JSP – JAVA SERVER PAGE

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

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

2) Easy to maintain

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

3) Fast Development: No need to recompile and redeploy

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

4) Less code than Servlet

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

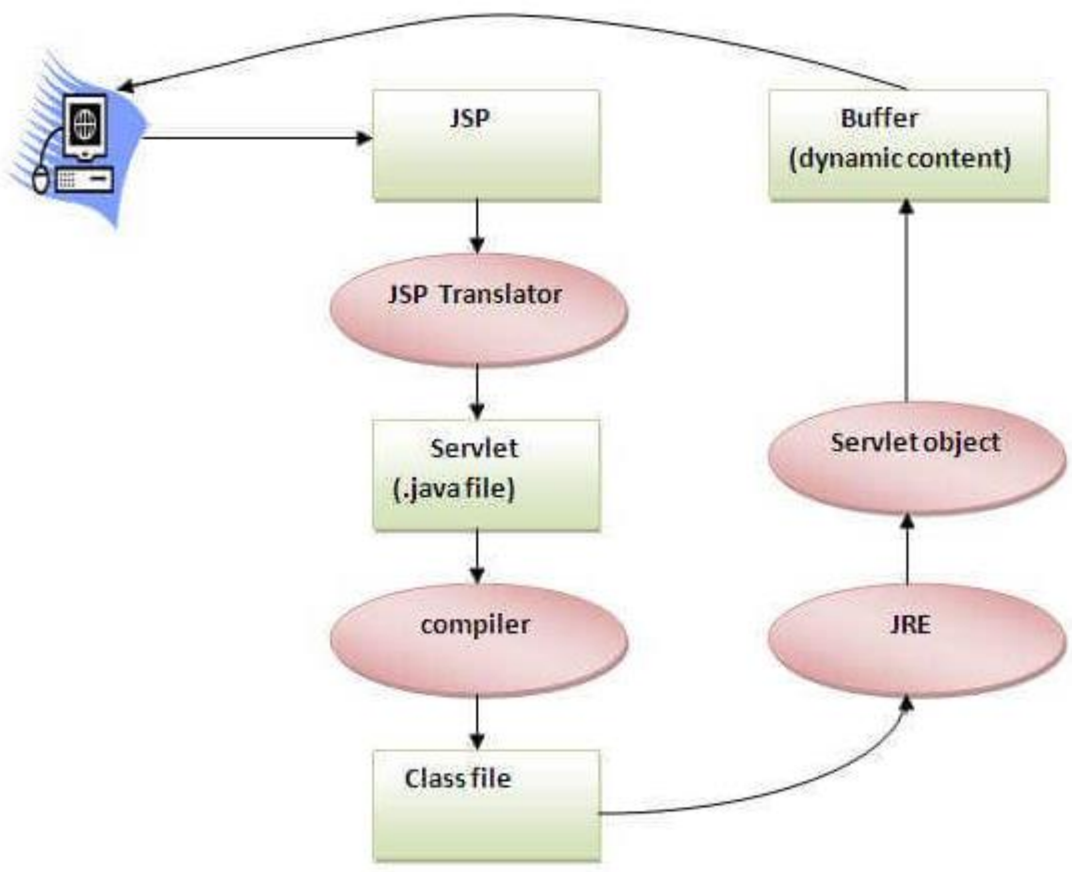
The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).

- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).

In eclipse – doGet()……, In netbeans – requestProcessing() – doGet(), doPost()



Creating a simple JSP Page

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

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

```
<% out.print(2*5); %>
</body>
</html>
```

It will print **10** on the browser.

How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

The JSP API

The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

javax.servlet.jsp package

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

1. JspPage
2. HttpJspPage

The classes are as follows:

- JspWriter
- PageContext
- JspFactory
- JspEngineInfo
- JspException
- JspError

The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.

Methods of JspPage interface

1. **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

The HttpJspPage interface

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

Method of HttpJspPage interface:

1. **public void _jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.

JSP Scripting elements

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag <>
- expression tag
- declaration tag

```
<%  java source code %>
```

Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

Example of JSP scriptlet tag that prints the user name

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body>
</html>
```

JSP expression tag

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

1. <%= statement %>

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

JSP expression tag that prints current time

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
```

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

JSP Declaration Tag

The **JSP declaration tag** is used *to declare fields and methods*.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

So it doesn't get memory at each request.

Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

```
<%!  field or method declaration %>
```

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>
<%! int data=50; %>
<%= "Value of the variable is:"+data %>
</body>
</html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JSP Implicit Objects

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.

The available implicit objects are out, request, config, session, application etc.

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext

A list of the 9 implicit objects is given below:

page	Object
exception	Throwable

JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter.

JSP request implicit object

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Example of JSP request implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

JSP response implicit object

In JSP, response is an implicit object of type HttpServletResponse.

The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

JSP config implicit object

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>

<servlet>
<servlet-name>Alliance</servlet-name>
<jsp-file>/welcome1.jsp</jsp-file>

<init-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>

</servlet>

<servlet-mapping>
<servlet-name>Alliance</servlet-name>
<url-pattern>/welcome1</url-pattern>
```

```
</servlet-mapping>

</web-app>
```

welcome.jsp

```
<%
out.print("Welcome "+request.getParameter("uname"));

String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

JSP application implicit object

In JSP, application is an implicit object of type *ServletContext*.

The instance of ServletContext is created only once by the web container when application or

project is deployed on the server.

This object can be used to get initialization parameter from configuaration file (web.xml).

It can also be used to get, set or remove attribute from the application scope.

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>
  <servlet>
<servlet-name>Alliance</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
</servlet>
  <servlet-mapping>
<servlet-name>Alliance</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
  <context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>
  </web-app>
```

welcome.jsp

```
<%
```

```
out.print("Welcome "+request.getParameter("uname"));

String driver=application.getInitParameter("dname");
out.print("driver name is="+driver);

%>
```

session implicit object

In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information.

Example of session implicit object

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%

String name=request.getParameter("uname");
out.print("Welcome "+name);

session.setAttribute("user",name);
  session.getID();%>

<a href="second.jsp">second jsp page</a>

</body>
</html>
```

second.jsp

```
<html>
<body>
<%

String name=(String)session.getAttribute("user");
out.print("Hello "+name);

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

pageContext implicit object

In JSP, pageContext is an implicit object of type PageContext class.
The pageContext object can be used to set,get or remove attribute from one of the following scopes:

- page
- request
- session
- application

In JSP, page scope is the default scope.

Example of pageContext implicit object

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%

String name=request.getParameter("uname");
```

```
out.print("Welcome "+name);

pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);

<a href="second.jsp">second jsp page</a>

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

second.jsp

```
<html>
<body>
<%

String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
out.print("Hello "+name);

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

page implicit object:

In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class. It is written as:

Object page=this;

For using this object it must be cast to Servlet type.For example:

<% (HttpServlet)page.log("message"); %>

Since, it is of type Object it is less used because you can use this object directly in jsp. For example:

<% this.log("message"); %>

exception implicit object

In JSP, exception is an implicit object of type java.lang.Throwable class.
This object can be used to print the exception. But it can only be used in error pages.
It is better to learn it after page directive.

Example of exception implicit object:

error.jsp

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

Sorry following exception occured:<%= exception %>

</body>
</html>
```

JSP directives

The **jsp directives** are messages that tells the web container how to translate a JSP page into the corr

There are three types of directives:

- page directive
- include directive
- taglib directive

Syntax of JSP Directive

```
1. <%@ directive attribute="value" %>
```

Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isELIgnored

- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

1)import

The import attribute is used to import class,interface or all the members of a package.It is similar t
in java class or interface.

Example of import attribute

```
<html>
<body>

<%@ page import="java.util.Date" %>
Today is: <%= new Date() %>

</body>
</html>
```

Jsp Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text fi
includes the original content of the included resource at page translation time (the jsp page is translate
better to include static resource).

Advantage of Include directive

Code Reusability

Syntax of include directive

```
1. <%@ include file="resourceName" %>
```

```
<html>
<body>
  <%@ include file="header.html" %>
  Today is: <%= java.util.Calendar.getInstance().getTime() %>
```

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

JSP Taglib directive

The JSP taglib directive is used to define a tag library that defines many tags.

We use the TLD (Tag Library Descriptor) file to define the tags.

Syntax JSP Taglib directive

```
<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary"


<html>
<body>
  <%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
  <mytag:currentDate/>
  </body>
</html>
```

Exception Handling in JSP

The exception is normally an object that is thrown at runtime. Exception Handling is the process to har
There may occur exception any time in your web application. So handling exceptions is a safer side fo
JSP, there are two ways to perform exception handling:

1. By **errorPage** and **isErrorPage** attributes of page directive
2. By **<error-page>** element in web.xml file

Eg)

- index.jsp for input values
- process.jsp for dividing the two numbers and displaying the result
- error.jsp for handling the exception

index.jsp

```
<form action="process.jsp">
No1:<input type="text" name="n1" /><br/><br/>
No1:<input type="text" name="n2" /><br/><br/>
<input type="submit" value="divide"/>
</form>
```

process.jsp

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

```
  String num1=request.getParameter("n1");
String num2=request.getParameter("n2");

int a=Integer.parseInt(num1);
int b=Integer.parseInt(num2);
int c=a/b;
out.print("division of numbers is: "+c);
 %>
```

error.jsp

```
<%@ page isErrorPage="true" %>
  <h3>Sorry an exception occured!</h3>
  Exception is: <%= exception %>
```

JSP Action Tags

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

The action tags are used to control the flow between pages and to use Java Bean.

The Jsp action tags are given below.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.

jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

jsp:forward action tag

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax of jsp:forward action tag without parameter

```
<jsp:forward page="relativeURL | <%= expression %>" />
```

Syntax of jsp:forward action tag with parameter

```
<jsp:forward page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:forward>
```

Example of jsp:forward action tag without parameter

In this example, we are simply forwarding the request to the printdate.jsp file.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" />
</body>
</html>
```

printdate.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Example of jsp:forward action tag with parameter

In this example, we are forwarding the request to the printdate.jsp file with parameter and printdate.jsp file prints the parameter value with date and time.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>

<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="Alliance" />
</jsp:forward>

</body>
</html>
```

printdate.jsp

```
<html>
<body>

<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
<%= request.getParameter("name") %>

</body>
</html>
```

jsp:include action tag

The **jsp:include action tag** is used to include the content of another resource it may be jsp, html or servlet.

The jsp include action tag includes the resource at request time so it is **better for dynamic pages** because there might be changes in future.

The jsp:include tag can be used to include static as well as dynamic pages

Advantage of jsp:include action tag

Code reusability : We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.

Difference between jsp include directive and include action

JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

Syntax of jsp:include action tag without parameter

```
<jsp:include page="relativeURL | <%= expression %>" />
```

Syntax of jsp:include action tag with parameter

```
<jsp:include page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>"
/>
</jsp:include>
```

Example of jsp:include action tag without parameter

In this example, index.jsp file includes the content of the printdate.jsp file.

File: index.jsp

```
<h2>this is index page</h2>

<jsp:include page="printdate.jsp" />

<h2>end section of index page</h2>
```

File: printdate.jsp

```
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime(
```

JavaBean

A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use JavaBean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

Simple example of JavaBean class

```
//Employee.java

package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;
public Employee(){}
public void setId(int id){this.id=id;}
public int getId(){return id;}
public void setName(String name){this.name=name;}
public String getName(){return name;}
}
```

How to access the JavaBean class?

To access the JavaBean class, we should use getter and setter methods

```
package mypack;
public class Test{
public static void main(String args[]){
Employee e=new Employee();//object is created
e.setName("Arjun");//setting value to the object
System.out.println(e.getName());
}}
```

JavaBean Properties

A JavaBean property is a named feature that can be accessed by the user of the object. The feature can be of any Java data type, containing the classes that you define.

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

1. getPropertyName ()

For example, if the property name is firstName, the method name would be getFirstName() to read that property. This method is called the accessor.

2. setPropertyName ()

For example, if the property name is firstName, the method name would be setFirstName() to write that property. This method is called the mutator.

Advantages of JavaBean

The following are the advantages of JavaBean:/p>

- The JavaBean properties and methods can be exposed to another application.
- It provides an easiness to reuse the software components.

Disadvantages of JavaBean

The following are the disadvantages of JavaBean:

- JavaBeans are mutable. So, it can't take advantages of immutable objects.
- Creating the setter and getter method for each property separately may lead to the boilerplate code.

jsp:useBean action tag

The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

Syntax of jsp:useBean action tag

```
<jsp:useBean id= "instanceName" scope= "page | request | session | application "
class= "packageName.className" type= "packageName.className"
beanName="packageName.className | <%= expression >" >
</jsp:useBean>
```

Attributes and Usage of jsp:useBean action tag

1. **id:** is used to identify the bean in the specified scope.
2. **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.
   - **page:** specifies that you can use this bean within the JSP page. The default scope is page.
   - **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
   - **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
   - **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
3. **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
4. **type:** provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
5. **beanName:** instantiates the bean using the java.beans.Beans.instantiate() method

Simple example of jsp:useBean action tag

Calculator.java (a simple Bean class)

```
1. package com.javatpoint;
public class Calculator{

public int cube(int n){return n*n*n;}

}
```

index.jsp file

```
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>

<%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

jsp:setProperty and jsp:getProperty action tags

The setProperty and getProperty action tags are used for developing web application with Java Bean. In web devlopment, bean class is mostly used because it is a reusable software component that represents data.

The jsp:setProperty action tag sets a property value or values in a bean using the setter method.

Syntax of jsp:setProperty action tag

```
<jsp:setProperty name="instanceOfBean" property= "*"   |
property="propertyName" param="parameterName"  |
property="propertyName" value="{ string | <%= expression %>}"
/>
```

Example of jsp:setProperty action tag if you have to set all the values of incoming request in the bean

```
1. <jsp:setProperty name="bean" property="*" />
```

Example of jsp:setProperty action tag if you have to set value of the incoming specific property

```
1. <jsp:setProperty name="bean" property="username" />
```

Example of jsp:setProperty action tag if you have to set a specific value in the property

```
1. <jsp:setProperty name="bean" property="username" value="Kumar" />
```

jsp:getProperty action tag

The jsp:getProperty action tag returns the value of the property

Syntax of jsp:getProperty action tag

```
1. <jsp:getProperty name="instanceOfBean" property="propertyName" />
```

Simple example of jsp:getProperty action tag

```
1. <jsp:getProperty name="obj" property="name" />
```

Example of bean development in JSP

In this example there are 3 pages:

- index.html for input of values
- welocme.jsp file that sets the incoming values to the bean object and prints the one value
- User.java bean class that have setter and getter methods

index.html

```
<form action="process.jsp" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
Email:<input type="text" name="email"><br>
<input type="submit" value="register">
</form>
```

process.jsp

```
<jsp:useBean id="u" class="org.sssit.User"></jsp:useBean>
<jsp:setProperty property="*" name="u"/>

Record:<br>
<jsp:getProperty property="name" name="u"/><br>
<jsp:getProperty property="password" name="u"/><br>
<jsp:getProperty property="email" name="u" /><br>
```

User.java

```
package org.sssit;

public class User {
private String name,password,email;
//setters and getters
}
```

Displaying applet in JSP (jsp:plugin action tag)

The jsp:plugin action tag is used to embed applet in the jsp file. The jsp:plugin action tag downloads
at client side to execute an applet or bean.

Syntax of jsp:plugin action tag

```
<jsp:plugin type= "applet | bean" code= "nameOfClassFile"
codebase= "directoryNameOfClassFile"
</jsp:plugin>
```

Example of displaying applet in JSP

In this example, we are simply displaying applet in jsp using the jsp:plugin tag. You mus MouseDrag.class file (an applet class file) in the current folder where jsp file resides. You may download this program that contains index.jsp, MouseDrag.java and MouseDrag.class files to r application.

index.jsp

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Mouse Drag</title>
   </head>
   <body bgcolor="khaki">
<h1>Mouse Drag Example</h1>

 <jsp:plugin align="middle" height="500" width="500"
    type="applet"  code="MouseDrag.class" name="clock" codebase="."/>

   </body>
</html>
```

Expression Language (EL) in JSP

The **Expression Language** (EL) simplifies the accessibility of data stored in the Java Bean component, and other objects like request, session, application etc.

There are many implicit objects, operators and reserve words in EL.

It is the newly added feature in JSP technology version 2.0.

Implicit Objects in Expression Language (EL)

There are many implicit objects in the Expression Language. They are as follows:

Implicit Objects	Usage
pageScope	it maps the given attribute name with the value set in the page scope

requestScope	it maps the given attribute name with the value set in the request scope
sessionScope	it maps the given attribute name with the value set in the session scope
applicationScope	it maps the given attribute name with the value set in the application scope
param	it maps the request parameter to the single value
paramValues	it maps the request parameter to an array of values
header	it maps the request header name to the single value
headerValues	it maps the request header name to an array of values
cookie	it maps the given cookie name to the cookie value
initParam	it maps the initialization parameter
pageContext	it provides access to many objects request, session etc.

EL param example

In this example, we have created two files index.jsp and process.jsp. The index.jsp file gets input from the user and sends the request to the process.jsp which in turn prints the name of the user using EL.

index.jsp

```
<form action="process.jsp">
Enter Name:<input type="text" name="name" /><br/><br/>
<input type="submit" value="go"/>
</form>
```

process.jsp

```
Welcome, ${ param.name }
```

EL sessionScope example

In this example, we printing the data stored in the session scope using EL. For this purpose, we have used sessionScope object.

index.jsp

```
<h3>welcome to index page</h3>
<%
session.setAttribute("user","sonoo");
%>

<a href="process.jsp">visit</a>
```

process.jsp

```
Value is ${ sessionScope.user }
```

download this example

EL cookie example

index.jsp

```
<h1>First JSP</h1>
<%
Cookie ck=new Cookie("name","abhishek");
response.addCookie(ck);
%>
<a href="process.jsp">click</a>
```

process.jsp

```
Hello, ${cookie.name.value}
```

Precedence of Operators in EL

There are many operators that have been provided in the Expression Language. Their precedence are as follows:

[] .
()
-(unary) not ! empty
* / div % mod

+ - (binary)
< <= > >= lt le gt ge
== != eq ne
&& and
|| or
?:

Reserve words in EL

There are many reserve words in the Expression Language. They are as follows:

lt	le	gt	ge
eq	ne	true	false
and	or	not	instanceof
div	mod	empty	null

MVC in JSP

MVC stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data.

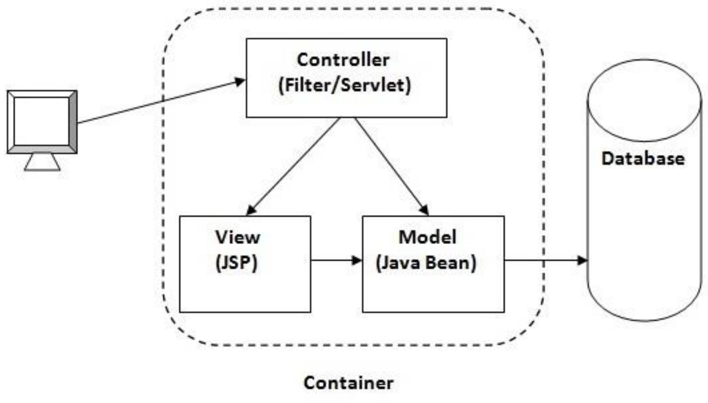
Controller acts as an interface between View and Model. Controller intercepts all the incoming requests.

Model represents the state of the application i.e. data. It can also have business logic

View represents the presentaion i.e. UI(User Interface).

Advantage of MVC (Model 2) Architecture

1. Navigation Control is centralized
2. Easy to maintain the large application



File: index.jsp

```
<form action="ControllerServlet" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
<input type="submit" value="login">
</form>
```

File: ControllerServlet

```
package com.javatpoint;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ControllerServlet extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse respons
e)
          throws ServletException, IOException {
       response.setContentType("text/html");
       PrintWriter out=response.getWriter();

       String name=request.getParameter("name");
       String password=request.getParameter("password");

       LoginBean bean=new LoginBean();
       bean.setName(name);
       bean.setPassword(password);
       request.setAttribute("bean",bean);

       boolean status=bean.validate();

       if(status){
            RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");
            rd.forward(request, response);
        }
        else{
            RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");
            rd.forward(request, response);
        }

    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
            throws ServletException, IOException {
        doPost(req, resp);
    }
}
```

File: LoginBean.java

```
package com.javatpoint;
public class LoginBean {
private String name,password;

public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}
public String getPassword() {
    return password;
}
public void setPassword(String password) {
    this.password = password;
}
public boolean validate(){
    if(password.equals("admin")){
        return true;
    }
    else{
        return false;
    }
}
}
```

File: login-success.jsp

```
<%@page import="com.javatpoint.LoginBean"%>

<p>You are successfully logged in!</p>
<%
LoginBean bean=(LoginBean)request.getAttribute("bean");
out.print("Welcome, "+bean.getName());
%>
```

File: login-error.jsp

```
<p>Sorry! username or password error</p>
<%@ include file="index.jsp" %>
```

File: web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/n
s/javaee/web-app_2_5.xsd"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/n
s/javaee/web-app_3_0.xsd"
id="WebApp_ID" version="3.0">

  <servlet>
  <servlet-name>s1</servlet-name>
  <servlet-class>com.javatpoint.ControllerServlet</servlet-class>
  </servlet>
  <servlet-mapping>
  <servlet-name>s1</servlet-name>
  <url-pattern>/ControllerServlet</url-pattern>
  </servlet-mapping>
</web-app>
```

Custom Tags in JSP

Custom tags are user-defined tags. They eliminates the possibility of scriptlet tag and separates the business logic from the JSP page.

The same business logic can be used many times by the use of custom tag.

Advantages of Custom Tags

The key advantages of Custom tags are as follows

1. **Eliminates the need of scriptlet tag** The custom tags eliminates the need of scriptlet tag which is considered bad programming approach in JSP.

2. **Separation of business logic from JSP** The custom tags separate the the business logic from the JSP page so that it may be easy to maintain.
3. **Re-usability** The custom tags makes the possibility to reuse the same business logic again and again.

Syntax to use custom tag

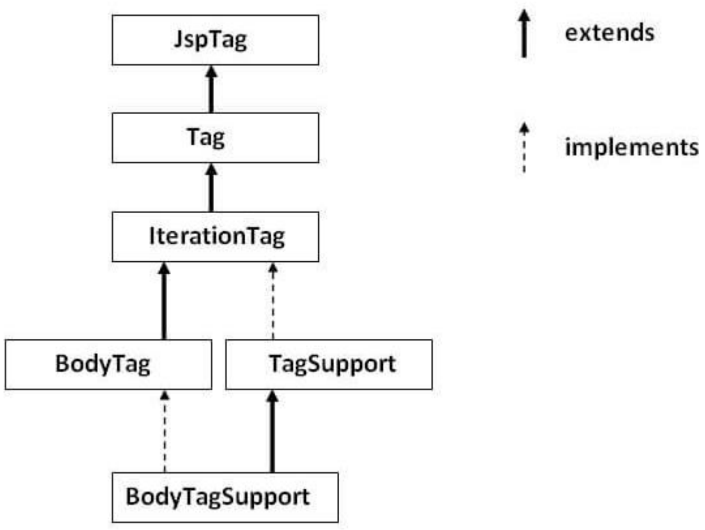
There are two ways to use the custom tag. They are given below:

1. **<prefix:tagname** attr1=value1....attrn=valuen **/>**

1. **<prefix:tagname** attr1=value1....attrn=valuen **>**
2. body code
3. **</prefix:tagname>**

JSP Custom Tag API

The javax.servlet.jsp.tagext package contains classes and interfaces for JSP custom tag API. The JspTag is the root interface in the Custom Tag hierarchy



JspTag interface

The JspTag is the root interface for all the interfaces and classes used in custom tag. It is a marker interface.

Tag interface

The Tag interface is the sub interface of JspTag interface. It provides methods to perform action at the start and end of the tag.

Fields of Tag interface

There are four fields defined in the Tag interface. They are:

Field Name	Description
public static int EVAL_BODY_INCLUDE	it evaluates the body content.
public static int EVAL_PAGE	it evaluates the JSP page content after the custom tag.
public static int SKIP_BODY	it skips the body content of the tag.
public static int SKIP_PAGE	it skips the JSP page content after the custom tag.

Methods of Tag interface

The methods of the Tag interface are as follows:

Method Name	Description
public void setPageContext(PageContext pc)	it sets the given PageContext object.
public void setParent(Tag t)	it sets the parent of the tag handler.
public Tag getParent()	it returns the parent tag handler object.
public int doStartTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this

	method and define the business logic to be performed at the start of the tag.
public int doEndTag()throws JspException	it is invoked by the JSP page implementation object. The JSP programmer should override this method and define the business logic to be performed at the end of the tag.
public void release()	it is invoked by the JSP page implementation object to release the state.

IterationTag interface

The IterationTag interface is the sub interface of the Tag interface. It provides an additional method to reevaluate the body.

Field of IterationTag interface

There is only one field defined in the IterationTag interface.

- **public static int EVAL_BODY_AGAIN** it reevaluates the body content.

Method of Tag interface

There is only one method defined in the IterationTag interface.

- **public int doAfterBody()throws JspException** it is invoked by the JSP page implementation object after the evaluation of the body. If this method returns EVAL_BODY_INCLUDE, body content will be reevaluated, if it returns SKIP_BODY, no more body cotent will be evaluated.

TagSupport class

The TagSupport class implements the IterationTag interface. It acts as the base class for new Tag Handlers. It provides some additional methods also.

Understanding Flow and Example of JSP Custom Tag

There is given two simple examples of JSP custom tag. One example of JSP custom tag, performs action at the start of the tag and second example performs action at the start and end of the tag.

Attributes in Custom Tag

Here, we will learn how we can define attributes for the custom tag.

Iteration using Custom Tag

In this example, we are iterating the body content of the custom tag.

Custom URI in Custom Tag

We may also refer the TLD file by using the URI. Here we will learn how can we use custom URI.

Example of JSP Custom Tag

In this example, we are going to create a **custom tag that prints the current date and time**. We are performing action at the start of tag.

For creating any custom tag, we need to follow following steps:

1. **Create the Tag handler class** and perform action at the start or at the end of the tag.
2. **Create the Tag Library Descriptor (TLD) file** and define tags
3. **Create the JSP file that uses the Custom tag defined in the TLD file**

File: MyTagHandler.java

```
package com.javatpoint.sonoo;
import java.util.Calendar;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
public class MyTagHandler extends TagSupport{

public int doStartTag() throws JspException {
    JspWriter out=pageContext.getOut();//returns the instance of JspWriter
    try{
```

```
    out.print(Calendar.getInstance().getTime());//printing date and time using JspWri
ter
   }catch(Exception e){System.out.println(e);}
   return SKIP_BODY;//will not evaluate the body content of the tag
}
}
```

2) Create the TLD file

Tag Library Descriptor (TLD) file contains information of tag and Tag Hander classes. It must be contained inside the **WEB-INF** directory.

File: mytags.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
   "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>simple</short-name>
  <uri>http://tomcat.apache.org/example-taglib</uri>

<tag>
<name>today</name>
<tag-class>com.javatpoint.sonoo.MyTagHandler</tag-class>
</tag>
</taglib>
```

3) Create the JSP file

Let's use the tag in our jsp file. Here, we are specifying the path of tld file directly. But it is recommended to use the uri name instead of full path of tld file. We will learn about uri later.

It uses **taglib** directive to use the tags defined in the tld file.

File: index.jsp

```
<%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>
Current Date and Time is: <m:today/>
```

Attributes in JSP Custom Tag

There can be defined too many attributes for any custom tag. To define the attribute, you need to perform two tasks:

Define the property in the TagHandler class with the attribute name and define the setter method

define the attribute element inside the tag element in the TLD file

Let's understand the attribute by the tag given below:

```
<m:cube number="4"></m:cube>
```

Simple example of attribute in JSP Custom Tag

In this example, we are going to use the cube tag which return the cube of any given number. Here, we are defining the number attribute for the cube tag. We are using the three file here:

index.jsp

CubeNumber.java

mytags.tld

index.jsp

```
<%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>
Cube of 4 is: <m:cube number="4"></m:cube>
```

CubeNumber.java

```
package com.javatpoint.taghandler;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;

public class CubeNumber extends TagSupport{
private int number;

public void setNumber(int number) {
    this.number = number;
}

public int doStartTag() throws JspException {
    JspWriter out=pageContext.getOut();
    try{
        out.print(number*number*number);
    }catch(Exception e){e.printStackTrace();}
    return SKIP_BODY;
  }
  }
```

mytags.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
        "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>simple</short-name>
  <uri>http://tomcat.apache.org/example-taglib</uri>
  <description>A simple tab library for the examples</description>

  <tag>
    <name>cube</name>
    <tag-class>com.javatpoint.taghandler.CubeNumber</tag-class>
    <attribute>
    <name>number</name>
    <required>true</required>
    </attribute>
  </tag>
</taglib>
```

JSP Custom Tag attribute example with database

Let's create a custom tag that prints a particular record of table for the given table name and id.

So, you have to have two properties in the tag handler class.

PrintRecord.java

```
package com.javatpoint;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
import java.sql.*;

public class PrintRecord extends TagSupport{
private String id;
private String table;

public void setId(String id) {
    this.id = id;
}
public void setTable(String table) {
    this.table = table;
}

public int doStartTag()throws JspException{
    JspWriter out=pageContext.getOut();
    try{
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con=DriverManager.getConnection(
                 "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
        PreparedStatement ps=con.prepareStatement("select * from "+table+" where i
d=?");
        ps.setInt(1,Integer.parseInt(id));
        ResultSet rs=ps.executeQuery();
        if(rs!=null){
        ResultSetMetaData rsmd=rs.getMetaData();
        int totalcols=rsmd.getColumnCount();
        //column name
        out.write("<table border='1'>");
        out.write("<tr>");
        for(int i=1;i<=totalcols;i++){
            out.write("<th>"+rsmd.getColumnName(i)+"</th>");
        }
        out.write("</tr>");
        //column value

        if(rs.next()){
            out.write("<tr>");
                for(int i=1;i<=totalcols;i++){
                out.write("<td>"+rs.getString(i)+"</td>");
            }
            out.write("</tr>");

        }else{
            out.write("Table or Id doesn't exist");
        }
        out.write("</table>");

        }
        con.close();
    }catch(Exception e){System.out.println(e);}
    return SKIP_BODY;
```

```
}
}
```

m.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
        PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
    "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>1.2</tlib-version>
  <jsp-version>2.0</jsp-version>
  <short-name>c</short-name>
  <uri>javatpoint</uri>

<tag>
<name>printRecord</name>
<tag-class>com.javatpoint.PrintRecord</tag-class>
<attribute>
<name>id</name>
<required>true</required>
</attribute>
<attribute>
<name>table</name>
<required>true</required>
</attribute>

</tag>
</taglib>
```

index.jsp

```
<%@ taglib uri="javatpoint" prefix="j" %>
<j:printRecord table="user874" id="1"></j:printRecord>
```

JSTL (JSP Standard Tag Library)

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

1. **Fast Development** JSTL provides many tags that simplify the JSP.
2. **Code Reusability** We can use the JSTL tags on various pages.

3. **No need to use scriptlet tag** It avoids the use of scriptlet tag.

JSTL Tags

There JSTL mainly provides five types of tags:

Tag Name	Description
Core tags	The JSTL core tag provide variable support, URL management, flow control, etc. The URL for the core tag is **http://java.sun.com/jsp/jstl/core**. The prefix of core tag is **c**.
Function tags	The functions tags provide support for string manipulation and string length. The URL for the functions tags is **http://java.sun.com/jsp/jstl/functions** and prefix is **fn**.
Formatting tags	The Formatting tags provide support for message formatting, number and date formatting, etc. The URL for the Formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt**.
XML tags	The XML tags provide flow control, transformation, etc. The URL for the XML tags is **http://java.sun.com/jsp/jstl/xml** and prefix is **x**.
SQL tags	The JSTL SQL tags provide SQL support. The URL for the SQL tags is **http://java.sun.com/jsp/jstl/sql** and prefix is **sql**.

JSTL Core Tags

The JSTL core tag provides variable support, URL management, flow control etc. The syntax used for including JSTL core library in your JSP is:

```
1. <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

JSTL Core Tags List

Tags	Description

c:out	It display the result of an expression, similar to the way <%=...%> tag work.
c:import	It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.
c:set	It sets the result of an expression under evaluation in a 'scope' variable.
c:remove	It is used for removing the specified scoped variable from a particular scope.
c:catch	It is used for Catches any Throwable exceptions that occurs in the body.
c:if	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
c:choose, c:when, c:otherwise	It is the simple conditional tag that includes its body content if the evaluated condition is true.
c:forEach	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.
c:forTokens	It iterates over tokens which is separated by the supplied delimeters.
c:param	It adds a parameter in a containing 'import' tag's URL.
c:redirect	It redirects the browser to a new URL and supports the context-relative URLs.
c:url	It creates a URL with optional query parameters.

JSTL Core <c:out> Tag

The <c:out> tag is similar to JSP expression tag, but it can only be used with expression. It will display the result of an expression, similar to the way < %=...% > work.

The < c:out > tag automatically escape the XML tags. Hence they aren't evaluated as actual tags.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:out value="${'Welcome to javaTpoint'}"/>
</body>
</html>
```

JSTL Core <c:import> Tag

The <c:import> is similar to jsp 'include', with an additional feature of including the content of any resource either within server or outside the server.

This tag provides all the functionality of the <include > action and it also allows the inclusion of absolute URLs.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:import var="data" url="http://www.javatpoint.com"/>
<c:out value="${data}"/>
</body>
</html>
```

JSTL Core <c:set> Tag

It is used to set the result of an expression evaluated in a 'scope'. The <c:set> tag is helpful because it evaluates the expression and use the result to set a value of java.util.Map or JavaBean.

This tag is similar to jsp:setProperty action tag.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
```

```
<body>
<c:set var="Income" scope="session" value="${4000*4}"/>
<c:out value="${Income}"/>
</body>
</html>
```

JSTL Core <c:remove> Tag

It is used for removing the specified variable from a particular scope. This action is not particularly helpful, but it can be used for ensuring that a JSP can also clean up any scope resources.

The <c:remove > tag removes the variable from either a first scope or a specified scope

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<p>Before Remove Value is: <c:out value="${income}"/></p>
<c:remove var="income"/>
<p>After Remove Value is: <c:out value="${income}"/></p>
</body>
</html>
```

JSTL Core <c:catch> Tag

It is used for Catches any Throwable exceptions that occurs in the body and optionally exposes it. In general it is used for error handling and to deal more easily with the problem occur in program.

The < c:catch > tag catches any exceptions that occurs in a program body.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>

<c:catch var ="catchtheException">
   <% int x = 2/0;%>
</c:catch>

<c:if test = "${catchtheException != null}">
```

```
  <p>The type of exception is : ${catchtheException} <br />
  There is an exception: ${catchtheException.message}</p>
</c:if>

</body>
</html>
```

JSTL Core <c:if> Tag

The < c:if > tag is used for testing the condition and it display the body content, if the expression evaluated is true.

It is a simple conditional tag which is used for evaluating the body content, if the supplied condition is true.

Let's see the simple example of c:if tag:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<c:if test="${income > 8000}">
  <p>My income is: <c:out value="${income}"/><p>
</c:if>
</body>
</html>
```

JSTL Core <c:choose>, <c:when>, <c:otherwise> Tag

The < c:choose > tag is a conditional tag that establish a context for mutually exclusive conditional operations. It works like a Java **switch** statement in which we choose between a numbers of alternatives.

The <c:when > is subtag of <choose > that will include its body if the condition evaluated be 'true'.

The < c:otherwise > is also subtag of < choose > it follows &l;twhen > tags and runs only if all the prior condition evaluated is 'false'

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
```

```
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<p>Your income is : <c:out value="${income}"/></p>
<c:choose>
   <c:when test="${income <= 1000}">
     Income is not good.
   </c:when>
   <c:when test="${income > 10000}">
      Income is very good.
   </c:when>
   <c:otherwise>
     Income is undetermined...
   </c:otherwise>
</c:choose>
</body>
</html>
```

JSTL Core <c:forEach> Tag

The <c:for each > is an iteration tag used for repeating the nested body content for fixed number of times or over the collection.

These tag used as a good alternative for embedding a Java **while, do-while, or for** loop via a scriptlet. The < c:for each > tag is most commonly used tag because it iterates over a collection of object.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:forEach var="j" begin="1" end="3">
  Item <c:out value="${j}"/><p>
</c:forEach>
</body>
</html>
```

JSTL Core <c:forTokens> Tag

The < c:forTokens > tag iterates over tokens which is separated by the supplied delimeters. It is used for break a string into tokens and iterate through each of the tokens to generate output.

This tag has similar attributes as < c:forEach > tag except one additional attributes **delims** which is used for specifying the characters to be used as delimiters.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:forTokens items="Rahul-Nakul-Rajesh" delims="-" var="name">
  <c:out value="${name}"/><p>
</c:forTokens>
</body>
</html>
```

JSTL Core <c:param> Tag

The < c:param > tag add the parameter in a containing 'import' tag's URL. It allow the proper URL request parameter to be specified within URL and it automatically perform any necessary URL encoding.

Inside < c:param > tag, the value attribute indicates the parameter value and name attribute indicates the parameter name.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:url value="/index1.jsp" var="completeURL"/>
 <c:param name="trackingId" value="786"/>
 <c:param name="user" value="Nakul"/>
</c:url>
${completeURL}
</body>
</html>
```

JSTL Core <c:redirect> Tag

The < c:redirect > tag redirects the browser to a new URL. It supports the context-relative URLs, and the < c:param > tag.

It is used for redirecting the browser to an alternate URL by using automatic URL rewriting.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
```

```
<head>
<title>Core Tag Example</title>
</head>
<body>
 <c:set var="url" value="0" scope="request"/>
 <c:if test="${url<1}">
   <c:redirect url="http://javatpoint.com"/>
 </c:if>
 <c:if test="${url>1}">
   <c:redirect url="http://facebook.com"/>
 </c:if>
</body>
</html>
```

JSTL Core <c:url> Tag

The < c:url > tag creates a URL with optional query parameter. It is used for url encoding or url formatting. This tag automatically performs the URL rewriting operation.

The JSTL url tag is used as an alternative method of writing call to the response.encodeURL() method. The advantage of url tag is proper URL encoding and including the parameters specified by children. **param** tag.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:url value="/RegisterDao.jsp"/>
</body>
</html>
```

JSTL Function Tags

The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions. The syntax used for including JSTL function library in your JSP is:

```
1. <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
```

JSTL Function Tags List

JSTL Functions	Description

fn:contains()	It is used to test if an input string containing the specified substring in a program.
fn:containsIgnoreCase()	It is used to test if an input string contains the specified substring as a case insensitive way.
fn:endsWith()	It is used to test if an input string ends with the specified suffix.
fn:escapeXml()	It escapes the characters that would be interpreted as XML markup.
fn:indexOf()	It returns an index within a string of first occurrence of a specified substring.
fn:trim()	It removes the blank spaces from both the ends of a string.
fn:startsWith()	It is used for checking whether the given string is started with a particular string value.
fn:split()	It splits the string into an array of substrings.
fn:toLowerCase()	It converts all the characters of a string to lower case.
fn:toUpperCase()	It converts all the characters of a string to upper case.
fn:substring()	It returns the subset of a string according to the given start and end position.
fn:substringAfter()	It returns the subset of string after a specific substring.
fn:substringBefore()	It returns the subset of string before a specific substring.
fn:length()	It returns the number of characters inside a string, or the number of items in a collection.

fn:replace()	It replaces all the occurrence of a string with another string sequence.

JSTL fn:contains() Function

The fn:contains() is used for testing if the string containing the specified substring. If the specified substring is found in the string, it returns true otherwise false.

The syntax used for including the fn:contains() function is:

```
boolean contains(java.lang.String, java.lang.String)
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title>Using JSTL Functions</title>
</head>
<body>

<c:set var="String" value="Welcome to javatpoint"/>

<c:if test="${fn:contains(String, 'javatpoint')}">
  <p>Found javatpoint string<p>
</c:if>

<c:if test="${fn:contains(String, 'JAVATPOINT')}">
  <p>Found JAVATPOINT string<p>
</c:if>

</body>
</html>
```

JSTL fn:containsIgnoreCase() Function

The fn:containsIgnoreCase() function is used to test if an input string contains the specified substring as a case insensitive way. During searching the specified substring it ignores the case

The syntax used for including the fn:containsIgnoreCase() function is:

```
boolean containsIgnoreCase(java.lang.String, java.lang.String)


<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
```

```
<html>
<head>
<title>Using JSTL Functions</title>
</head>
<body>

<c:set var="String" value="Welcome to javatpoint"/>

<c:if test="${fn:containsIgnoreCase(String, 'javatpoint')}">
   <p>Found javatpoint string<p>
</c:if>

<c:if test="${fn:containsIgnoreCase(String, 'JAVATPOINT')}">
   <p>Found JAVATPOINT string<p>
</c:if>

</body>
</html>
```

JSTL fn:endsWith() Function

The fn:endsWith() function is used for testing if an input string ends with the specified suffix. If the string ends with a specified suffix, it returns true otherwise false.

The syntax used for including the fn:endsWith() function is:

```
boolean endsWith(java.lang.String, java.lang.String)
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title>Using JSTL Functions</title>
</head>
<body>

<c:set var="String" value="Welcome to JSP programming"/>

<c:if test="${fn:endsWith(String, 'programming')}">
   <p>String ends with programming<p>
</c:if>

<c:if test="${fn:endsWith(String, 'JSP')}">
   <p>String ends with JSP<p>
</c:if>
```

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

JSTL fn:escapeXml() Function

The fn:escapeXml() function escapes the characters that would be interpreted as XML markup. It is used for escaping the character in XML markup language.

The syntax used for including the fn:escapeXml() function is:

```
java.lang.String escapeXml(java.lang.String)
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title>Using JSTL Functions</title>
</head>
<body>

<c:set var="string1" value="It is first String."/>
<c:set var="string2" value="It is <xyz>second String.</xyz>"/>

<p>With escapeXml() Function:</p>
<p>string-1 : ${fn:escapeXml(string1)}</p>
<p>string-2 : ${fn:escapeXml(string2)}</p>

<p>Without escapeXml() Function:</p>
<p>string-1 : ${string1}</p>
<p>string-2 : ${string2}</p>

</body>
</html>
```

JSTL fn:indexOf() Function

The fn:indexOf() function return an index of string. It is used for determining the index of string specified in substring.

The syntax used for including the fn:indexOf() function is:

```
int indexOf(java.lang.String, java.lang.String)


<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
```

```
<head>
<title>Using JSTL Functions</title>
</head>
<body>

<c:set var="string1" value="It is first String."/>
<c:set var="string2" value="It is <xyz>second String.</xyz>"/>

<p>Index-1 : ${fn:indexOf(string1, "first")}</p>
<p>Index-2 : ${fn:indexOf(string2, "second")}</p>

</body>
</html>
```

JSTL fn:startsWith() Function

The fn:startsWith() function test if an input string is started with the specified substring.

The syntax used for including the fn:startsWith() function is:

```
boolean fn:startsWith(String input, String prefix)
```

This function is used for returning a boolean value. It gives the true result when the string is started with the given prefix otherwise it returns a false result

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title>Using JSTL Function</title>
</head>
<body>
<c:set var="msg" value="The Example of JSTL fn:startsWith() Function"/>
The string starts with "The": ${fn:startsWith(msg, 'The')}
<br>The string starts with "Example": ${fn:startsWith(msg, 'Example')}
</body>
</html>
```

JSTL fn:split() Function

The fn:split() function splits the string into an array of substrings. It is used for splitting the string into an array based on a delimiter string.

The syntax used for including the fn:split() function is:

```
java.lang.String[] split(java.lang.String, java.lang.String)
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title>Using JSTL Functions</title>
</head>
<body>

<c:set var="str1" value="Welcome-to-JSP-Programming."/>
<c:set var="str2" value="${fn:split(str1, '-')}" />
<c:set var="str3" value="${fn:join(str2, ' ')}" />

<p>String-3 : ${str3}</p>
<c:set var="str4" value="${fn:split(str3, ' ')}" />
<c:set var="str5" value="${fn:join(str4, '-')}" />

<p>String-5 : ${str5}</p>

</body>
</html>
```

JSTL fn:toLowerCase() Function

The fn:toLowerCase() function converts all the characters of a string to lower case. It is used for replacing any upper case character in the input string with the corresponding lowercase character.

The syntax used for including the fn:toLowerCase() function is:

```
String fn:toLowerCase(String  input)
```

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<html>
<head>
<title> Using JSTL Function </title>
</head>
<body>
<c:set var="string" value="Welcome to JSP Programming"/>
${fn:toLowerCase("HELLO,")}
${fn:toLowerCase(string)}
</body>
</html>
```

JSTL Formatting tags

The formatting tags provide support for message formatting, number and date formatting etc. The url for the formatting tags is **http://java.sun.com/jsp/jstl/fmt** and prefix is **fmt.**

The JSTL formatting tags are used for internationalized web sites to display and format text, the time, the date and numbers. The syntax used for including JSTL formatting library in your JSP is:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"  %>
```

Formatting Tags	Descriptions
fmt:parseNumber	It is used to Parses the string representation of a currency, percentage or number.
fmt:timeZone	It specifies a parsing action nested in its body or the time zone for any time formatting.
fmt:formatNumber	It is used to format the numerical value with specific format or precision.
fmt:parseDate	It parses the string representation of a time and date.
fmt:bundle	It is used for creating the ResourceBundle objects which will be used by their tag body.
fmt:setTimeZone	It stores the time zone inside a time zone configuration variable.
fmt:setBundle	It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.
fmt:message	It display an internationalized message.
fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.

JSTL XML tags

The JSTL XML tags are used for providing a JSP-centric way of manipulating and creating XML documents.

The xml tags provide flow control, transformation etc. The url for the xml tags is **http://java.sun.com/jsp/jstl/xml** and prefix is x. The JSTL XML tag library has custom tags used for interacting with XML data. The syntax used for including JSTL XML tags library in your JSP is:

```
1. <%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>
```

Xalan.jar: Download this jar file from the link:

1. http://xml.apache.org/xalan-j/index.html

XercesImpl.jar: Download this jar file from the link:

1. http://www.apache.org/dist/xerces/j/

JSTL XML tags List

XML Tags	Descriptions
x:out	Similar to <%= ... > tag, but for XPath expressions.
x:parse	It is used for parse the XML data specified either in the tag body or an attribute.
x:set	It is used to sets a variable to the value of an XPath expression.
x:choose	It is a conditional tag that establish a context for mutually exclusive conditional operations.
x:when	It is a subtag of that will include its body if the condition evaluated be 'true'.
x:otherwise	It is subtag of that follows tags and runs only if all the prior conditions evaluated be 'false'.
x:if	It is used for evaluating the test XPath expression and if it is true, it will processes its body content.

x:transform	It is used in a XML document for providing the XSL(Extensible Stylesheet Language) transformation.
x:param	It is used along with the transform tag for setting the parameter in the XSLT style sheet.

JSTL XML <x:out> Tag

The <x:out> tag is used for displaying the result of an xml Path expression and writes the result to JSP writer object. It is similar to the scriptlet tag <%= %> used in JSP.

The syntax used for including the <x:out> tag is:

```
1.  <x:out attributes/>
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
 <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

<html>
<head>
 <title>XML Tags</title>
</head>
<body>
<h2>Vegetable Information:</h2>
<c:set var="vegetable">
<vegetables>
    <vegetable>
      <name>onion</name>
      <price>40/kg</price>
    </vegetable>
 <vegetable>
      <name>Potato</name>
      <price>30/kg</price>
    </vegetable>
 <vegetable>
      <name>Tomato</name>
      <price>90/kg</price>
    </vegetable>
</vegetables>
</c:set>
<x:parse xml="${vegetable}" var="output"/>
<b>Name of the vegetable is</b>:
```

```
<x:out select="$output/vegetables/vegetable[1]/name" /><br>
<b>Price of the Potato is</b>:
<x:out select="$output/vegetables/vegetable[2]/price" />
</body>
</html>
```

JSTL XML <x:parse> Tag

The <x:parse> tag is used for parse the XML data specified either in the tag body or an attribute. It is used for parse the xml content and the result will stored inside specified variable.

The syntax used for including the <x:parse> tag is:

```
<x:parse attributes> body content </x:parse>
<books>
<book>
  <name>Three mistakes of my life</name>
  <author>Chetan Bhagat</author>
  <price>200</price>
</book>
<book>
  <name>Tomorrow land</name>
  <author>NUHA</author>
  <price>2000</price>
</book>
</books>
```

Now put the following content in index.jsp, keeping in the same directory:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

<html>
<head>
  <title>x:parse Tag</title>
</head>
<body>
<h2>Books Info:</h2>
<c:import var="bookInfo" url="novels.xml"/>

<x:parse xml="${bookInfo}" var="output"/>
<p>First Book title: <x:out select="$output/books/book[1]/name" /></p>
<p>First Book price: <x:out select="$output/books/book[1]/price" /></p>
<p>Second Book title: <x:out select="$output/books/book[2]/name" /></p>
<p>Second Book price: <x:out select="$output/books/book[2]/price" /></p>
```

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

JSTL XML <x:set> Tag

The <x:set> tag is used to set a variable with the value of an XPath expression. It is used to store the result of xml path expression in a scoped variable.

The syntax used for including the <x:set> tag is:

```
<x:set attributes/>
```

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head>
 <title>x:set Tag</title>
</head>
<body>
<h3>Books Information:</h3>
<c:set var="book">
<books>
<book>
  <name>Three mistakes of my life</name>
  <author>Chetan Bhagat</author>
  <price>200</price>
</book>
<book>
  <name>Tomorrow land</name>
  <author>Brad Bird</author>
  <price>2000</price>
</book>
</books>
</c:set>
<x:parse xml="${book}" var="output"/>
<x:set var="fragment" select="$output/books/book[2]/price"/>
<b>The price of the Tomorrow land book</b>:
<x:out select="$fragment" />
</body>
</html>
```

JSTL XML <x:choose>, <x:when>, <x:otherwise> Tags

The <x:choose> tag is a conditional tag that establish a context for mutually exclusive conditional operations. It works like a Java **switch** statement in which we choose between a numbers of alternatives.

The <x:when> is subtag of <x:choose> that will include its body if the condition evaluated be 'true'.

The <x:otherwise> is also subtag of <x:choose> it follows <x:when> tags and runs only if all the prior condition evaluated is 'false'.

The <x:when> and <x:otherwise> works like if-else statement. But it must be placed inside <x:choose> tag.

The syntax used for including the <x:choose;> tag is:

```
<x:choose> body content </x:choose>
```

The syntax used for including the <x:when> tag is:

```
<x:when attribute> body content </x:when>
```

The syntax used for including the < x:otherwise > tag is:

```
<x:otherwise> body content </x:otherwise>
```

Let's see the simple example to understand the xml <x:choose>, <x:when>, <x:otherwise> tag:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

<html>
<head>
  <title>x:choose Tag</title>
</head>
<body>
<h3>Books Information:</h3>

<c:set var="xmltext">
<books>
<book>
  <name>Three mistakes of my life</name>
  <author>Chetan Bhagat</author>
  <price>200</price>
</book>
<book>
  <name>Tomorrow land</name>
  <author>Brad Bird</author>
```

```
 <price>2000</price>
</book>
</books>
</c:set>

<x:parse xml="${xmltext}" var="output"/>
<x:choose>
   <x:when select="$output//book/author = 'Chetan bhagat'">
      Book is written by Chetan bhagat
   </x:when>
   <x:when select="$output//book/author = 'Brad Bird'">
      Book is written by Brad Bird
   </x:when>
   <x:otherwise>
      The author is unknown...
   </x:otherwise>
</x:choose>

</body>
</html>
```

JSTL XML <x:if> Tag

The <x:if> tag is used for evaluating the test XPath expression. It is a simple conditional tag which is used for evaluating its body if the supplied condition is true.

The syntax used for including the <x:if> tag is:

```
<x:if attributes> body content </x:if>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head>
 <title>x:if Tags</title>
</head>
<body>
<h2>Vegetable Information:</h2>
<c:set var="vegetables">
<vegetables>
    <vegetable>
      <name>onion</name>
      <price>40</price>
    </vegetable>
 <vegetable>
      <name>Potato</name>
      <price>30</price>
    </vegetable>
 <vegetable>
      <name>Tomato</name>
      <price>90</price>
    </vegetable>
</vegetables>
</c:set>
<x:parse xml="${vegetables}" var="output"/>
<x:if select="$output/vegetables/vegetable/price < 100">
  Vegetables prices are very low.
</x:if>
</body>
</html>
```

JSTL XML <x:transform> Tag

The <x:transform> tag is used in a XML document for providing the XSL (Extensible Stylesheet Language) transformation. It is used for transforming xml data based on XSLT script.

The syntax used for including the <x:transform> tag is:

```
<x:transform attributes> body content </x:transform>


<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform
">
<xsl:param name="doc"/>
<xsl:template match="/">
<html>
<body>
<h2>Company's Employee detail</h2>
<table border="2">
<tr>
<th align="left">Name
</th>
<th align="left">Designation
</th>
<th align="left">Age
</th>
</tr>
<xsl:for-each select="organisation/company/emp">
<tr>
<td>
<xsl:value-of select="name"/>
```

```
</td>
<td>
<xsl:value-of select="designation"/>
</td>
<td>
<xsl:value-of select="age"/>
</td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Now put the following program in **transfer.xml**, keeping in the same directory:

```
<?xml version="1.0" encoding="UTF-8"?>
<organisation>
<company>
<emp>
<name>Rajan Singh</name>
<designation>Bussiness Developer</designation>
<age>40</age>
</emp>

<emp>
<name>Supriya Gaur</name>
<designation>HR Executive</designation>
<age>22</age>
</emp>
</company>

<company>
<emp>
<name>Shashnak Singhal</name>
<designation>Sr. Java Programmer</designation>
<age>26</age>
</emp>

<emp>
<name>Hemant Kishor</name>
<designation>Sr. PHP Programmer</designation>
<age>23</age>
```

```
</emp></company></organisation>
```

Now put the following program in **index.jsp**, keeping in the same directory:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head>
  <title>x:transform Tag</title>
</head>
</html>
<c:import var="xml" url="transfer.xml" />
<c:import var="xsl" url="transfer.xsl" />
<x:transform xml="${xml}" xslt="${xsl}" />
```

JSTL XML <x:param> Tag

The <x:param> tag is used to set the parameter in the XSLT style sheet. It use along with the transform tag for sending parameter along with the value.

The syntax used for including the < x:param > tag is:

```
<x:param name="name" value="value"></x:param>
```

transfer.xsl file:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0
">
<xsl:output method="html" indent="yes"/>
<xsl:param name="bgColor"/>

<xsl:template match="/">
  <html>
  <body>
   <xsl:apply-templates/>
  </body>
  </html>
</xsl:template>

<xsl:template match="books">
  <table border="1" width="60%" bgColor="{$bgColor}">
    <xsl:for-each select="book">
      <tr>
        <td>
          <b><xsl:value-of select="name"/></b>
```

```
         </td>
         <td>
           <xsl:value-of select="author"/>
         </td>
         <td>
           <xsl:value-of select="price"/>
         </td>
       </tr>
     </xsl:for-each>
   </table>
 </xsl:template>
 </xsl:stylesheet>
```

Now put the following program in **index.jsp** , keeping in the same directory:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>

<html>
<head>
  <title>x:transform Tag</title>
</head>
<body>
<h3>Novels Information:</h3>
<c:set var="xmltext">
 <books>
<book>
  <name>Three mistakes of my life</name>
  <author>Chetan Bhagat</author>
  <price>200</price>
</book>
<book>
  <name>Tomorrow land</name>
  <author>Brad Bird</author>
  <price>1000</price>
</book>
<book>
  <name>Wings of fire</name>
  <author>Dr. APJ Abdul Kalam</author>
  <price>500</price>
</book>
</books>
</c:set>
```

```
<c:import url="transfer.xsl" var="xslt"/>
<x:transform xml="${xmltext}" xslt="${xslt}">
  <x:param name="bgColor" value="yellow"/>
</x:transform>

</body>
</html>
```

JSTL SQL Tags

The JSTL sql tags provide SQL support. The url for the sql tags is **http://java.sun.com/jsp/jstl/sql** and prefix is **sql**.

The SQL tag library allows the tag to interact with RDBMSs (Relational Databases) such as Microsoft SQL Server, mySQL, or Oracle. The syntax used for including JSTL SQL tags library in your JSP is:

```
1. <%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
```

JSTL SQL Tags List

SQL Tags	Descriptions
sql:setDataSource	It is used for creating a simple data source suitable only for prototyping.
sql:query	It is used for executing the SQL query defined in its sql attribute or the body.
sql:update	It is used for executing the SQL update defined in its sql attribute or in the tag body.
sql:param	It is used for sets the parameter in an SQL statement to the specified value.
sql:dateParam	It is used for sets the parameter in an SQL statement to a specified java.util.Date value.
sql:transaction	It is used to provide the nested database action with a common connection.

JSTL SQL <sql:setDataSource> Tag

The <sql:setDataSource> tag is used for creating a simple data source suitable only for prototyping.

It is used to create the data source variable directly from JSP and it is stored inside a scoped variable. It can be used as input for other database actions

Consider the below information about your MySQL database setup:

- We are using the **JDBC MySQL driver**
- We are using the **test** database on local machine
- We are using the **"root"** as username and **"1234"** as password to access the test database.

Let's see the simple example to understand the xml <sql:setDataSource> tag is:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
<head>
<title>sql:setDataSource Tag</title>
</head>
<body>

<sql:setDataSource var="db" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/test"
    user="root"  password="1234"/>
</body>
</html>
```

JSTL SQL <sql:query> Tag

The <sql:query> tag is used for executing the SQL query defined in its sql attribute or the body. It is used to execute an SQL SELECT statement and saves the result in scoped variable.

Example:

```
<sql:query dataSource="${db}" var="rs">
SELECT * from Students;
</sql:query>
```

We are using the **JDBC MySQL driver**

We are using the **test** database on local machine

We are using the **"root"** as username and **"1234"** as password to access the test database.

To understand the basic concept, let us create a simple table **Students** in the test database and creates the few records in that table using **command prompts** as follows:

Step-1: Open the command prompt and change to the installation directory as follows:

```
C:\Users\javatpoint>
C:\Users\javatpoint>cd C:\Program Files\MySQL\MySQL Server 5.7\bin
C:\Program Files\MySQL\MySQL Server 5.7\bin>
```

Step – 2: C:\Program Files\MySQL\MySQL Server 5.7\bin>mysql -u root -p

```
Enter password: ****
mysql>
```

Step-3: Create the table Students in test database as shown below:

```
mysql> use test;
mysql> create table Students
   (
    id int not null,
    First_Name varchar (255),
    Last_Name varchar (255),
    Age int not null
   );
Query OK, 0 rows affected (0.08 sec)
mysql>
```

Step 4: In final step you need to create few data records in Students table as shown below:

```
mysql> INSERT INTO Students VALUES (150, 'Nakul', 'Jain', 22);
Query OK, 1 row affected (0.05 sec)

mysql>  INSERT INTO Students VALUES (151, 'Ramesh', 'Kumar', 20);
Query OK, 1 row affected (0.00 sec)

mysql>  INSERT INTO Students VALUES (152, 'Ajeet', 'Singhal', 22);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Students VALUES (153, 'Hamza', 'Hussain', 22);
Query OK, 1 row affected (0.00 sec)
```

simple JSP example to understand the use of <sql:query> tag is:

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
```

```
<html>
<head>
<title>sql:query Tag</title>
</head>
<body>

<sql:setDataSource var="db" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/test"
    user="root"  password="1234"/>

<sql:query dataSource="${db}" var="rs">
SELECT * from Students;
</sql:query>

<table border="2" width="100%">
<tr>
<th>Student ID</th>
<th>First Name</th>
<th>Last Name</th>
<th>Age</th>
</tr>
<c:forEach var="table" items="${rs.rows}">
<tr>
<td><c:out value="${table.id}"/></td>
<td><c:out value="${table.First_Name}"/></td>
<td><c:out value="${table.Last_Name}"/></td>
<td><c:out value="${table.Age}"/></td>
</tr>
</c:forEach>
</table>

</body>
</html>
```

JSTL SQL <sql:update> Tag

The <sql:update> tag is used for executing the SQL DML query defined in its sql attribute or in the tag body. It may be SQL UPDATE, INSERT or DELETE statements.

Example:

```
<sql:update dataSource="${db}" var="count">
INSERT INTO Students VALUES (154,'Nasreen', 'jaha', 25);
</sql:update>
```

simple JSP example to understand the use of <sql:update> tag is:

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>
<head>
<title>sql:update Tag</title>
</head>
<body>

<sql:setDataSource var="db" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/test"
    user="root"  password="1234"/>
<sql:update dataSource="${db}" var="count">
INSERT INTO Students VALUES (154,'Nasreen', 'jaha', 25);
</sql:update>

<sql:query dataSource="${db}" var="rs">
SELECT * from Students;
</sql:query>

<table border="2" width="100%">
<tr>
<th>Student ID</th>
<th>First Name</th>
<th>Last Name</th>
<th>Age</th>
</tr>
<c:forEach var="table" items="${rs.rows}">
<tr>
<td><c:out value="${table.id}"/></td>
<td><c:out value="${table.First_Name}"/></td>
<td><c:out value="${table.Last_Name}"/></td>
<td><c:out value="${table.Age}"/></td>
</tr>
</c:forEach>
</table>

</body>
</html>
```

JSTL SQL <sql:transaction> Tag

The <sql:transaction> tag is used for transaction management. It is used to group multiple <sql:update> into common transaction. If you group multiple SQL queries in a single transaction, database is hit only once.

It is used for ensuring that the database modifications are performed by the nested actions which can be either rolled back or committed

```
<%
Date DoB = new Date("2000/10/16");
int studentId = 151;
%>
<sql:transaction dataSource="${db}">
  <sql:update var="count">
    UPDATE Student SET First_Name = 'Suraj' WHERE Id = 150
  </sql:update>
  <sql:update var="count">
    UPDATE Student SET Last_Name= 'Saifi' WHERE Id = 153
  </sql:update>
  <sql:update var="count">
   INSERT INTO Student
   VALUES (154,'Supriya', 'Jaiswal', '1995/10/6');
  </sql:update>
</sql:transaction>
```

simple JSP example to understand the use of <sql:transaction> tag:

```
<%@ page import="java.io.*,java.util.*,java.sql.*"%>
<%@ page import="javax.servlet.http.*,javax.servlet.*" %>
<%@ page import="java.util.Date,java.text.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>

<html>
<head>
<title>sql:transaction Tag</title>
</head>
<body>

<sql:setDataSource var="db" driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/test"
    user="root"  password="1234"/>

<%
```

```
Date DoB = new Date("2000/10/16");
int studentId = 151;
%>

<sql:transaction dataSource="${db}">
  <sql:update var="count">
    UPDATE Student SET First_Name = 'Suraj' WHERE Id = 150
  </sql:update>
  <sql:update var="count">
    UPDATE Student SET Last_Name= 'Saifi' WHERE Id = 153
  </sql:update>
  <sql:update var="count">
   INSERT INTO Student
   VALUES (154,'Supriya', 'Jaiswal', '1995/10/6');
  </sql:update>
</sql:transaction>

<sql:query dataSource="${db}" var="rs">
  SELECT * from Student;
</sql:query>

<table border="2" width="100%">
<tr>
  <th>Emp ID</th>
  <th>First Name</th>
  <th>Last Name</th>
  <th>DoB</th>
</tr>
<c:forEach var="table" items="${rs.rows}">
<tr>
  <td><c:out value="${table.id}"/></td>
  <td><c:out value="${table.First_Name}"/></td>
  <td><c:out value="${table.Last_Name}"/></td>
  <td><c:out value="${table.dob}"/></td>
</tr>
</c:forEach>
</table>

</body>
</html>
```

Registration Form in JSP

For creating registration form, you must have a table in the database. You can write the database logic in JSP file, but separating it from the JSP page is better approach. Here, we are going to use DAO, Factory Method, DTO and Singletion design patterns. There are many files:

index.jsp for getting the values from the user

User.java, a bean class that have properties and setter and getter methods.

process.jsp, a jsp file that processes the request and calls the methods

Provider.java, an interface that contains many constants like DRIVER_CLASS, CONNECTION_URL, USERNAME and PASSWORD

ConnectionProvider.java, a class that returns an object of Connection. It uses the Singleton and factory method design pattern.

RegisterDao.java, a DAO class that is responsible to get access to the database

Example of Registration Form in JSP

In this example, we are using the Oracle10g database to connect with the database. Let's first cre
the table in the Oracle database:

```
CREATE TABLE  "USER432"
  (    "NAME" VARCHAR2(4000),
   "EMAIL" VARCHAR2(4000),
   "PASS" VARCHAR2(4000)
  )
/
```

index.jsp

We are having only three fields here, to make the concept clear and simplify the flow of the application. You can have other fields also like country, hobby etc. according to your requirement

```
<form action="process.jsp">
<input type="text" name="uname" value="Name..." onclick="this.value=''"/><br/>

<input type="text" name="uemail"  value="Email ID..." onclick="this.value=''"/><br
/>
<input type="password" name="upass"  value="Password..." onclick="this.value=''"
/><br/>
<input type="submit" value="register"/>
</form>
```

process.jsp

This jsp file contains all the incoming values to an object of bean class which is passed as an argument in the register method of the RegisterDao class.

```
<%@page import="bean.RegisterDao"%>
<jsp:useBean id="obj" class="bean.User"/>

<jsp:setProperty property="*" name="obj"/>

<%
int status=RegisterDao.register(obj);
if(status>0)
out.print("You are successfully registered");

%>
```

User.java

It is the bean class that have 3 properties uname, uemail and upass with its setter and getter metho

```
package bean;

public class User {
private String uname,upass,uemail;

public String getUname() {
    return uname;
}

public void setUname(String uname) {
    this.uname = uname;
}

public String getUpass() {
    return upass;
}

public void setUpass(String upass) {
    this.upass = upass;
}

public String getUemail() {
    return uemail;
}
public void setUemail(String uemail) {
    this.uemail = uemail;
}

}
```

Provider.java

This interface contains four constants that can vary from database to database.

```
package bean;

public interface Provider {
String DRIVER="oracle.jdbc.driver.OracleDriver";
String CONNECTION_URL="jdbc:oracle:thin:@localhost:1521:xe";
String USERNAME="system";
String PASSWORD="oracle";

}
```

ConnectionProvider.java

This class is responsible to return the object of Connection. Here, driver class is loaded only once and connection object gets memory only once.

```
package bean;
import java.sql.*;
import static bean.Provider.*;

public class ConnectionProvider {
private static Connection con=null;
static{
try{
Class.forName(DRIVER);
con=DriverManager.getConnection(CONNECTION_URL,USERNAME,PASSWORD);
}catch(Exception e){}
}

public static Connection getCon(){
    return con;
}

}
```

RegisterDao.java

This class inserts the values of the bean component into the database.

```
package bean;

import java.sql.*;

public class RegisterDao {

public static int register(User u){
int status=0;
try{
Connection con=ConnectionProvider.getCon();
PreparedStatement ps=con.prepareStatement("insert into user432 values(?,?,?)");
ps.setString(1,u.getUname());
ps.setString(2,u.getUemail());
ps.setString(3,u.getUpass());

status=ps.executeUpdate();
}catch(Exception e){}

return status;
}

}
```

Login and Logout Example in JSP

In this example of creating login form, we have used the DAO (Data Access Object), Factory meth and DTO (Data Transfer Object) design patterns. There are many files:

index.jsp it provides three links for login, logout and profile

login.jsp for getting the values from the user

loginprocess.jsp, a jsp file that processes the request and calls the methods.

LoginBean.java, a bean class that have properties and setter and getter methods.

Provider.java, an interface that contains many constants like DRIVER_CLASS, CONNECTION_U USERNAME and PASSWORD

ConnectionProvider.java, a class that is responsible to return the object of Connection. It uses
Singleton and factory method design pattern.

LoginDao.java, a DAO class that verifies the emailId and password from the database.

logout.jsp it invalidates the session.

profile.jsp it provides simple message if user is logged in, otherwise forwards the request to
login.jsp page.

In this example, we are using the Oracle10g database to match the emailId and password with the database. The table name is user432 which have many fields like name, email, pass etc. You may use this query to create the table:

```
CREATE TABLE  "USER432"
  (    "NAME" VARCHAR2(4000),
   "EMAIL" VARCHAR2(4000),
   "PASS" VARCHAR2(4000)
  )
/
```

index.jsp

It simply provides three links for login, logout and profile.

```
<a href="login.jsp">login</a>|
<a href="logout.jsp">logout</a>|
<a href="profile.jsp">profile</a>
```

login.jsp

This file creates a login form for two input fields name and password. It is the simple login form, you can change it for better look and feel. We are focusing on the concept only.

```
<%@ include file="index.jsp" %>
<hr/>

<h3>Login Form</h3>
<%
String profile_msg=(String)request.getAttribute("profile_msg");
if(profile_msg!=null){
out.print(profile_msg);
}
String login_msg=(String)request.getAttribute("login_msg");
if(login_msg!=null){
out.print(login_msg);
}
```

```
 %>
 <br/>
<form action="loginprocess.jsp" method="post">
Email:<input type="text" name="email"/><br/><br/>
Password:<input type="password" name="password"/><br/><br/>
<input type="submit" value="login"/>"
</form>
```

loginprocess.jsp

This jsp file contains all the incoming values to an object of bean class which is passed as an argument in the validate method of the LoginDao class. If emailid and password is correct, it displays a message you are successfully logged in! and maintains the session so that we may recognize the user.

```
<%@page import="bean.LoginDao"%>
<jsp:useBean id="obj" class="bean.LoginBean"/>

<jsp:setProperty property="*" name="obj"/>

<%
boolean status=LoginDao.validate(obj);
if(status){
out.println("You r successfully logged in");
session.setAttribute("session","TRUE");
}
else
{
out.print("Sorry, email or password error");
%>
<jsp:include page="index.jsp"></jsp:include>
<%
}
%>
```

LoginBean.java

It is the bean class that have 2 properties email and pass with its setter and getter methods.

```
package bean;

public class LoginBean {
private String email,pass;
public String getEmail() {
    return email;
}

public void setEmail(String email) {
    this.email = email;
}

public String getPass() {
    return pass;
}

public void setPass(String pass) {
    this.pass = pass;
}


}
```

Provider.java

This interface contains four constants that may differ from database to database.

```
package bean;

public interface Provider {
String DRIVER="oracle.jdbc.driver.OracleDriver";
String CONNECTION_URL="jdbc:oracle:thin:@localhost:1521:xe";
String USERNAME="system";
String PASSWORD="oracle";

}
```

ConnectionProvider.java

This class provides a factory method that returns the object of Connection. Here, driver class is loaded only once and connection object gets memory only once because it is static.

```
package bean;
import java.sql.*;
import static bean.Provider.*;

public class ConnectionProvider {
private static Connection con=null;
static{
```

```
try{
Class.forName(DRIVER);
con=DriverManager.getConnection(CONNECTION_URL,USERNAME,PASSWORD);
}catch(Exception e){}
}

public static Connection getCon(){
    return con;
}

}
```

LoginDao.java

This class varifies the emailid and password.

```
package bean;
import java.sql.*;
public class LoginDao {

public static boolean validate(LoginBean bean){
boolean status=false;
try{
Connection con=ConnectionProvider.getCon();

PreparedStatement ps=con.prepareStatement(
    "select * from user432 where email=? and pass=?");

ps.setString(1,bean.getEmail());
ps.setString(2, bean.getPass());

ResultSet rs=ps.executeQuery();
status=rs.next();

}catch(Exception e){}

return status;

}
}
```