*** JSP – Directives ***

These directives provide directions and instructions to the container, telling it how to handle certain aspects of the JSP processing.

A JSP directive affects the overall structure of the servlet class. It usually has the following form –

<%@ directive attribute = "value" %>

Directives can have a number of attributes which you can list down as key-value pairs and separated by white space.

The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

There are three types of directive tag –

S.No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

JSP - The page Directive

The **page** directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of the page directive –

<%@ page attribute = "value" %>

You can write the XML equivalent of the above syntax as follows –

<jsp:directive.page attribute = "value" />

Attributes

Following table lists out the attributes associated with the page directive –

S.No.	Attribute & Purpose
1	**buffer** Specifies a buffering model for the output stream.
2	**autoFlush** Controls the behavior of the servlet output buffer.
3	**contentType** Defines the character encoding scheme.
4	**errorPage** Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
5	**isErrorPage** Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
6	**extends** Specifies a superclass that the generated servlet must extend.
7	**import** Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
8	**info** Defines a string that can be accessed with the servlet's **getServletInfo()** method.
9	**isThreadSafe**

	Defines the threading model for the generated servlet.
10	**language** Defines the programming language used in the JSP page.
11	**session** Specifies whether or not the JSP page participates in HTTP sessions
12	**isELIgnored** Specifies whether or not the EL expression within the JSP page will be ignored.
13	**isScriptingEnabled** Determines if the scripting elements are allowed for use.

The buffer Attribute

The **buffer** attribute specifies the buffering characteristics for the server output response object.

You may code a value of "**none**" to specify no buffering so that the servlet output is immediately directed to the response object or you may code a maximum buffer size in kilobytes, which directs the servlet to write to the buffer before writing to the response object.

To direct the servlet to write the output directly to the response output object, use the following –

```
<%@ page buffer = "none" %>
```

Use the following to direct the servlet to write the output to a buffer of size not less than 8 kilobytes –

```
<%@ page buffer = "8kb" %>
```

The autoFlush Attribute

The **autoFlush** attribute specifies whether buffered output should be flushed automatically when the buffer is filled, or whether an exception should be raised to indicate the buffer overflow.

A value of **true (default)** indicates automatic buffer flushing and a value of false throws an exception.

The following directive causes the servlet to throw an exception when the servlet's output buffer is full –

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

This directive causes the servlet to flush the output buffer when full –

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

Usually, the buffer and the autoFlush attributes are coded on a single page directive as follows –

```
<%@ page buffer = "16kb" autoflush = "true" %>
```

The contentType Attribute

The contentType attribute sets the character encoding for the JSP page and for the generated response page. The default content type is **text/html**, which is the standard content type for HTML pages.

If you want to write out XML from your JSP, use the following page directive –

```
<%@ page contentType = "text/xml" %>
```

The following statement directs the browser to render the generated page as HTML –

```
<%@ page contentType = "text/html" %>
```

The following directive sets the content type as a Microsoft Word document –

```
<%@ page contentType = "application/msword" %>
```

You can also specify the character encoding for the response. For example, if you wanted to specify that the resulting page that is returned to the browser uses **ISO Latin 1**, you can use the following page directive –

```
<%@ page contentType = "text/html:charset=ISO-8859-1" %>
```

The errorPage Attribute

The **errorPage** attribute tells the JSP engine which page to display if there is an error while the current page runs. The value of the errorPage attribute is a relative URL.

The following directive displays MyErrorPage.jsp when all uncaught exceptions are thrown –

```
<%@ page errorPage = "MyErrorPage.jsp" %>
```

The isErrorPage Attribute

The **isErrorPage** attribute indicates that the current JSP can be used as the error page for another JSP.

The value of isErrorPage is either true or false. The default value of the isErrorPage attribute is false.

For example, the **handleError.jsp** sets the isErrorPage option to true because it is supposed to handle errors –

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

The extends Attribute

The **extends** attribute specifies a superclass that the generated servlet must extend.

For example, the following directive directs the JSP translator to generate the servlet such that the servlet extends ***somePackage.SomeClass*** –

```
<%@ page extends = "somePackage.SomeClass" %>
```

The import Attribute

The **import** attribute serves the same function as, and behaves like, the Java import statement. The value for the import option is the name of the package you want to import.

To import **java.sql.***, use the following page directive –

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

To import multiple packages, you can specify them separated by comma as follows –

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

By default, a container automatically imports **java.lang.*, javax.servlet.*, javax.servlet.jsp.*,** and **javax.servlet.http.***.

The info Attribute

The **info** attribute lets you provide a description of the JSP. The following is a coding example –

```
<%@ page info = "This JSP Page Written By ZARA"  %>
```

The isThreadSafe Attribute

The **isThreadSafe** option marks a page as being thread-safe. By default, all JSPs are considered thread-safe. If you set the isThreadSafe option to false, the JSP engine makes sure that only one thread at a time is executing your JSP.

The following page directive sets the **isThreadSafe** option to false –

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

The language Attribute

The **language** attribute indicates the programming language used in scripting the JSP page.

For example, because you usually use Java as the scripting language, your language option looks like this –

```
<%@ page language = "java" %>
```

The session Attribute

The **session** attribute indicates whether or not the JSP page uses HTTP sessions. A value of true means that the JSP page has access to a builtin **session** object and a value of false means that the JSP page cannot access the builtin session object.

Following directive allows the JSP page to use any of the builtin object session methods such as **session.getCreationTime()** or **session.getLastAccessTime()** –

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

The isELIgnored Attribute

The isELIgnored attribute gives you the ability to disable the evaluation of Expression Language (EL) expressions which has been introduced in JSP 2.0.

The default value of the attribute is true, meaning that expressions, **${...}**, are evaluated as dictated by the JSP specification. If the attribute is set to false, then expressions are not evaluated but rather treated as static text.

Following directive sets an expression not to be evaluated –

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

The isScriptingEnabled Attribute

The **isScriptingEnabled** attribute determines if the scripting elements are allowed for use.

The **default value (true)** enables scriptlets, expressions, and declarations. If the attribute's value is set to false, a translation-time error will be raised if the JSP uses any scriptlets, expressions (non-EL), or declarations.

The attribute’s value can be set to false if you want to restrict the usage of scriptlets, expressions (non-EL), or declarations –

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

The include Directive

The **include** directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the ***include*** directives anywhere in your JSP page.

The general usage form of this directive is as follows –

```
<%@ include file = "relative url" >
```

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write the XML equivalent of the above syntax as follows –

```
<jsp:directive.include file = "relative url" />
```

Example:-

File name : Include_directive.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>JSP Page</title>
   </head>
   <body>

      <h1>Hello World!</h1>
      Today's date :
      <%@include file="next.jsp" %>
   </body>
</html>
```

File name : next.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8" import="java.util.Date"%>
<!DOCTYPE html>
<html>
   <head>
```

```
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <%
        Date d=new Date();
        out.println(d);
        %>
    </body>
</html>
```

The taglib Directive

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The **taglib** directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

The taglib directive follows the syntax given below –

```
<%@ taglib uri="uri" prefix = "prefixOfTag" >
```

Here, the **uri** attribute value resolves to a location the container understands and the **prefix** attribute informs a container what bits of markup are custom actions.

You can write the XML equivalent of the above syntax as follows –

```
<jsp:directive.taglib uri = "uri" prefix = "prefixOfTag" />
```

Example :-

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
   <head>
      <title> c:out Tag Example</title>
   </head>

   <body>
      <c:out value = "${2*2000}"></c:out>
```

```
  </body>
</html>
```

*** JSP - Action ***

These actions use constructs in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

There is only one syntax for the Action element, as it conforms to the XML standard –

```
<jsp:action_name attribute = "value" />
```

Action elements are basically predefined functions. The following table lists out the available JSP actions –

S.No.	Syntax & Purpose
1	**jsp:include** Includes a file at the time the page is requested.
2	**jsp:useBean** Finds or instantiates a JavaBean.
3	**jsp:setProperty** Sets the property of a JavaBean.
4	**jsp:getProperty** Inserts the property of a JavaBean into the output.
5	**jsp:forward** Forwards the requester to a new page.
6	**jsp:plugin** Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
7	**jsp:element** Defines XML elements dynamically.

8	**jsp:attribute** Defines dynamically-defined XML element's attribute.
9	**jsp:body** Defines dynamically-defined XML element's body.
10	**jsp:text** Used to write template text in JSP pages and documents.

Common Attributes

There are two attributes that are common to all Action elements: the **id** attribute and the **scope** attribute.

Id attribute

The id attribute uniquely identifies the Action element, and allows the action to be referenced inside the JSP page. If the Action creates an instance of an object, the id value can be used to reference it through the implicit object PageContext.

Scope attribute

This attribute identifies the lifecycle of the Action element. The id attribute and the scope attribute are directly related, as the scope attribute determines the lifespan of the object associated with the id. The scope attribute has four possible values: **(a) page, (b)request, (c)session**, and **(d) application**.

The <jsp:include> Action

This action lets you insert files into the page being generated. The syntax looks like this –

<jsp:include page = "relative URL" flush = "true" />

Unlike the **include** directive, which inserts the file at the time the JSP page is translated into a servlet, this action inserts the file at the time the page is requested.

Following table lists out the attributes associated with the include action –

S.No.	Attribute & Description
1	**page**

	The relative URL of the page to be included.
2	**flush** The boolean attribute determines whether the included resource has its buffer flushed before it is included.

Example

Let us define the following two files **(a)date.jsp** and **(b) main.jsp** as follows −

Following is the content of the **date.jsp** file −

```
<p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
```

Following is the content of the **main.jsp** file −

```
<html>
  <head>
    <title>The include Action Example</title>
  </head>

  <body>
    <center>
      <h2>The include action Example</h2>
      <jsp:include page = "date.jsp" flush = "true" />
    </center>
  </body>
</html>
```

Let us now keep all these files in the root directory and try to access **main.jsp**. You will receive the following output −

```
The include action Example


  Today's date: 12-Sep-2010 14:54:22
```

The <jsp:useBean> Action

The **useBean** action is quite versatile. It first searches for an existing object utilizing the id and scope variables. If an object is not found, it then tries to create the specified object.

The simplest way to load a bean is as follows –

```
<jsp:useBean id = "name" class = "package.class" />
```

Once a bean class is loaded, you can use **jsp:setProperty** and **jsp:getProperty** actions to modify and retrieve the bean properties.

Following table lists out the attributes associated with the useBean action –

S.No.	Attribute & Description
1	**class** Designates the full package name of the bean.
2	**type** Specifies the type of the variable that will refer to the object.
3	**beanName** Gives the name of the bean as specified by the instantiate () method of the java.beans.Beans class.

Let us now discuss the **jsp:setProperty** and the **jsp:getProperty** actions before giving a valid example related to these actions.

The <jsp:setProperty> Action

The **setProperty** action sets the properties of a Bean. The Bean must have been previously defined before this action. There are two basic ways to use the setProperty action –

You can use **jsp:setProperty** after, but outside of a **jsp:useBean** element, as given below –

```
<jsp:useBean id = "myName" ... />
...
<jsp:setProperty name = "myName" property = "someProperty" .../>
```

In this case, the **jsp:setProperty** is executed regardless of whether a new bean was instantiated or an existing bean was found.

A second context in which jsp:setProperty can appear is inside the body of a **jsp:useBean** element, as given below –

```
<jsp:useBean id = "myName" ... >
  ...
  <jsp:setProperty name = "myName" property = "someProperty" .../>
```

```
</jsp:useBean>
```

Here, the jsp:setProperty is executed only if a new object was instantiated, not if an existing one was found.

Following table lists out the attributes associated with the **setProperty** action –

S.No.	Attribute & Description
1	**name** Designates the bean the property of which will be set. The Bean must have been previously defined.
2	**property** Indicates the property you want to set. A value of "*" means that all request parameters whose names match bean property names will be passed to the appropriate setter methods.
3	**value** The value that is to be assigned to the given property. The the parameter's value is null, or the parameter does not exist, the setProperty action is ignored.
4	**param** The param attribute is the name of the request parameter whose value the property is to receive. You can't use both value and param, but it is permissible to use neither.

The <jsp:getProperty> Action

The **getProperty** action is used to retrieve the value of a given property and converts it to a string, and finally inserts it into the output.

The getProperty action has only two attributes, both of which are required. The syntax of the getProperty action is as follows –

```
<jsp:useBean id = "myName" ... />
...
<jsp:getProperty name = "myName" property = "someProperty" .../>
```

Following table lists out the required attributes associated with the **getProperty** action –

S.No.	Attribute & Description
1	**name** The name of the Bean that has a property to be retrieved. The Bean must have been previously defined.
2	**property** The property attribute is the name of the Bean property to be retrieved.

Example

Let us define a test bean that will further be used in our example −

```
/* File: TestBean.java */
package action;

public class TestBean {
  private String message = "No message specified";

  public String getMessage() {
    return(message);
  }
  public void setMessage(String message) {
    this.message = message;
  }
}
```

Compile the above code to the generated **TestBean.class** file and make sure that you copied the TestBean.class in **C:\apache-tomcat-7.0.2\webapps\WEB-INF\classes\action** folder and the **CLASSPATH** variable should also be set to this folder −

Now use the following code in **main.jsp** file. This loads the bean and sets/gets a simple String parameter −

```
<html>

  <head>
    <title>Using JavaBeans in JSP</title>
  </head>

  <body>
    <center>
```

```
      <h2>Using JavaBeans in JSP</h2>
      <jsp:useBean id = "test" class = "action.TestBean" />
      <jsp:setProperty name = "test"  property = "message"
        value = "Hello JSP..." />

      <p>Got message....</p>
      <jsp:getProperty name = "test" property = "message" />
    </center>
  </body>
</html>
```

Let us now try to access **main.jsp**, it would display the following result –

```
Using JavaBeans in JSP


Got message....
Hello JSP...
```

The <jsp:forward> Action

The **forward** action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet.

Following is the syntax of the **forward** action –

<jsp:forward page = "Relative URL" />

Following table lists out the required attributes associated with the forward action –

S.No.	Attribute & Description
1	**page** Should consist of a relative URL of another resource such as a static page, another JSP page, or a Java Servlet.

Example

Let us reuse the following two files **(a) date.jsp** and **(b) main.jsp** as follows –

Following is the content of the **date.jsp** file –

```
<p>Today's date: <%= (new java.util.Date()).toLocaleString()%></p>
```

Following is the content of the **main.jsp** file –

```
<html>
  <head>
    <title>The include Action Example</title>
  </head>

  <body>
    <center>
      <h2>The include action Example</h2>
      <jsp:forward page = "date.jsp" />
    </center>
  </body>
</html>
```

Let us now keep all these files in the root directory and try to access **main.jsp**. This would display result something like as below.

Here it discarded the content from the main page and displayed the content from forwarded page only.

```
Today's date: 12-Sep-2010 14:54:22
```

The <jsp:plugin> Action

The **plugin** action is used to insert Java components into a JSP page. It determines the type of browser and inserts the **<object>** or **<embed>** tags as needed.

If the needed plugin is not present, it downloads the plugin and then executes the Java component. The Java component can be either an Applet or a JavaBean.

The plugin action has several attributes that correspond to common HTML tags used to format Java components. The **<param>** element can also be used to send parameters to the Applet or Bean.

Following is the typical syntax of using the plugin action –

```
<jsp:plugin type = "applet" codebase = "dirname" code = "MyApplet.class"
  width = "60" height = "80">
  <jsp:param name = "fontcolor" value = "red" />
  <jsp:param name = "background" value = "black" />

  <jsp:fallback>
    Unable to initialize Java Plugin
  </jsp:fallback>

</jsp:plugin>
```

You can try this action using some applet if you are interested. A new element, the **<fallback>** element, can be used to specify an error string to be sent to the user in case the component fails.

The <jsp:element> Action

The <jsp:attribute> Action

The <jsp:body> Action

The **<jsp:element>**, **<jsp:attribute>** and **<jsp:body>** actions are used to define XML elements dynamically. The word dynamically is important, because it means that the XML elements can be generated at request time rather than statically at compile time.

Following is a simple example to define XML elements dynamically –

```
<%@page language = "java" contentType = "text/html"%>
<html xmlns = "http://www.w3c.org/1999/xhtml"
  xmlns:jsp = "http://java.sun.com/JSP/Page">

  <head><title>Generate XML Element</title></head>

  <body>
    <jsp:element name = "xmlElement">
      <jsp:attribute name = "xmlElementAttr">
        Value for the attribute
      </jsp:attribute>

      <jsp:body>
        Body for XML element
      </jsp:body>

    </jsp:element>
  </body>
</html>
```

This would produce the following HTML code at run time –

```
<html        xmlns        =        "http://www.w3c.org/1999/xhtml"        xmlns:jsp        =
"http://java.sun.com/JSP/Page">
  <head><title>Generate XML Element</title></head>

  <body>
    <xmlElement xmlElementAttr = "Value for the attribute">
      Body for XML element
    </xmlElement>
  </body>
</html>
```

The <jsp:text> Action

The **<jsp:text>** action can be used to write the template text in JSP pages and documents. Following is the simple syntax for this action –

```
<jsp:text>Template data</jsp:text>
```

The body of the template cannot contain other elements; it can only contain text and EL expressions (Note – EL expressions are explained in a subsequent chapter). Note that in XML files, you cannot use expressions such as **${whatever > 0}**, because the greater than signs are illegal. Instead, use the **gt** form, such as **${whatever gt 0}** or an alternative is to embed the value in a **CDATA** section.

```
<jsp:text><![CDATA[<br>]]></jsp:text>
```

If you need to include a **DOCTYPE** declaration, for instance for **XHTML**, you must also use the **<jsp:text>** element as follows –

```
<jsp:text><![CDATA[<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "DTD/xhtml1-strict.dtd">]]></jsp:text>

  <head><title>jsp:text action</title></head>

  <body>
    <books><book><jsp:text>
      Welcome to JSP Programming
    </jsp:text></book></books>
  </body>
</html>
```

Try the above example with and without **<jsp:text>** action.