Adv. Java means DURGA SIR..

ADV.JAVA

With

SCWCD / OCWCD

JSP Material

2. Building JSP Pages Using Standard Actions

DURGA M.Tech

(Sun certified & Realtime Expert)

Ex. IBM Employee

Trained Lakhs of Students
for last 14 years across INDIA

JSP Standard Actions

Agenda:

1) <jsp:useBean> tag
2) <jsp:setProperty> tag
3) <jsp:getProperty> tag
4) <jsp:include> tag
5) <jsp:forward> tag
6) <jsp:param> tag
7) <jsp:plug-in> tag
8) <jsp:params> tag
9) <jsp:fallback> tag
10) Summary of Standard actions
11) JSP Documentation

Introduction :

case 1: problem

```
public class JspDemoServlet extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException, IOException {
            String uname = request.getParameter("userName");
            request.setAttribute("UserName", uname);
            RequestDispatcher rd = request.getRequestDispatcher("view.jsp");
            rd.forward(request, response);
    }
}
```

view.jsp

```
Welcome to : <%= (String)request.getAttribute("UserName") %>
```

OR

case 2 :

```
public class JspDemoServlet extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                        throws ServletException, IOException {
            String uname = request.getParameter("userName");
            foo.Person p = new foo.Person();
            p.setName(uname);
            request.setAttribute("person",p);
            RequestDispatcher rd = request.getRequestDispatcher("view1.jsp");
            rd.forward(request, response);
      }
}
```

view1.jsp

```
Welcome to :
<%@page import="foo.Person"%>
Welcome to :
<%
        Person p=(Person)request.getAttribute("person");
        out.println(p.getName());
%>
```

Person.java

```
package foo;
public class Person {
        String name;
        public void setName(String name) {
              this.name = name;
         }
         public String getName() {
              return name;
         }
}
```

Solution : view1.jsp

```
<jsp:useBean id="person" class="foo.Person" scope="request" />
Welcome to : <jsp:getProperty property="name" name="person" />
```

Problem : demo.jsp

```
<%!
      public int squareIt(int n) {
              return n*n;
      }
%>
<h1>The square of 4 is : <%= squareIt(4) %></h1>
<h1>The square of 2 is : <%= squareIt(2) %></h1>
```

<u>This approach has several serious dis-advantages :</u>

- There is no clear separation of presentation logic and business logic. The person who is writing the jsp should have compulsory knowledge on both java & html which may not be possible always, this approach doesn't promote re-usability .
- It reduces readability.
- We can resolve these problem by separating entire business logic inside java bean.

javabean :

```
package foo;
public class Calculator {
      public int squareIt() {
          return n*n;
      }
}
```

view.jsp

```
<jsp:useBean id="calc" class="foo.Calculator" scope="session" />
The square of 4 is : <%= calc.squareIt(4) %>
```

This approach has several advantages :

- Separation of presentation and business logic (presentation logic available in Jsp and business logic available in java bean) , so that readability is improved.
- Java developer can concentration on business logic where as Html developer can concentration on presentation logic as both can work simultaneously so that we can reduce development time and increases productivity.
- It promotes re-usability of the code i.e., where ever this squareIt() functionality is used we can reuse the same bean with rewriting.
- We can purchase bean for file uploading and we can start uploading of the file with in minutes.

Note : It is never recommended to write scriptlets, expressions, declarations and business logic with in the Jsp.

java bean: java bean is a simple java class

To use useBean inside jsp the bean class has to follow the following rules

- Jsp engine is responsible to perform instantiation of Java bean for this always executes public no argument constructor otherwise <jsp:useBean> tag won't be work. (because it internally calls that setter method only)
- For every property bean class should contains public getter methods otherwise<jsp:getProperty> tag won't be work.(because it internally calls corresponding getter method only)
- It is highly recommended to keep bean class as part of some package.

Jsp Standard Actions :

<jsp:useBean> :

We can use this tag to make bean functionality available to the Jsp, There are 2 forms of <jsp:useBean>

without body :

```
<jsp:useBean id="c" class="CustomerBean" />
```

with body :

```
<jsp:useBean id="c" class="CustomerBean" >
   ----------------------
<jsp:useBean>
```

The main objective of body is to perform initialization for the newly created bean object, The bean object is already available then it won't be created new bean object it will reuse existing object only , in this case body won't be executed.

Attribute list of <jsp:useBean> :

1. id
2. class
3. type
4. scope
5. beanName

id :

- This id attribute represents name of the reference variable of the bean object.
- By means of this id attribute only we can access bean in rest of the Jsp's
- This attribute is a mandatory attribute.

```
<jsp:useBean id="c" class="CustomerBean" />
  // the appropriate equivalent java code
CustomerBean c = new CustomerBean();
```

class :

- This attribute represents fully qualified name of the Java bean class, for this class only jsp engine perform instantiation hence the value of class attribute should be concrete class.

- abstract class and interfaces are not allowed.
- The bean class compulsory should contain public no-arg constructor otherwise we will get instantiation problem.
- This attribute is optional but when ever we are not specify class attribute compulsory we should specify type attribute.

type :

- We can use type attribute to specify the type of reference variable.
- The value of type attribute can be concrete class or abstract class or interface.
- This attribute is optional but when ever we are not specify type attribute compulsory we should specify class attribute.

```
<jsp:useBean id="p" type="foo.Person" class="foo.Student" />
  // equivalent java code
foo.Person p = new foo.Student();
```

scope :

- This attribute specifies in which scope jsp engine has to search for the required bean object, in that scope if the bean object is not already available then Jsp engine creates a bean object and store into specified scope for the future purpose.
- The allowed values for the scope attributes are page, request, session, application.
- This attribute is optional and default scope is "page".

```
<jsp:useBean id="c" class="CustomerBean" scope="session" />
//equivalent java code
CustomerBean c=null;
syncronized(session) {
    c = (CustomerBean)pageContext.getAttribute("c", pageContext.SESSION_SCOPE);
    if(c == null) {
        c = new CustomBean();
        pageContext.setAttribute("c",c,pageConext.SESSION_SCOPE);
        ----------------
    }
}
```

If we specify only type attribute without class attribute then bean object should be already available with in the specified scope if it is not already available then we will get instantiation exception.

```
<jsp:useBean id="c" type="CustomerBean" scope="session" />
```

To use session scope compulsory session object should be available otherwise we will get translation time error.

```
<%@page session="false" %>
<jsp:useBean id="c" class="CustomerBean" scope="session" />
```

Exception: Illegal for useBean to use session scope when jsp page declare that it doesn't participate into the session.

```
package foo;
public class CalculatorBean {
        static int i = 0;
        public CalculatorBean() {
              System.out.println("calculator bean object created"+ (++i)+"times");
        }
        public int squareIt(int n) {
               return n*n;
        }
}
```

test.jsp

```
<jsp:useBean id="calc" class="foo.CalculatorBean" scope="session" />
The square of 4 is : <%= calc.squareIt(4) %>
The square of 5 is : <%= calc.squareIt(5) %>
<a href="beanTest.jsp">Click Here</a>
```

beanTest.jsp

```
<jsp:useBean id="calc" type="foo.CalculatorBean" scope="session" />
The square of 8 is : <%= calc.squareIt(8) %>
The square of 9 is : <%= calc.squareIt(9) %>
```

case 2 :

```
<jsp:useBean id="calc" type="foo.CalculatorBean" scope="session" >
          The square of 8 is : <%= calc.squareIt(8) %>
          The square of 9 is : <%= calc.squareIt(9) %>
</jsp:useBean>
```

case 3 :

```
<jsp:useBean id="c" type="foo.CalculatorBean" scope="session" >
        The square of 8 is : <%= calc.squareIt(8) %>
        The square of 9 is : <%= calc.squareIt(9) %>
</jsp:useBean>
```

exception: javax.servlet.ServletException:java.lang.InstantiationException

person.java

```
package foo;
public abstract class Person {
       public abstract String getMessage();
}
```

Student.java

```
package foo;
public  class Student extends Person {
        public  String getMessage() {
              return "Ashok";
        }
        public String getName() {
              return "Aggidi";
        }
}
```

test.jsp

```
<jsp:useBean id="p" type="foo.Person" class="foo.Student" scope="session" />
The person name is : <%= p.getMessage() %>
<a href="beanTest.jsp">Click Here</a>
```

beanTest.jsp

```
<jsp:useBean id="person" class="foo.Student" scope="session" />
The student name is : <%= person.getName() %>
```

beanName :

There may be a chance of using serialized bean object from local file system in that case we have to used beanName attribute.

Conclusions for <jsp:useBean> :

1. 'id' attribute is mandatory
2. 'scope' attribute is optional and default scope is page.
3. The attributes class, type, beanName can be used in the following combinations
    1. class
    2. type
    3. class and type
    4. type and beanName
4. beanName is always associated with type attribute but not class attribute.
5. If we are using class attribute whether we are using type attribute or not compulsory it should be concrete class and it should contains public no-arg constructor.
6. If we are using only type attribute without class attribute then the specified bean object compulsory should be available specified scope otherwise this tag won't create new bean object and raises instantiation exception.

Consider the following code

```
package pack1;
public class CustomerBean {
  ----------------
}
```

Assume that no CustomerBean object is already created then which of the following standard actions create a new bean object and store in request scope ?

1. <jsp:useBean id="c" class="pack1.CustomerBean" /> //invalid
2. <jsp:useBean id="c" type="pack1.CustomerBean" scope="request" />//invalid
3. <jsp:useBean name="c" class="pack1.CustomerBean" scope="request" />//invalid
4. <jsp:useBean id="c" class="pack1.CustomerBean" scope="request" /> //valid

<jsp:getProperty> :

This tag can be used to get the properties of bean object

```
<%
        CustomerBean c = new CustomerBean();
        out.println(c.getName());
%>

//equalant jsp

<jsp:useBean id="c" class="CustomerBean" />
<jsp:getProperty name="c" property="name" />
```

<jsp:getProperty> tag contains the following 2 attributes

1. name
2. property

name :

It represents name of the bean object reference variable from which the required property has to retrieve , it is exactly equal to id attribute of <jsp:useBean>.

property :

It represents the name of the java bean property whose value has to retrieve.

Note : both attributes are mandatory attributes.

```
package foo;
public class CustomerBean {
        private String name = "ashok";
        private String mail = "ashok@jobs4times.com";
        public String getName() {
            return name;
        }
        public String getMail() {
             return mail;
        }
}
```

index.jsp

```
<%@page import="foo.CustomerBean" %>
<%
        CustomerBean c=new CustomerBean();
        out.println("The customer name is :"+c.getName());
        out.println("The customer mail is :"+c.getMail());
 %>

 (OR)

<jsp:useBean id="c" class="foo.CustomerBean"/>

The Customer name is : <jsp:getProperty property="name" name="c"/>
The Customer mail is : <jsp:getProperty property="mail" name="c"/>
```

<jsp:setProperty> :

We can use this tag to set the properties of bean object , <jsp:setProperty> tag contains the following forms

form 1 :

```
<jsp:setProperty property="age" name="c" value="30" />
     (OR)
 c.setAge("30");
```

form 2 :

```
<jsp:setProperty property="age" name="c" param="age" />
     (OR)
 c.setAge(request.getParameter("age"));
```

Note : The param attribute to retrieve the request parameter value "age" and assign in to java bean property age.

form 3 :

```
<jsp:setProperty property="age" name="c"  />
     (OR)
 c.setAge(request.getParameter("age"));
```

Note: To get this type of advantage it is highly recommended to maintain bean property names same as form parameter names.

form 4 :

```
<jsp:setProperty property="*" name="c"  />
 // "*" specifies all the properties of java bean
```

Note : It iterates all request parameters and if any request parameter name match with java bean property name then assign request parameter value to the java bean property.

<jsp:setProperty> contains the following attributes

1. name
2. property
3. value
4. param

name :

It represents name of the reference variable of the bean object whose property has to set, this is exactly same as id attribute of <jsp:useBean> , It is mandatory attribute.

property :

It represents name of the java bean property which has to set, it is mandatory attribute.

value :

It specifies the value which has to set to the java bean property, it is an optional and should not come combination with param attribute.

param :

This attribute specifies name of the request parameter whose value has to set to the java bean property, it is an optional attribute and should not come combination with value attribute.

CustomerBean.java

```
package foo;
public class CustomerBean {
        private String name = null;
        private String mail = null;
        private String age = null;
        public String getName() {
              return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public String getMail() {
            return mail;
        }
        public void setMail(String mail) {
              this.mail = mail;
        }
```

```
        public String getAge() {
                return age;
        }
        public void setAge(String age) {
                this.age = age;
        }
}
```

index.jsp

```
<form action = "test.jsp" method="post">
        <h2>Enter Your Details :</h2>
        Enter Name : <input type="text" name="uname">
        Enter Email : <input type="text" name="mail">
        Enter Age : <input type="text" name="age">
        <input type="submit" value="submit">
</form>
```

test.jsp

```
<h1>Set Property Example : </h1>
<jsp:useBean id="c" class="foo.CustomerBean" />
<jsp:setProperty property="*" name="c"/>
Entered Name is :
<jsp:getProperty property="name" name="c" />
Entered Mail is :
<jsp:getProperty property="mail" name="c" />
Entered Age is :
<jsp:getProperty property="age" name="c" />
```

Note : All required type conversions(String to int) will be performed automatically by <jsp:setProperty>.

Case 1 :

test.jsp

```
<h1>Set Property Example : </h1>

<jsp:useBean id="c" class="foo.CustomerBean" />

<jsp:setProperty property="mail" name="c"
 value="<%= request.getParameter("mail") %>" /> //valid
<jsp:setProperty property="mail" name="c"
 value="<%= request.getParameter("age") %>" /> //invalid

Entered Name is :
<jsp:getProperty property="name" name="c" />
```

```
Entered Mail is :
<jsp:getProperty property="mail" name="c" />
Entered Age is :
<jsp:getProperty property="age" name="c" />
```

Note : Automatic String to primitive conversions doesn't work if we use scripting elements, it fails even if we use expression inside <jsp:setProperty> standard actions.

```
<jsp:useBean id="c" class="foo.CustomerBean" />
<jsp:setProperty property="*" name="c" /> //valid
<jsp:setProperty property="empId" name="c" param="empId" /> //valid
<jsp:setProperty property="empId" name="c"  /> //valid
<jsp:setProperty property="empId" name="c" value="1123" /> //valid
```

Note : If we are using scripting elements inside jsp standard actions automatic conversions doesn't work but it is possible scripting elements inside standard actions.

```
<jsp:useBean id="c" class="foo.CustomerBean" />
<jsp:setProperty property="empId" name="c"
 value="<%= request.getParameter("empId") %>" /> //invalid
```

Developing reusable web-components :

We can develop reusable web-components by using the following 3 standard actions

1. <jsp:include>
2. <jsp:forward>
3. <jsp:param>

<jsp:include> :

The response of include page will be included in current response at request processing time, hence it is a dynamic include.

This tag representation of pageContext.include("header.jsp");

This standard action contains the following 2 attributes

1. page
2. flush

- **page :** page attribute represents name of the included page and it is mandatory.
- **flush :** It specifies whether the response will be flushed before inclusion or not , it is an optional attribute and default value is false.

index.jsp

```
<jsp:include page="header.jsp" />
```

```
<h2>Welcome to Jobs4Times.com</h2>
<jsp:include page="footer.jsp" />
```

header.jsp

```
<h2> We are Master in Java Certification </h2>
```

footer.jsp

```
 <h4>copyright © www.jobs4times.com </h4>
```

Following 4 possible ways to implement include mechanism in JSP's :

1) By using include directive :

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

2) By using include action :

```
<jsp:include page="header.jsp" flush="true" />
 // by default flush is false
```

3) By using pageContext implicit object :

```
<% pageContext.include("header.jsp");  %>
```

4) By using RequestDispatcher :

```
<%
      RequestDispatcher rd=request.getRequestDispatcher("header.jsp");
      rd.include(request,response);
%>
```

<jsp:forward> :

If the first jsp is responsible to perform some preliminary processing activities and second jsp is responsible for providing complete response then we should go for forward mechanism.

Syntax :

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

It contains only one attribute, i.e., page , and it is a mandatory attribute.

first.jsp

```
<h2>This is first Jsp page </h2>
<jsp:forward page="second.jsp" />
```

```
<h2>This is after forwarding </h2>
```

second.jsp

```
<h2>This is second Jsp page </h2>
```

Following 3 possible ways to implement forward mechanism in JSP's :

1. By using forward action :

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

2. By using pageContext implicit object :

```
<% pageContext.forward("second.jsp");  %>
```

3. By using RequestDispatcher :

```
<%
      RequestDispatcher rd=request.getRequestDispatcher("header.jsp");
      rd.forward(request,response);
%>
```

<jsp:param> :

While forwarding or including if we want to send parameters to the target Jsp we can achieve this by using <jsp:param> tag
<jsp:param> tag defines the following 2 mandatory attributes

1. name : It represents name of the parameter.
2. value : It represents value of the Parameter.

Note : The parameters which are sending by using <jsp:param> tag are available as form parameters in target Jsp.

first.jsp

```
<jsp:include page="second.jsp">
 <jsp:param value="ashok" name="username"/>
 <jsp:param value="25" name="age"/>
</jsp:include>
```

second.jsp

```
<%@page isELIgnored="false" %>
<h2>Hi i'm getting the values from Jsp param</h2>
The UserName is : <%=request.getParameter("username") %>
The Age is : <%=request.getParameter("age") %>
```

Conclusions :

case 1 :

```
<jsp:forward page="http://localhost:2020/jspApp/second.jsp"/>  //invalid
<jsp:include page="http://localhost:2020/jspApp/second.jsp"/>  //invalid
<%@include file="http://localhost:2020/jspApp/second.jsp"%>  //invalid
```

The value of page and file attributes should be a relative path or absolute path but not server port, protocol and etc.,

case 2 :

```
<jsp:forward page="/test" /> //valid
<jsp:include page="/test" /> //valid
<%@include file="/test" %>  //invalid
```

In the case of forward and include actions page attribute can pointing to a servlet.
But in the case of include directive file attribute can't pointing to a servlet but it can pointing to a Jsp,html,xhtml,xml and etc.,

case 3 :

```
<jsp:forward page="/test?name=ashok" />   //valid
<jsp:forward page="second.jsp?name=ashok" /> //valid
<jsp:include page="second.jsp?name=ashok" /> //valid
<%@include file="second.jsp?name=ashok" %> //invalid
```

In the case of include and forward actions we are allowed to pass query string,
But in the case of include directive we are not allowed to pass query string.

<jsp:plugin> :

- Using <jsp:plugin> action provides the support for including java applet in a jsp page, If we have java applets that are part of your web-applications.
- While it's possible to have appropriate html code embedded in your Jsp page, the use of <jsp:plugin> allows the functionality to the browser is neutral.

There are the following 2 optional support tags those work with <jsp:plugin>. i.e., <jsp:params> and <jsp:fallback>

<jsp:params> :

We can pass any additional parameters to the target applet or bean.

<jsp:fallback> :

Which specifies any content that should be display to the browser, if the plugin is not started or because of some runtime issues.
A plugin specific message will be presented to the end user.

A translation time error will occurs if we use <jsp:params> , <jsp:fallback> any other context other than child of <jsp:plugin>.

Syntax :

```
<jsp:plugin code="classFileName .class extention"
  codebase="The directory of class file name" type="applet/bean" >
  {align="alignment"}
  {height=""}
  {width=""}

{ <jsp:params>
  { <jsp:param value="pvalue" name="pname"/> }
</jsp:params> }

<jsp:fallback>  Arbitrary test </jsp:fallback>
</jsp:plugin>

  {} ---->optional attributes
```

type :

The type of object plugin will execute you must specify either applet or bean, as this attribute there is no default value.
It is a mandatory attribute.
type="applet/bean"

code : (class="ClassFileName")

The name of the java class file that plugin will execute we must include .classextension, file name is relative to the directory named in codeBase attribute , it is also mandatory attribute.

codeBase : (codeBase="ClassFileDirectoryName")

This is a absolute path or relative path to the directory that contains applet code, it is also mandatory attribute.

index.jsp

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

plugIn.jsp

```
<h2>Welcome to Jsp Application</h2>
```

First Applet :

```
<jsp:plugin code="ClockApplet.class" codebase="applets" type="applet"  width="260"  height="150">
 <jsp:fallback>Plug in tag not supported ..... </jsp:fallback>
</jsp:plugin>

Second Applet :
<jsp:plugin code="edu.com.MessageApplet.class"
      codebase="applets" type="applet" width="500"  height="100">
 <jsp:params>
  <jsp:param value="Welcome to SCWCD" name="msg"/>
 </jsp:params>
 <jsp:fallback>No support to this Applet ..... </jsp:fallback>
</jsp:plugin>
```

ClockApplet.java

```
public class ClockApplet extends Applet implements Runnable {
       String time;
       public void init() {
            Thread t = new Thread(this);
            t.start();
      }

      public void Paint(Graphics g) {
             g.setColor(Color.lightGray);
             g.fill3DRect(40, 40, 150, 100, true);
             g.setFont(new Font("monotypeCorsiva",Font.ITALIC,35));
             g.setColor(Color.red);

             g.drawString(time, 60, 100);
}

public void run() {

       while(true) {
          time = String.valueOf (new SimpleDateFormat("hh:mm:ss").format(new Date()));
          repaint();
          try {
             Thread.sleep(1000);
          } catch (InterruptedException e) {
                    e.printStackTrace();
          }
     }//run
  }
}//class
```

MessageApplet.java

```
package edu.com;
```

```
public class MessageApplet extends Applet implements Runnable {
      String message;
      public void init() {
            message = getParameter("msg");//available in applet class
            if(message == null) {
                  message = "This is default message";
            }
}

public void Paint(Graphics g) {
       g.setColor(Color.BLUE);
       g.setFont(new Font("monotype Corsiva",Font.ITALIC,40));
        g.drawString(message, 20, 50);
}

public void run() {}

}
```

first.jsp

```
<h2>This is first Jsp Page</h2>
<%! String nextPage="second.jsp"; %>
<jsp:forward page="<%= nextPage %>" /> //valid
          // static or dynamic
<h2>This is after forwarding</h2>
```

second.jsp

```
<h2>This is second Jsp Page</h2>
```

Summary of Standard actions :

Standard action	Purpose	attributes
<jsp:useBean>	To make bean object available to the Jsp	id, class, type, scope, beanName
<jsp:getProperty>	To get and print properties of Java bean	name, property
<jsp:setProperty>	To set the properties of bean	name, property, value, param
<jsp:forward>	To forward a request from one Jsp to another Jsp	page
<jsp:include>	To include the response of some other Jsp into current Jsp at request processing time	page, flush
<jsp:param>	To send parameters to the target Jsp while forwarding or including	name, value
<jsp:plugin>	To embedded a java applet/bean into your Jsp	type, code, codeBase, {height,width,align}

<jsp:params>	To pass additional data or parameters to the target applet or java bean	no
<jsp:fallback>	A plugin specific message will be display to the end-user when a plugin is not started or due to some runtime issue	no

JSP Documentation :

They are 2 types syntaxes are possible to write a Jsp

1. Jsp Standard syntax
2. Xml based syntax

- Jsp Standard Syntax : If we are writing a Jsp by using Jsp standard syntax such type of Jsp's are called Jsp pages.
- Xml based syntax : If we are writing Jsp by using Xml based syntax such type of Jsp's are called Jsp document.

	Standard Syntax	XML based Syntax (all are case sensitive)
scriptlet :	<% %>	<jsp:script>
declararion :	<%! %>	<jsp:declaration>
expression :	<%= %>	<jsp:expression>

directives :

directives	Standard Syntax	XML based Syntax (all are case sensitive)
page	<%@page import="java.util.*" %>	<jsp:directive.page import="java.util.*" />

include	<%@include file="second.jsp" %>	<jsp:directive.include file="second.jsp" />
taglib	<%@taglib prefix="mime" uri="www.jobs4times.com" %>	For the taglib directive there is no equivalent syntax in xml but we can provide its purpose by using <jsp:root>

Note :

```
<jsp:root xmlns:mime="www.jobs4times.com" version="4.3" />
  //version is mandatory attribute
  //xmlns:mime is an optional attribute
   //and we can take any no.of xml name spaces
```

Standard actions :

There is no difference in standard actions representation between standard syntax and xml based syntax.

Ex :

	Standard Syntax	XML based Syntax
useBean	<jsp:useBean id="c" class="CustomerBean" />	// same

Comments :

Jsp specification doesn't provide any specific syntax for writing comments , hence we can use normal syntax

Jsp comment	XML comment
<%-- --%>	<!-- -->

Template text :

Jsp Standard syntax doesn't provide any specific way to write template text but Xml based syntax it provides a special tag <jsp:text>

Jsp standard syntax	XML based syntax
This is secons jsp	<jsp:text> This is second Jsp </jsp:text>

How the web-container identifies Jsp document ?

1. Save the jsp file with .jspx extension.
2. Enclose entire Jsp with in <jsp:text> tag.
3. Use <jsp-config> element in web.xml as follows

```
<web-app>
    <jsp-config>
        <jsp-property-group>
            <url-pattern>/jspx/*</url-pattern>
            <is-xml>true</is-xml>
        </jsp-property-group>
    </jsp-config>
</web-app>
```

Note : From servlet 2.4v onwards web-container can identify Jsp document automatically , no special arrangement is reuired.

Write a Jsp document to print hit count of the Jsp ?

index.jsp (servlet 2.5v)

```
<jsp:directive.page isELIgnored="false" />
<!--
    including header.jsp to current  jsp
-->
<jsp:directive.include file="header.jsp" />
<jsp:declaration> int count=0; </jsp:declaration>
<jsp:scriptlet>  count++; </jsp:scriptlet>
<jsp:text>The hit count of this page is : </jsp:text>
<jsp:expression> count </jsp:expression>
```

- It is never recommended to use both standard and Xml based syntax simultaneously.
- The main advantage of Xml based syntax is we can use Xml editors like Xml-spy for writing and debugging jsp's
- All xml based tags and attributes are case-sensitive and attributes enclosed either single single quote(') or double quote("").

Which of the following valid way of importing java.util.ArrayList in our Jsp ?

```
<jsp:page import="java.util.ArrayList" /> //invalid

<jsp:page.directive import="java.util.ArrayList" /> //invalid

<jsp:directive  page import="java.util.ArrayList" /> //invalid

<jsp:directive.page import="java.util.ArrayList" /> //valid
```

Person.java

```
package foo;

public class Person {
    private String name;
    private Dog dog;
```

```
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
        public Dog getDog() {
            return dog;
        }
        public void setDog(Dog dog) {
            this.dog = dog;
        }
}
```

Dog.java

```
package foo;
public class Dog {
        private String name;
        public String getName() {
            return name;
        }
        public void setName(String name) {
            this.name = name;
        }
}
```

JspDemoServlet.java

```
public class JspDemoServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws ServletException, IOException {
            foo.Person p = new Person();
            p.setName("ashok");

            foo.Dog d = new Dog();
            d.setName("spike");

            p.setDog(d);

            request.setAttribute("person", p);
            RequestDispatcher rd=request.getRequestDispatcher("view.jsp");
            rd.forward(request, response);
    }
}
```

view.jsp

```
The Person's Dog name is :
<%=((foo.Person)request.getAttribute("person")).getDog().getName()%>
```

It is never recommended to use Scripting elements in Jsp , hence the above solution is not proper approach

```
<jsp:useBean id="person" class="foo.Person" scope="request" />
The Person's Dog name is :
<jsp:getProperty property="dog" name="person" />

output:
The Person's Dog name is : foo.Dog@1234...
//If request scope is not specified the output is null
Solution : ${Person.dog.name}
```

The <Jsp:getProperty> tag lets you access only property of bean attributes, there is no capability for nested properties, where you want a property of property rather than property of attribute.

Limitations of Standard Actions :

Standard Actions can handle String and primitive properties , if we know the standard actions can deal with in a attribute of any type as long as all the attribute properties are String/primitives.
there is no capability for nested properties.