will become argument to out.write( ) present in _jspService( ) of generated servlet.**

date.jsp

```
The server time is :
<%= new java.util.Date() %>
```

date_jsp.java

```
public final class date_jsp extends HttpJspBase {
    public void _jspService(-,-)-- {
        out.write("The server time is:");
        out.print(new java.util.Date());
    }
}
```

The Template text will become argument to write() where as Expression value will become argument to print() , what is the reason.

write( ) can take only character data as argument and template text is always character data. Hence template text will become argument to out.write( ).
Expression value can be any data type , hence we can't use write( ) , compulsory we should go for print( ) which can take any type of argument.

JSP Directives :

A directive is a translation time instruction to the jsp engine by using directives.
we can specify whether current jsp error page or not, whether the current jsp participating into session traffic or not.

Syntax: <%@directive-name attribute-name=attribute-value1,.... %>

There are 3 types of directives

1) Page directive :
   - It can be used to specify overall properties of jsp page to the jsp engine
   - <%@ page isErrorPage="true" %>
   - This directive specifies that the current jsp is error page and hence jsp engine makes exception implicit object available to the jsp page.

2) Include directive :
   We can use this directive to include the content of some other page into current page at translation time(static include)
   <%@ include file="header.jsp" %>

3) taglib directive :
   We can use this directive to make custom tag functionality available to the jsp
   <%@ taglib prefix="mime" uri="www.jobs4times.com" %>

JSP Actions :

Actions are commands to Jsp engine , They direct jsp engine to perform certain task at execution time(Runtime).

The following are various standard actions available in jsp

1) < jsp:useBean >
2) < jsp:getProperty >
3) < jsp:setProperty >
4) < jsp:include >
5) < jsp:forward >
6) < jsp:param >
7) < jsp:plugIn >
8) < jsp:fallBack >
9) < jsp:params >

Ex: <jsp: include page="header.jsp" />
It includes the response of header.jsp in the current page response at runtime.(Dynamic include).

Scripting Elements :

There are 2 types of Scripting elements

1. Traditional Scripting elements
    1. Expressions
    2. Scriptlets
    3. Declarative
    4. Comments
2. Modern Scripting elements
    1. EL
    2. JSTL

Expressions :(<%= %>)

- Expression can be used to print java expression to the jsp.
  Example : <%= 2+3 %> (OR) out.print(2+3);

- Expression will become argument to out.print( ) present in _jspService( )
- Inside jsp expression , expression value should not ends with semicolon otherwise we will get http status 500 error.
  Ex : <%= newDate( ); %> (OR) out.print(new Date( ) ;); // 500 status code

- Inside expressions we can take method calls also but void return type method calls are not allowed.
  <%= math.random( ) %> //valid
  <%=(new AL( )).clear( ) %> // invalid
  because void return type methods are not allowed
- Inside expression space is not allowed between % and = , otherwise it will be treated as scriptlet which causes compile time error .
  <% =20 %>
  _jspService() {
  =20 ; // invalid statememt
  }

Declarations are not allowed in expressions i.e., declaring variables,methods etc.,
<%= String s="durga" %> OR

out.println(String s="durga"); //invalid

Which of the following Expressions are valid ?

1. <%= 2*3 %> //valid
2. <%= 2<3 %> //valid
3. <%= math.sqrt(4) %> //valid
4. <%= 2*3 ; %> //invalid
5. <%= new Student( ) %> // sop(new Student( )); //valid

Scriptlet : (<% %>)

1. We can use scriptlet to place java code in the jsp

```
<% any java code %>
// this java code in _jspService()
```

2. Scriptlet code will be placed directly inside _jspService( ) of generated servlet.
3. Every java statement present inside scriptlet should ends with semicolon(;).

Write Jsp to print hit count :

```
<%
      int count = 0;
      count++;
   //every request count initialized to 0 and print 1
     out.println(count);
%>
```

This Jsp is invalid and generates '1' as response every time, here count variable is local.

```
<%!
      int  count = 0 ;
%>

<%
     count++;
     out.println(count);
%>

<%
    /*
         any java code;
         it will be placed in generated servlet _jspService()
   */
%>

<%
      System.out.println("hello");
      // valid
%>

<%
       public int x=20; // invalid, x is not final
       out.println(x*2);
%>

<%
      final int x = 20; // valid, x is final
      out.println(x*2);
%>

<%
       public int m1() {
             int x = 10;
             int y = 20; //invalid
             return x*y;
       }
%>
Result is :<%= m1() %>
```

What is the difference between the following ?

<% int count=0 ; %>	<%! int count=0 ; %>

It is the local variable present inside _jspService()	It is the instance variable present in generated Servlet , out side of _jspService()

Note : It's never recommended to write scriptlets in the jsp.

Declarations : (<%! %>)

1. We can use declaration tag to declare instance variables , instance methods, static variables , static methods , instance and static blocks , constructors, inner classes etc.,

```
<%!
      // any java declarations
%>
```

2. These declarations will be placed directly in the generated Servlet but outside of _jspService( )

```
<%!
      int x = 10 ;
      static int y = 20;
      public void m1() {
            System.out.println("allowed");
      }
%>
```

3. Every statement present in declaration tag should compulsory ends with semicolon.

```
<%!
      public  void m1( ) {
            out.println("hello");
       }
%>
<%
      m1( ) ;
%>
```

CE: out cannot be resolved

```
class  {
      public void m1( ) {
            out.println("Hello");
       }
       _jspService(- - )- - {
            JspWriter out ;
            ----------
       }
}
```

out is local variable to _jspService( ) we can't access outside .

4. All implicit objects available in Jsp's are local variables inside _jspService( )
5. Declaration tag code will be placed outside of _jspService() , Hence implicit objects we can't use in declaration tags. Otherwise we will get Compile time error.
6. But we can use implicit objects inside scriptlets and expressions because their code will be placed inside _jspService( )

Comments :

There are 3 types of comments are allowed in Jsp

1. Jsp comments
   <%-- Jsp comments --%>
2. Html comments
   <!-- html/xml comments -->
3. Java comments
   1. Single line
   2. Multi line

Jsp Comments :

<%-- Jsp comment --%>

1. Also known as hidden comments , because these are not visible in the remaining phases of jsp execution (like .java, .class, and response).
2. This comments are highly recommended to use in the Jsp

Html Comments :

<!-- Html/XML comments -->

1. Also known as template text comments
2. These are visible to the end-user as the part of generated response source code, hence these are not recommended to use with in the Jsp.

Java Comments :

```
<%
    // single line  java comment
    /* multi line java comment */
%>
```

1. Also known as scripting Comments
2. These comments are visible in the generated Servlet source code(.java) but in all other phases these are not visible.

VISIBILITY :

Comments	Syntax	in JSP	in generated servlet source code(.java)	in end-user response
JSP	<%-- --%>	visible	not visible	not visible
Html	<!-- -->	visible	visible	visible

Java	// /* ..... */	visible	visible	not visible

Translator can perform only one level translation , hence among declaration,scriptlet,expressions and comments we can't take another i.e., nesting of scripting elements are not allowed otherwise we will get compiletime error.

```
<%!
 <%-- Jsp comments --%> // invalid
      int a=10;
%>
<%
    <% out.println("hello"); %>//invalid
%>
<%
     out.print(<%= 10+2+3 %>);//invalid
%>
<%--
    <% out.print("It is valid because i'm in comments"); %>
--%>
```

Comparision of Scripting Elements :

Elements	Syntax	is ends with simicolon	code will be _jspService()
Scriptlet	<% %>	Yes	Yes
Expression	<%= %>	No	Yes
Declaration	<%! %>	Yes	No
Jsp Comments	<%-- --%>	not applicable	not applicable

Directives :(<%@ %>)

Write a jsp code that uses the following directives are

1. page
2. include
3. taglib

Directives are translation time instruction to the jsp engine, by using these directives we can specify whether Jsp is an error page or not, whether Jsp is participating or not and etc.,

Syntax:
<%@ directive-name attribute-name=attribute-value1,..... %>
attribute-values should encode with singlecode(' ') or doublecode(" ")

page :

This page directive specifies overall properties of Jsp page to Jsp engine,
we can use this directive no.of times and any where.
Syntax:
<%@ page attribute-name=attribute-value %>

The following are list of possible attributes of the page directive (13):

1. import
2. session
3. isELIgnored
4. isThreadSafe
5. contentType
6. language
7. buffer
8. autoFlush
9. extends
10. info
11. isErrorPage
12. errorPage
13. pageEncoding

import :

We can use this import attribute to import classes and interfaces in our Jsp, this is exactly similar to core java import statement.

date.jsp (with import attribute)

```
<%@ page  import="java.util.Date" %>
The Server Time is : <%= new Date() %>
```

date.jsp (without import attribute)

```
The server time is: <%= new java.util.Date() %>
```

To import multiple packages the following various possibilities :

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

or

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

or

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

With in the same Jsp we are not allowed to take any attribute except import attribute of page directive multiple times with different values but we can take same attributemultiple times with same values for other attributes.

Q : Which of the following are valid ?

```
<%@ page  import="java.util.*" import="java.io.*" %>
    //valid
<%@ page  session="true"  session="false" %>
   //invalid
<%@ page isELIgnored="true" isELIgnored="false" %>
<%@ page   session="true"   session="true" %>
   //valid
<%@ page  language=java %>
 //invalid
```

The following packages are not required to import bydefault available in every Jsp.

1. java.lang.*;
2. javax.servlet.*;
3. javax.servlet.http.*;
4. javax.servlet.jsp.*;

session :

- We can use session attribute to make session object unavailable to the jsp by default session object is available to every jsp page.
- If we don't want to session object we have to declare page directive as follows

```
<%@ page session="false" %>
```

- In this case session object is not available in the jsp, in the rest of the jsp when ever we are trying to use this session object that time we will get an error like session can't be resolved.

```
<%@ page session="false" %>
<% out.println(session.getId()); %>
   //session can't be resolved
%>
```

- If we are not declaring session attribute explicitly (or) declaring session attribute with true value , then the equalent generated servlet code as follows
-

```
HttpSession session=null;
 session=pageContext.getSession();
```

If we take session attribute with false value then the generated servlet above 2 lines of code won't be available.

- The allowed values for session attribute are True/true/TRUE/TruE or False/false/FALSE/FalsE.
- In this case case is not important, consider only content.

- If we are taking other than true or false , then we will ger translation time error.

```
<%@ page session="ashok" %> //invalid
```

Which of the folowing make session object is available to Jsp ?

```
<%@ page session="true" %> //valid
<%@ page session="false" %> //invalid
<%@ page session=true %> //invalid
<%@ page contentType="text/html" %> //valid
```

isELIgnored

Expression Language introduced in jsp 1.2v
The main objective of EL is to eliminate java code from jsp

```
<%= request.getParameter("user") %>
```
Or
```
${param.user}
```

we can use isELIgnored attribute to enable or disable in our Jsp.
isELIgnored="true"

EL syntax just treated as template text without any processing

isELIgnored="false"
EL syntax will be evaluated and print its value.

```
<%@ page isELIgnored="true" %>
The result is : ${2+3}
```
output : The result is : ${2+3}

```
<%@ page isELIgnored="false" %>
 The result is : ${2+3}
```
output : The result is : 5

The default value of Jsp 1.2v is "true".
But from Jsp 2.0v onwards default value is "false".

isThreadSafe

We can use this attribute to provide Thread Safety to the Jsp
isThreadSafe="true"
It means the jsp is already thread safe and it is not required to implement SingleThreadModel interface by the generated sevrlet

isThreadSafe="false"

The current Jsp is not Thread Safe hence to provide thread safety generated servlet should implement SingleThreadModel interface, In this case the jsp can process only one request at a time.

The default value for isThreadSafe value is "true".

Which of the following to provide Thread Safety to the Jsp ?

- <%@ page isThreadSafe="true" %> // not provide
- <%@ page isThreadSafe="false" %> // provide
- <%@ page isThreadSafe="true" isThreadSafe="false" %> // not provide
- No special arrangement is required because jsp is Thread Safe by default.

contentType

- We have to use this attribute to specify content type of the response(mime type)
- The defaut value of the contentType is "text/html".

If we want to send JSON object as response to the client ,what is the content type required in the Jsp ?

<%@ page contentType="application/json" %>

language

- Defines the language used in scriptlets, expressions, declarations, write now the only possible value is Java.
- The defaut value of the language is "java".

Ex: <%@ page lanjuage="java" %>

buffer

Defines how buffering is handled by the implicit out object (reference to JspWriter)

autoFlush

Defines whether the buffered output is flushed automatically , the dafault value is "true".

extends

Defines the super class of the class this jsp will become
<%@ page extends="com. ....." %>//compiler error

info

Defines a string that gets put in to the translated page just show that we can get it using the generated servlet inherited getServletInfo()) .

isErrorPage

It is defined in exception implicit object.

errorPage

It is defined in exception implicit object.

pageEncoding

Defines the character encoding for the jsp's the default is "ISO-8859-1"

Summary of the page directive attributes :

Attribute name	Purpose	default value
import	To import classes and interfaces of package	There is no default value, but default packages are java.lang.*; javax.servlet.*; javax.servlet.http.*; javax.servlet.jsp.*;
session	To make session object unavailable to Jsp	true
contentType	To specify mime type of the response	text/html
isELIgnored	To enable or disable Expression Language in our Jsp	false
isThreadSafe	To provide Thread Safety to the Jsp	true
language	If can be used to specify the scripting language which is used in Jsp	java
pageEncoding	Defines character encoding for the Jsp	ISO-8859-1
outoFlush	Defines whether buffered output is flush automatically	true

Is Servlet is ThreadSafe ?

No, by default Sevlet is not ThreadSafe and we van provide Thread Safety by implementing SingleThreadModel interface.

Is Jsp is ThreadSafe by default ?

No, Jsp is not ThreadSafe by default and we provide Thread Safety by taking page directive.
<%@ page isThreadSafe="false" %>

Include Mechanism :

- several Jsp's required for common functionality then it's recommended to write that common functionality seperately in every jsp.
- We have to write that common code into seperate file and we have to include that file where ever it is required.

The main advantages in this approach is

1. code re-usuability
2. improves maintanability
3. Enhancements will be come very easy

We can achieve this include mechanism either by using include directive or include action.

include directive :

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

This inclusion will happen at translation time hence this inclusion is also called static include or translation time inclusion

include action :

Syntax : <jsp: include page="footer.jsp" flush="true" />

This inclusion will happen at request processing time and hence it is considered asdynamic include or run time inclusion

header.jsp

```
Master in java certification
```

footer.jsp

```
learning subject is not enough,
utilization is very important
```

main.jsp

```
<%@ include file="header.jsp" %>
welcome to learners , it is enough
<jsp:include page="footer.jsp" flush="true" />
```

Difference between include directive and include action :

include directive	include action
<%@ include file="header.jsp" %> It contains only one attribute and it is mandatory	<jsp: include page="footer.jsp" flush="true" /> It contains 2 attributes i.e., page,flush. page is mandatory,flush is optional
This inclusion will happen at translation time hence it's considered as static include.	This inclusion will happen at request processing time and hence it is considered as dynamic include.
For both including and inluded Jsp's a single servet will be generated hence code sharing between the components is possible.	For including and included Jsp's seperate servlets will be generated, hence code sharing between components is not possible.
Relatively performance is high	Relatively performance is low
There is no guarantee for the inclusion of latest version of included Jsp, it is a vendor dependent.	Always latest version of included page will be included.
If the target resource won't change frequently then it is recommended to use static include.	The target resource will change frequently then recommended to use dynamic include.

- To include header.jsp at translation time the required code is
  <%@ include file="header.jsp" %>
- To include header.jsp at request processing time the required code is
  <jsp: include page="header.jsp" />

1. <%@ include page="footer.jsp" %> //invalid
2. <%@ include file="footer.jsp" flush="true" %> //invalid
3. <%@ include file="footer.jsp" />//invalid
4. <%@ include file="footer.jsp" %> //valid
5. <jsp:include file="footer.jsp" flush="true" />//invalid
6. <jsp:include page="footer.jsp" flush="true" %>//invalid
7. <jsp:include page="footer.jsp" flush="true" />//valid

- page attribute applicable for include action but not for include directive where as the, file attribute applicable for include directive but not for include action.
- flush attribute applicable only for include action but not for include directive.

Example :

header.jsp

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

main.jsp

```
<%@ include file="header.jsp" %>
The result of x is : <%= x*100 %> //1000
```

main.jsp

```
<jsp:include page="header.jsp" />
The result of x : <%= x*100 %> //x can't be resolved
```

taglib directive :

syntax :

```
<%@ taglib prefix="mime" uri="www.jobs4times.com" %>
```

- We can use Taglib directive to make custom tag functionality available to the jsp
- taglib directive contains 2 attributes
  1. prefix
  2. uri
- prefix can be used with in the Jsp, uri can be used to specify location of TLD file.
- For every custom tag in the Jsp seperate taglib directive is required.

JSP Implicit Objects :

- Objective: For the given scinario write a jsp code by using appropriate jsp implicit objects.
- When compared with Servlet programming developing jsp's is very easy. Because the required mandatory stuff automatically available in every jsp.
- Implicit objects also one such area , implicit objects also bydefault available .

The following is the list of all implicit objects :

Implicit Objects	Type
request	javax.servlet.http.HttpServletRequest(I)
response	javax.servlet.http.HttpServletResponse(I)
config	javax.servlet.ServletConfig(I)
application	javax.servlet.ServletContext(I)
session	javax.servlet.http.HttpSession(I)
out	javax.servlet.jsp.JspWriter(AC)
page	java.lang.object(CC)
pageContext	javax.servlet.jsp.PageContext(AC)
exception	java.lang.Throwable(CC)

request & response implicit objects:

- request and response implicit objects are available to the jsp are arguments to _jspService()

- All the methods present in HttpServletRequest and HttpServletResponse can be applicable on this request and response implicit objects.

index.html

```
<form action="test.jsp">
      Enter UserName : <input type="text" name="uname" /> <br>
      Enter Course : <input type="text" name="course" /> <br>
      Enter Place : <input type="text" name="place"  /> <br>
      <input type="submit" value="submit">
</form>
```

test.jsp

```
<%@page import="java.util.*" %>
   The request method:<%= request.getMethod() %>
   The User Name : <%=request.getParameter("uname")  %>
<%
     Enumeration e=request.getParameterNames();
     while(e.hasMoreElements() ) {
 %>
 <%= request.getParameter((String)e.nextElement())  %>
<% } %>
<%
     response.setContentType("text/xml");
     response.setHeader("ashok","SCWCD");
 %>
```

application : (javax.servlet.ServletContext)

This implicit object of type javax.servlet.ServletContext , which provides environment of the web-application, what ever the methods available in ServletContext those methods are available application implicit object also.

Ex:

```
<%
     RequestDispatcher rd = application.getRequestDispatcher("/header.jsp");//absolute path
     rd.include(request,response); //OR
     rd.forward(request,response);
 %>
```

application.jsp

```
<%
     java.util.Enumeration e=application.getInitParameterNames();
     while(e.hasMoreElements()) {
%>
<%= application.getInitParameter((String)e.nextElement()) %>
<% } %>
```

web.xml

```
<web-app>
    <display-name>MyJsp</display-name>
    <context-param>
        <param-name>uname</param-name>
        <param-value>ashok</param-value>
    </context-param>
    ------------
</web-app>
```

config

This implicit object is of the type javax.servlet.ServletConfig what ever the methods present in ServletConfig interface, we can apply all these methods on config implicit object also.
The following are the methods available in ServletConfig interface

1. public String getServletName()
2. public String getInitParameter(String pname)
3. public Enumeration getInitParameterNames())
   http://localhost:2018/app/config.jsp
4. public ServletContext getServletContext())
   http://localhost:2018/app/test

config.jsp

```
The init parameter is :
  <%= getServletConfig.getInitParameter("uname") %>
The servlet parameter is :
      <%= config.getServletParameter("course") %>
The servlet name is :<%= config.getServletName() %>
```

web.xml

```
<web-app>
    <servlet>
        <servlet-name>configDemo</servlet-name>
        <jsp-file>/config.jsp</jsp-file>
        <init-param>
            <param-name>uname</param-name>
            <param-value>ashok</param-value>
        </init-param>
     </servlet>
     <servlet-mapping>
        <servlet-name>configDemo</servlet-name>
        <url-pattern>/test</url-pattern>
        <url-pattern>/test1</url-pattern>
        <url-pattern>/test2</url-pattern>
    </servlet-mapping>
</web-app>
```

If we are placing jsp with in the context root in the time, we can access that jsp the following 2 ways

1. By using it's name
2. By using it's url-pattern

```
http://localhost:2018/jspApp/config.jsp
Answer : null
       null
       jsp
       (the default logical name of the jsp's is jsp)
http://localhost:2018/jspApp/test
Answer : ashok
       test
       configDemo
```

To result servlet level configuration for the Jsp's compulsory we should access the jsp by using it's url-pattern.

page

page implicit object always pointing to current servlet object in the generated servlet it is declared as follows

```
public class .....
public void _jspService(-,-)--{
    Object page=this;
}
```

Which of the following are valid ?

1. <%= page.getServletInfo() %> //invalid
2. <%= ((Servlet)page).getServletInfo() %> //valid
3. <%= ((HttpServlet)page).getServletInfo() %> //valid
4. <%= this.getServletInfo() %> //valid
5. <%= getServletInfo() %> //valid

page implicit object is declared as java.lang.Object type, hence we are allowed to call only the methods present in the Object class but not Servlet specific methods.
Hence this implicit object is most rarely used implicit object.

session

session implicit object is by default available to every jsp and it is of the type of HttpSession.

```
<%@page session="true" %>
The session timeout is : <%= session.getMaxInactiveInterval() %>
The session id is : <%= session.getId() %>
The session new is : <%= session.isNew() %>
```

We make a session object unavailable to the jsp

```
<%@page session="false" %>
```

```
<%= session.getMaxInactiveInterval() %>
 //session can't be resolved
```

Which of the following make session object available to the Jsp ?

1. <%@page session=true %> //invalid
2. <%@page session="false" %> //invalid
3. <%@page session="true" session="false" %> //invalid
4. <%@page session="true" %> //valid
5. <%@page contentType="text/xml" %> //valid

session.jsp

```
<%@page session="true" %>
<% session.setAttribute("ashok","SCWCD"); %>
```

date.jsp

```
<%@page session="true" %>
<%= session.getId() %>
<%= session.getAttribute("ashok") %>
```

http://localhost:2018/scwdcApp/session.jsp
http://localhost:2018/scwdcApp/date.jsp

same browser : (date.jsp)
output : same sessionid, SCWCD
different browser : (date.jsp)
output : different sessionid, null

out : javax.servlet.jsp.JspWriter(AC)

- This implicit object of type javax.servlet.jsp.JspWriter, This object is specially designed for Jsp's to write character data to the response object.
- The main difference between PrintWriter and jspWriter in the case of PrintWriter there is no buffering concept is available . hence whenever we are writing any data by using PrintWriter it will be return directly into the response object.
- But in the case of JspWriter buffer concept is available, hence whenever we are writing by using JspWriter it will be written to the buffer and at the end total buffer data will be added to the response object.
- Except this buffer difference , there is no other difference between JspWriter and PrintWriter.
  JspWriter=PrintWriter+buffer

out.jsp

```
<%@page autoFlush="true" %>
<%
      java.io.PrintWriter pw=response.getWriter();
      pw.println("Good Morning Ashok");
      pw.println("Good Morning Akshay");
```

```
        pw.println("Good Morning Arun");
        pw.println("Good Morning Sai");
%>

output:
Good Morning Ashok
Good Morning Sai
Good Morning Akshay
Good Morning Arun
```

Note:

1. autoFlush="true" A value of true indicates automatically buffer flushing , A value of false indicates and value of false throws an exception.
2. It causes the servlet throws an exception when the servlet output buffer is full.
3. In Jsp can't set the false value for autoFlush attribute , if the buffer is "none".
4. The default buffer size is 8kb and none is also valid for the buffer attribute , In this case whenever we are writing any data by using JspWriter that time it will be return directly to the response object.

out.jsp

```
<%@page autoFlush="true" buffer="none" %>
<%
        java.io.JspWriter jw=response.getWriter();
        jw.println("Good Morning Ashok");
        jw.println("Good Morning Akshay");
        jw.println("Good Morning Arun");
        jw.println("Good Morning Sai");
%>

output:
Good Morning Ashok
Good Morning Akshay
Good Morning Arun
Good Morning Sai
```

- With in the Jsp we can use either PrintWriter or JspWriter but not recommended to use both symultaniously.
- JspWriter is the child class of Writer, hence all methods available in writer are available by default to the JspWriter through Inheritence.
- In addition to these methods JspWriter contains print(), println() methods, add any type of data add to these methods.

```
Writer(AC)   write(int)
             write(char)
             write(String)
```

```
JspWriter(AC)   print()
                println()

                    //AC = abstract class
```

Example:

```
<%
      out.print("The PI value is :");
      out.println(3.14);
      out.print("This exactly ");
      out.println(true);

output:
The PI value is : 3.14
This exactly true
%>
```

How to write response in Jsp

By using out implicit object which is of the type Writer.

What is the difference between PrintWriter and JspWriter ?

Buffering
JspWriter = PrintWriter + buffer

What is the difference between out.write(100) and out.print(100) ?

- **In the case of write(100), the corresponding character "d" will be added to the response.**
- **In the case of print(100) , the int value 100 will be added to the response.**

Ex:

```
<%
      out.write(100); //d
      out.print(100); //100
%>
```

pageContext : (javax.servlet.jsp.PageContext)

- **The pageContext implicit object is a type of javax.servlet.jsp.pageContext(AC).**
- **It is an abstract class and vendor is responsible to provide implementation , PageContext is the child class of JspContext(jsp 2.0v)**
- **PageContext class is useful in ClassicTagModel.**

We can pageContext implicit object for the following 3 purposes

1. To get all other Jsp implicit objects.
2. To perform RequestDispatcher Mechanism
3. To perform Attribute Management in any scope (Jsp scopes)

How to get all other Jsp implicit objects ?

By using pageContext implicit object we can get any other jsp implicit objects, hence pageContext implicit object act as a single point of contact for the remaining implicit objects.

PageContext class defines the following methods to retrive remaining all other implicit objects.

```
request     --> getRequest()
response    --> getResponse()
config      --> getServletConfig()
application --> getServletContext()
session     --> getSession()
page        --> getPage()
out         --> getOut()
exception   --> getException()
```

- These methods are not useful with in the Jsp because all jsp implicit objects already available to every Jsp,
- These methods are useful outside of Jsp mostly in custom tags(Tag Handler Classes).

To perform RequestDispatcher Mechanism :

We can use pageContext implicit object to perform RequestDispatcher activity also, for this purpose pageContext class defines the following methods

```
public void forward(String targetPath);
public void include(String targetPath);
```

Note : All the rules of ServletRequestDispatcher will be applicable for pageContext implicit object based on RequestDispatcher mechanism.

first.jsp

```
This is RD mechanism using pageContext implicit object
<%
pageContext.forward("second.jsp");
System.out.println("control come back");
%>
```

second.jsp

```
This is second jsp
<% java.util.Enumeration e=request.getAttributeNames(); %>
 ----------
```


To perform Attribute Management in any scope (Jsp scopes) :

We can use pageContext implicit object to perform attribute management in any scope (will be covered in Jsp scopes)

exception :

configuring error pages in Jsp.

We can configuring the following 2 approaches

1. Declarative Approach
2. Programmatic Approach

Declarative Approach :

We can configure error pages according to a particular exceptions or perticular error code in web.xml as follows

```
<web-app>
    <error-page>
          <exception-type>java.lang.ArithmeticExeption</exception-type>
         <location>/error.jsp</location>
    </error-page>
    <error-page>
        <error-code>404</error-code>
        <location>/error404.jsp</location>
    </error-page>
</web-app>
```

Note : The error pages configured in this approach is applicable entire web-application(not a perticular servlet Or not a perticular Jsp).

Programmatic Approach :

We can configure error pages for a particular Jsp by using error page attribute of page directive.

demo.jsp

```
<%@page errorPage="error.jsp" %>
<% out.println(10/0); %>
```

- In demo.jsp any problem or exception occurs then error.jsp is responsible to report that error.
- This way of configuring error pages for a particular jsp we can declare jsp as a error page by using isErrorPage attribute of page directive.
- In error pages only exception implicit object is available.

error.jsp

```
<%@page isErrorPage="true" %>
The raised Exception is : <%= exception %>
      //internally toString() calling
```

If Jsp is not configuring/declared as a error page and if we are trying to use exception implicit object then we will get compile time error saying that exception can't be resolved.

error.jsp

```
The raised Exception is : <%= exception %>
```

If we are trying to access error page directly without taking any exception then exception implicit object value refers null.

Ex: http://localhost:2018/jspApp/error.jsp

If we are configuring both declarative and programmatic approach on web-application, then programmatic approach will be effected.

Which approach is best approach to configure error pages ?

Declarative approach is best approach because we can customize error pages based on exception type and error code.

- jsp implicit objects are local variables of _jspService() in the generated servlet, hence we are allowed to use these implicit objects inside scriptlet and expression tag because the corresponding code will be place inside _jspService(), but we are not allowed to access declaration tag because it's code will be placed outside of _jspService().
- The most powerful implicit object is pageContext and rarely used implicit object is page.
- Among 9 implicit objects except session & exception , all remaining implicit objects are always available in every Jsp we can't make them unavailable.
- session implicit object is by default available in every jsp but we can make it's unavailable by using session attribute of page directive.

```
<%@page session="false" %>
```

- exception implicit object is by default not available to every jsp but we can make it's available by using isErrorPage attribute of page directive.

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

JSP Scopes

In our servlets we have follwing 3 scopes

1. request scope
2. session scope
3. application scope (or) context scope
4. page scope (In addition to these scopes we have page scope also in our jsp's)

request scope :

- In servlets this scope will be maintained by using ServletRequest object but in jsp's it is maintained by request implicit object.
- Information stored in request scope is available or applicable for all components which are processing the same request.
- The request scope will be started at the time of request object creation. i.e., just before starting service() and will be lost at the time of request object destruction, i.e., just after completion service().

We can perform attribute management in request scope by using the following methods of ServletRequest interface

1. public void setAttribute(String aname, Object avalue)
2. public Object getAttribute(String aname)
3. public void removeAttribute(String aname)
4. public Enumeration getAttributeNames()

session scope :

- session scope will be maintained in our servlet by using HttpSession object but in Jsp's maintained by session implicit object.
- Information stored in the session scope is available or applicable for all the components which are participating in the session scope.
- session scope will be started at the time of session object creation and will be lost once session will be expire either by invalidate() or session time out mechanism.

We can perform attribute management in session scope by using the following methods of HttpSession interface

1. public void setAttribute(String aname, Object avalue)

2. public Object getAttribute(String aname)
3. public void removeAttribute(String aname)
4. public Enumeration getAttributeNames()

application scope :

- We can maintain application scope in servlet by using ServletContext object but in Jsp's by using application implicit object.
- Information stored in the application scope is available or applicable for all the components of web-application.
- application scope will be started at the time of ServletContext object creation. i.e., at the time of application deployment or server startup, and ends at the time of ServletContext destruction i.e., application undeployment or server shutdown.

We can perform attribute management in application scope by using the following methods of ServletContext interface

1. public void setAttribute(String aname, Object avalue)
2. public Object getAttribute(String aname)
3. public void removeAttribute(String aname)
4. public Enumeration getAttributeNames()

page scope :

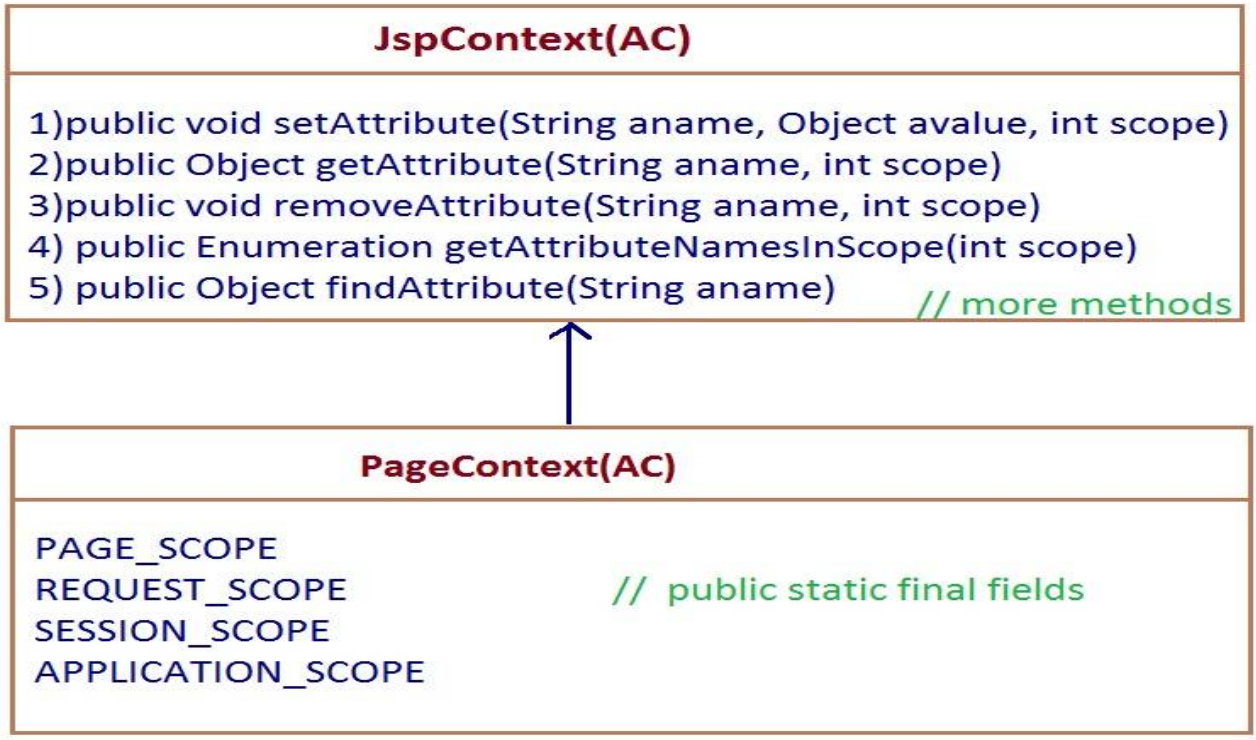
- This scope is applicable only for Jsp's but not for servlets, This scope will be maintained by pageContext implicit object (but not page implicit object).
- Information stored in the page scope is by default available with in the translation unit only.
- page scope is most commonly used scope in custom tag share information between TagHandler classes.

We can perform attribute management in page scope by using the following methods of PageContext class

1. public void setAttribute(String aname, Object avalue)
2. public Object getAttribute(String aname)
3. public void removeAttribute(String aname)

We can perform attribute management in any scope by using the following methods of pageContext implicit object :

We can use pageContext implicit object to perform attribute management in any scope for this purpose PageContext class following these methods



public void setAttribute(String aname, Object value, int scope)

The allowed values for the scope is :	**pageContext.PAGE_SCOPE or 1** **pageContext.REQUEST_SCOPE or 2** **pageContext.SESSION_SCOPE or 3** **pageContext.APPLICATION_SCOPE or 4**

If we are passing other than these values we will get Runtime exception saying java.lang.IllegalArgumentException(invalid scope)

public Object findAttribute(String aname)

First this method searches in page scope for the required attribute if it is available it return the corresponding value, if it is not available then it will search in request scope followed by session and application scope(order is important).

page > request > session > application

suppose that attribute is not available even in application then this method returns null.

demo.jsp

```
<%
    pageContext.setAttribute("ashok","scwcd",2);
    pageContext.setAttribute("ashok","scjp",4);
    pageContext.forward("header.jsp");
    pageContext.setAttribute("akshay","scjp");
%>
```

- In this case , if we are not mension in any scope by default scope in Jsp is page scope.
- The information is available in current jsp but not outside of the Jsp.

header.jsp

```
<%
    out.println(pageContext.getAttribute("akshay"));
    out.println(pageContext.findAttribute("ashok"));
    out.println(pageContext.findAttribute("abc"));
%>
```

output:

```
null
scwcd
null
```

Difference between getAttribute(-) and findAttribute(-) ?

- getAttribute(-) method it searches by default in page scope if we are not mension any scope.
- findAttribute(-) method it will search all scopes for the required attribute.

Which of the following are valid to store attribute in page scope ?

1. page.setAttribute("ashok","scjp"); //invalid
2. pageContext.setAttribute("ashok","scjp",2); //invalid
3. pageContext.setAttribute("ashok","scjp",pageContext.PAGE_SCOPE);//valid
4. pageContext.setAttribute("ashok","scwcd"); //valid
5. pageContext.setAttribute("ashok","scjp", pageContext.PAGE_CONTEXT_SCOPE); //invalid

```
<%!
    // java comments are allowed in declarations
%>

<%
   // java comments are allowed in scriptlet and expression also
  /* java comments are allowed in scriptlet and expression also */
%>
```

demo.jsp

```
<%
    pageContext.setAttribute("ashok","SCWCD",3);
    pageContext.setAttribute("ashok","SCJP",4);
    pageContext.forward("header.jsp");
%>
```

header.jsp

```
<%@page session="false" %>
<% out.println(pageContext.findAttribute("ashok")); %>
// session scope is not available so then go for application scope
```

scope	in Servlets	in Jsp's(using jsp implicit objects)
application/context	getServletContext. setAttribute("ashok","scwcd");	application. setAttribute("ashok","scwcd");
session	request.getSession(). setAttribute("ashok","scwcd");	session. setAttribute("ashok","scwcd");
request	request. setAttribute("ashok","scwcd");	request. setAttribute("ashok","scwcd");
page	Not Applicable	pageContext. setAttribute("ashok","scwcd");