**1JSTL(JSP Standard Tag Library)**

Supported with JSP1.2 and standardized with JSP 2.1

This is a specification from sun and is implemented by server vendors.

(Tomcat ,Websphere, weblogic etc)

sun people included the core functionality which is common to many web applications in JSTL API.

Programmers can use this predefined library without writing tags on his own.

(to some extent no need to go for custom tags we can use predefined custom tags ie JSTL)

**JSTL contains:**

**1)core library:**

It defines several standard actions to perform programming general stuff like implementing conditions and loops.

**2)XML library**

It defines several standard actions use ful for writing and formatting XML data.

**3) Formatting library:**

Defines serveral standard actions which can be used for formating numbers and dates in the process of I18N.

**4) SQL library:**

defines serveral standard actions which can be used for data base operations.

**5)Functional library:**

Defines several standard actions which can be used for manuplating collections and string objects

**Core Library:-**

JSTL core library deivided into the following four:

**1)General-purpose tags:-**

<c:out>

<c:set>

<c:remove>

<c:catch>

**2)conditional tags:-**

<c:if>

<c:choose>

<c:when>

<c:otherwise>

**3)Iteration tags:**

<c:forEach>

<c:forTokens>

**4) URL related tags**

<c:import>

<c:url>

<c:redirect>

<c:param>

**JSTL versions and:**

**Important note:**

There are three version of jstl

* JSTL 1.0(is supported by J2EE 1.3/JSP 1.2)
* JSTL 1.1(is supported by J2EE 1.4/JSP 2.0)
* JSTL 1.2(is supported by J2EE 5,JEE 6/JSP 2.1)

We must select correct JSTL version for our web application based on servlet contianer we are using (ie web server or application server).

->JSTL1.2 should be used for JSP 2.1 supporting web container (TOMCAT7)

->JSTL 1.1 should used for JSP 2.0 supporting web containers (TOMCAT5)

->while JSTL 1.0 should be used for containers supporting JSP 1.x(TOMCAT4)

By default JSTL functionality is not available to the jsp.

**taglib directvies used in jsp page as JSTL 1.2:**

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

**JSTL1.2 Core Lib**

If we are using JSTL 1.2 core lib then in our jsp file the following tab lib directive is compulsory:

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

or

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

This tag lib directive maps our core lib tags to tld file the tld file then maps java classes.

**General-purpose tags**

**1.<c:out>**

we can use this tag to write template data and expression to the jsp (just like out.println())

<c:out>

c -> prefix name

out-> tag name

**form1:**

<c:out value="ram" />

similar jsp code:

<%

out.println("ram");

%>

it prints ram to the browser/jsp.

**steps to create jsp application that uses JSTL**

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

**jstlApp1:-**

|  |
| --- |
| C:\TOMCAT7\WEBAPPS\jstlApp1  │ view.jsp  │  └───WEB-INF  │ web.xml  │  ├───classes  └───lib  Jstl-1.2.jar |

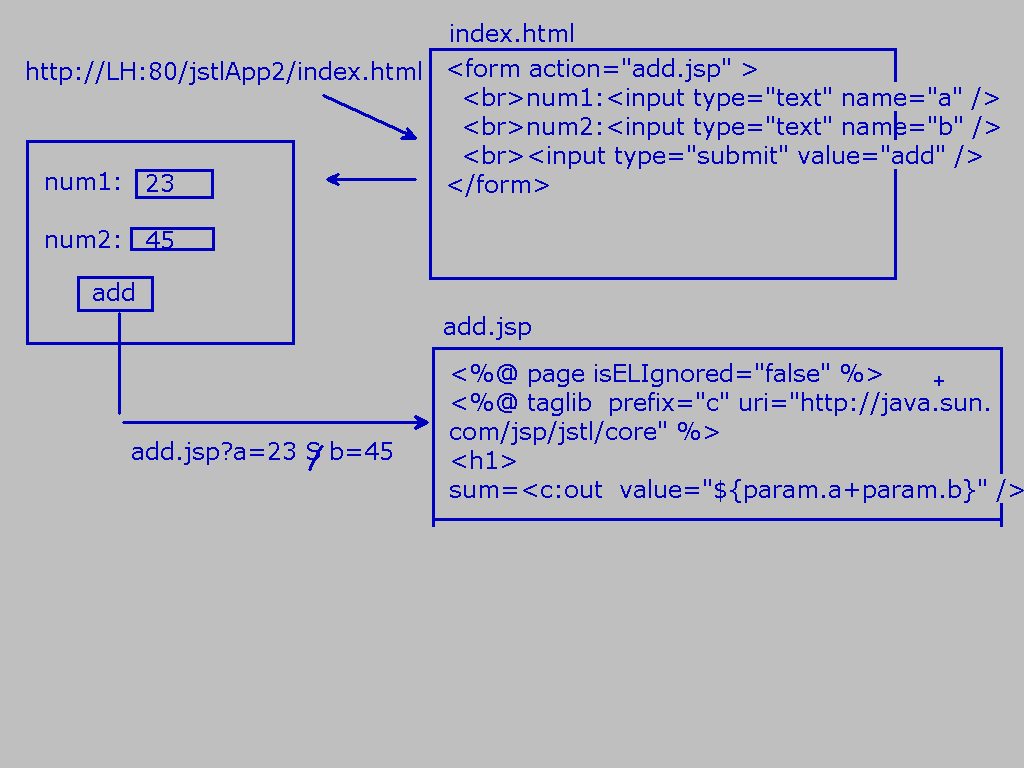
**view.jsp**

|  |
| --- |
| <%@ page isELIgnored="false" %>  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  <c:out value="ramu" /> |

**important point:**

Instead of hard coding with strings we can provide runtime expressions ie EL expressions.

so runtime expression values also can printed on the browser.

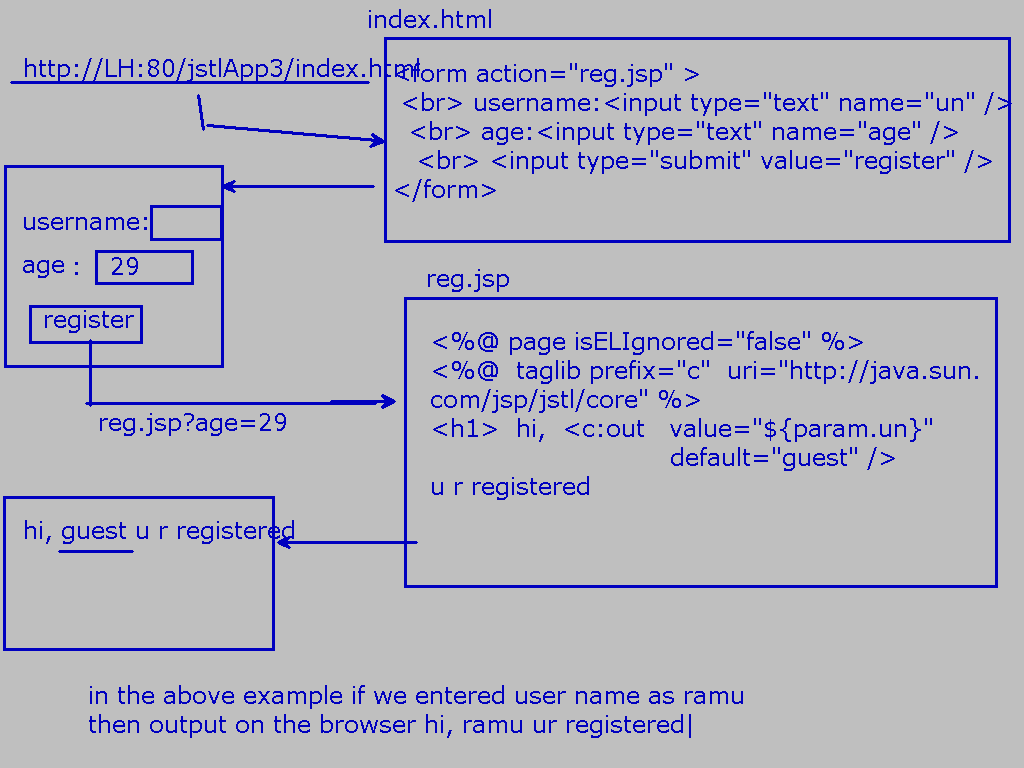


**form2:**

if the main value is not availbale or it evaluates to null. Then in this case we can provide default value by using default attribute.

Example:

<c:out value="${param.uid}" default="guest" />



<c:out> tag defines the following two attribures:

1) value

it is mandatory attribute and it is for providing value it can be literal or runtime expression

2) default:

It is optional attribute and it is used to provide default value

jsp engine consider default value if and only if main value evaluates to null

**<c:set>**

By using this tag we can set attributes in any scope and we can set the properties of bean or map object.

**form1:**

<c:set var="x" value="sathish" scope="session" />

we use above form1 To set an attribute "x" in session scope with value "sathish".

equal jsp code:

<%

session.setAttribute("x","sathish");

%>

here name and value are mandotory attributes.

scope is optional and default scope is page scope.

**examples:**

<c:set var="x" value="10" scope="session" />

<c:set var="y" value="20" scope="session" />

<c:set var="result" value="${x+y}" scope="request" />

<br>The result is:<c:out value="${result}" />

**equal jsp java code:**

<%

session.setAttribute("x",10);

session.setAttribute("y",20); request.setAttribute("result",(Integer)session.getAttribute("x")+(Integer)session.getAttribute("y"));

%>

<br>The result is:

<%

out.println(request.getAttribute("result"));

%>

**form2:**

we can use <c:set> to set properties of Map or Bean object

we can specify Map or bean object by using "target" attributes and property can be specified by using "property' attribute

**Inside servelt1**

Student c = new Student();

c.setName("sathish");

request.setAttribute("c",c);

- - -

rd = reqeust.getRD("one.jsp");

rd.farword(request,response);

**one.jsp**

<%@ page isELIgnored="false" %>

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

<c:out value="${requestScope.c.name}" />-->prints->sathish

<c:set target="${requestScope.c}" property="name" value="subbarao" />

<c:out vlaue="${requestScope.c.name}" />-->prints->subbarao

**attibutes of <c:set> are:**

1.var 2.value 3.scope 4.target 5.property

**<c:remove>**

To remove attributes in the specified scope we can use this tag

<c: remove var="x" scope="session" />

Equal java code is:

---------------------

session.removeAttribute(“x”);

**<c:catch>**

We can use this tag to catch an exception with in the jsp instead of forwardingw to error page.

generally exception handling code in java:

try{

ricky code

}

catch(){

handler code

}

note:

But in case of jsp jstl we have to place risky code as the body of <c:catch>.

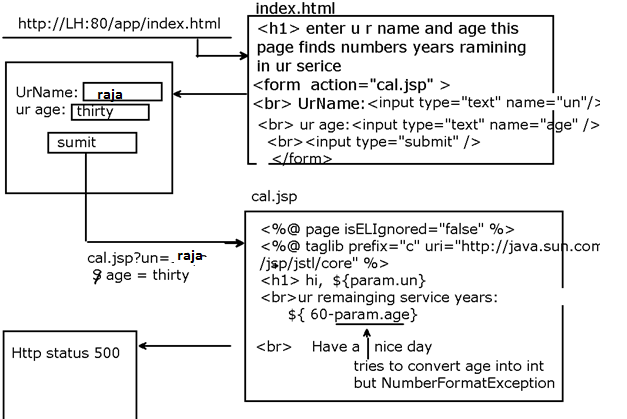
<c:catch>

risky code

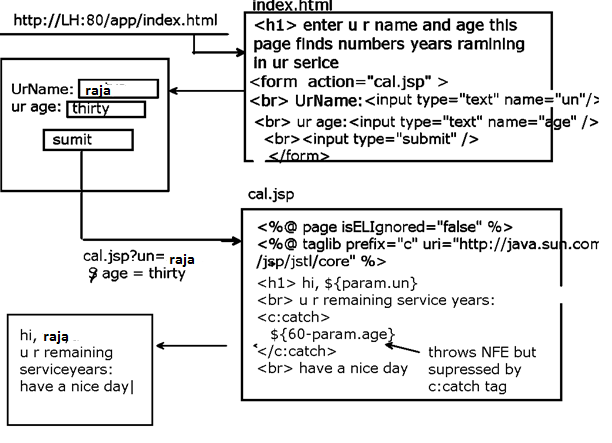
</c:catch>

if an exception raised in the risky code this tag supresses that exception and rest of the jsp will be executed normally.

if an exception raised we can hold it by using var attribute which is a page scoped attribute.

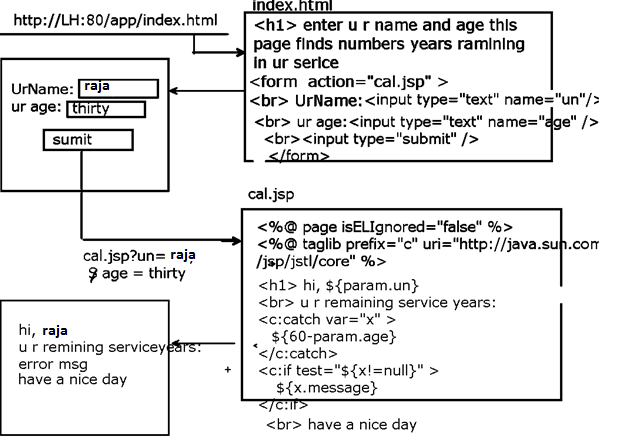


To avoid Http status 500 we use <c:catch> to supress the exception



But in this example exception is suppressed

But no error printed on the browser to print error message on the browser we use var attribute



**2)Conditional tags:**

**<c:if>**

we can use this tag to implement simple if statement.

There are two forms:

**1st form without body:**

<c:if test="test-condition" var="a" scope="session" />

In this form we use three attributes

**test->** contains test-condition to be executed

**var->**contain name of attribute that contains test result as value.

**scope->**specifies in which scope attribute name and value must be stored.

if scope is not specified default scope is page.

In this form test-condition is evaluated and its value is stored in the variable "a" and variable "a" is set as attribute in a scope.

In the rest of the jsp page whenever the same test condition is required we can use directly use var "a" value without evaluating test-condition once again.

In this case both "var" and "test" attributes are mandatory but "scope" attribute is optional.

**2nd form with body:**

<c:if test="test-condition" var="a" scope="session">

body

</c:if>

If test-condition is true, then body is executed else body is not executed but rest of the jsp will be executed.

In this case also we can store test-condition result in a variable "a".

Here "var" and "scope" attributes are optional but test attribute is compulsory.

eg:-

**Servlet1.java**

|  |
| --- |
| request.setAttribute("sub1",40);  request.setAttribute("sub2",50);  request.setAttribute("sub3",60);  . . .  rd=request.getRequestDispatcher("/result.jsp");  rd.forward(request,response); |

**result.jsp**

|  |
| --- |
| <%@ page isELIgnored="false"%>  <%@ taglib .....%>  <c:if test="${((requestScope.sub1+requestScope.sub2+requestScope.sub3)/3)>40}">  U r selected in the interview  </c:if> |

**2.<c:choose>,<c:when> and <c:otherwise>**

we can construct if else by using these three.

we can also construct switch case by using these three.

**constructing if else**

|  |
| --- |
| <c:choose>  <c:when test="condition" >  true part action  </c:when>  <c:otherwise>  false part action  </c:otherwise>  </c:choose> |

Same above example with true and false parts:

eg:-

**Servlet1.java**

|  |
| --- |
| request.setAttribute("sub1",40);  request.setAttribute("sub2",50);  request.setAttribute("sub3",60);  . . .  rd=request.getRequestDispatcher("/result.jsp");  rd.forward(request,response); |

**result.jsp**

|  |
| --- |
| <%@ page isELIgnored="false"%>  <%@ taglib .....%>  <c:choose >  <c:when test="${((requestScope.sub1+requestScope.sub2+requestScope.sub3)/3)>40}">  <h1>U r selected in the interview  </c:when>  <c:otherwise>  <h1> u r not selected in the interview  </c:otherwise>  </c:choose> |

**Constructing switch case:**

|  |
| --- |
| <c:choose>  <c:when test="condition1" >  true part action1  </c:when>  <c:when test="condition2" >  true part action2  </c:when>  <c:when test="condition3" >  true part action3  </c:when>  <c:otherwise>  false part action(default)  </c:otherwise>  </c:choose> |

**Note:**

Every <c:when> tag implicitly contains break so there is no chance of fallthrough inside switch.

**important points:**

<c:choose> should compulsory contain atleast one <c:when> tag.

But <c:otherwise> is optinal

<c:choose> and <c:otherwise> won't take any attributes

but<c:when> must take one mandatory attribute 'test'.

**page1.jsp**

|  |
| --- |
| <h1> select a number:<br>  <form action="page2.jsp" >  <select name="combobox" >  <option value="1">One</option>  <option value="2">Two</option>  <option value="3">Three</option>  <option value="4">Four</option>  <option value="5">Five</option>  </select>  <input type="submit" value="select" />  </form> |

**page2.jsp**

|  |
| --- |
| <%@ page isELIgnored="false" %>  <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>  <h1>  <c:set var="a" value="${param.combobox}" />  <c:choose>  <c:when test="${a==1}" >  U r number one hero  </c:when>  <c:when test="${a==2}" >  U r number two hero  </c:when>  <c:when test="${a==3}" >  U r number three hero  </c:when>  <c:otherwise>  u r vialan  </c:otherwise>  </c:choose> |

**Iteration tags:**

**<c:forEach>**

1st form:

<c:forEach begin="0" end="10" step="1">

<h1>This is JSTL!!!</h1>

</c:forEach>

then it is equivalent to

<%

for(int i=0;i<=10;i++)

out.println("<h1>This is JSTL!!!</h1>");

%>

Here the "begin" attribute specifies the index where the loop has to start.

"end" attribute specifies the index where the loop has to terminates.

This loop internally maintains a counter value which is incremented by "step" attribute value every time.

**Step attribute value is optional and default value is one.**

<c:forEach begin="0" end="10" step="3">

<h1>This is sathish JSTL!!!</h1>

</c:forEach>

4 times loop is executed

**2nd form of forEach:**

<c:forEach> internally maintains a counter variable which can be accessed by using "var" attribute.

This variable is a local variable of forEach

Hence we cannot access from outside forEach.

<c:forEach begin="0" end="10" step="1" var="i">

<h1>The current Coutnter=${i}</h1>

</c:forEach>

**3rd form of forEach:**

<c:forEach> for iterating Arrays and Collections using "items" attribute.

The "items" attribute should contain either collection or Array.

This action will iterate over each item in the collection until all elements completion.

We can represent current object by using var attribute.

**servlet1**

|  |
| --- |
| String a[]={"rama","sita","laxmana"};  session.setAttribute("a",a);  ArrayList<String> al = new ArryaList<String>();  al.add("sathish");  al.add("mani");  al.add("nani");  session.setAttribute("al",al);  RequestDispatcher rd = request.getRequestDispatcher("view.jsp");  rd.forward(request, response); |

**view.jsp**

|  |
| --- |
| <h1>String values:  <c:forEach items="${sessionScope.a}" var="e">  <br>${e}  </c:forEach>  <br>ArrayList values:  <c:forEach items="${sessionScope.al}" var="e">  <br>${e}  </c:forEach> |

output is:

String values:

rama

sita

laxmana

ArrayList values:

sathish

mani

nani

|  |
| --- |
| items attribute data type--->Var attribute datatype  1)primitive array int[] ---> Integer(Wrapper object)  2)String[] ---->String  3)Student[] ---->Student  4)Collection ----->Object  5)Map ----->Map.Entry  6)List of Strings With comma Separation --->String |

**4th form of forEach**

Getting request parameters by using forEach:

http://LH:80/app/view.jsp?uid=rama&pwd=sita

**view.jsp**

......

<c:forEach items="${param}" var="e">

${e.key} -> ${e.value}

</c:forEach>

**output:**

uid->rama pwd->sita

**5th form of forEach:**

**<c:forEach> with 'varStatus' attribute:**

This attribute describes the status of the iteration like current iteration number, is it first iteration or not etc

eg:

<c:forEach items="rama, sita, laxmana" varStatus="x" >

<br>is It first iteration:${x.first}

<br>The current object is:${x.current}

<br>is It Last iteration:${x.last}

<br>..............................

</c:forEach>

**output is:**

is It first iteration:true

The current object is:rama

is It Last iteration:false

...........................

is It first iteration:false

The current object is:sita

is It Last iteration:false

...........................

is It first iteration:false

The current object is:laxmana

is It Last iteration:true

**<c:forTokens>**

It is a specialized version of <c:forEach> to perform string tokenization based on some delimiter(separator) it behaves exactly same as java.util.StringTokenizer class.

<c:forTokens items="rama#sita#laxmana" delims="#" var="x">

The current token is:${x}

</c:forToken>

**The forTokens can accept the following attributes also:**

1) begin

2) end

3) step

4) varStatus

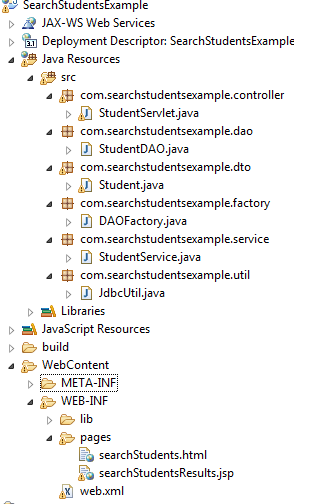
**Note:**

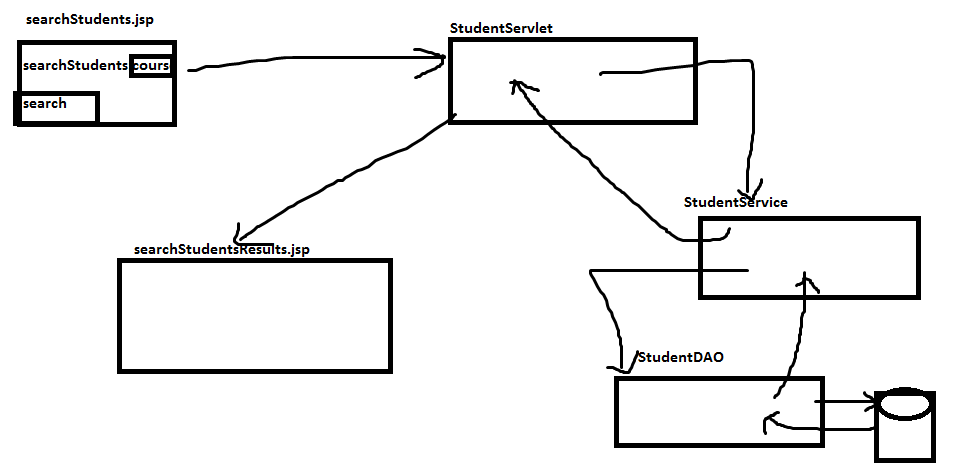
In the case of <c:forTokens> the items attribute must be a string.

But in the case of <c:forEach> items attribute can be collection, array, Map or String.

*Hence, <c:forTokens> is considered as specialized version of <c:forEach>*

**Example:-**

****

****

searchStudent.html

<html>

<body>

<form action=*"searchStudents"*>

Search Students :<input type=*"text"* name=*"course"* placeholder=*"Enter course"*>

<input type=*"submit"* value=*"search"*>

</form>

</body>

</html>

**web.xml**

<web-app >

<servlet>

<servlet-name>StudentServlet</servlet-name>

<servlet-class>com.searchstudentsexample.controller.StudentServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>StudentServlet</servlet-name>

<url-pattern>/searchStudents</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>/WEB-INF/pages/searchStudents.html</welcome-file>

</welcome-file-list>

</web-app>

**StudentServlet.java**

package com.searchstudentsexample.controller;

import java.io.IOException;

import java.util.List;

import javax.servlet.RequestDispatcher;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import com.searchstudentsexample.dto.Student;

import com.searchstudentsexample.service.StudentService;

public class StudentServlet extends HttpServlet {

private StudentService studentService;

public void init(){

studentService=new StudentService();

}

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String course=request.getParameter("course");

List<Student> list=studentService.searchStudents(course);

System.out.println(list.size());

request.setAttribute("list",list);

String target="/WEB-INF/pages/searchStudentsResults.jsp";

RequestDispatcher rd=request.getRequestDispatcher(target);

rd.forward(request,response);

}

}

**StudentService.java**

package com.searchstudentsexample.service;

import java.util.List;

import com.searchstudentsexample.dto.Student;

import com.searchstudentsexample.factory.DAOFactory;

public class StudentService {

public List<Student> searchStudents(String course) {

return DAOFactory.getStudentDAO().searchStudents(course);

}

}

**StudentDAO**

**package** com.searchstudentsexample.dao;

**import** java.sql.Connection;

**import** java.sql.PreparedStatement;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.util.ArrayList;

**import** java.util.List;

**import** com.searchstudentsexample.dto.Student;

**import** com.searchstudentsexample.util.JdbcUtil;

**public** **class** StudentDAO {

**public** List<Student> searchStudents(String course) {

List<Student> list=**new** ArrayList<Student>();

**try**{

Connection con=JdbcUtil.*getConnection*();

String sql="select studentId,name,email,mobile from Student where course like ?";

PreparedStatement pst=con.prepareStatement(sql);

pst.setString(1,course);

ResultSet rs=pst.executeQuery();

**while**(rs.next()){

Student student=**new** Student();

student.setStudentId(rs.getInt("studentId"));

student.setName(rs.getString("name"));

student.setMobile(rs.getString("mobile"));

student.setEmail(rs.getString("email"));

list.add(student);

}

}**catch**(SQLException se){

se.printStackTrace();

}

**return** list;

}

}

Student.java

**package** com.searchstudentsexample.dto;

**import** java.io.Serializable;

**public** **class** Student **implements** Serializable{

**private** **int** studentId;

**private** String name;

**private** String email,mobile;

**public** **int** getStudentId() {

**return** studentId;

}

**public** **void** setStudentId(**int** studentId) {

**this**.studentId = studentId;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getMobile() {

**return** mobile;

}

**public** **void** setMobile(String mobile) {

**this**.mobile = mobile;

}

}

**SearchStudentsResults.jsp**

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

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

<%@include file=*"searchStudents.html"* %>

<table border=*"1"*>

<tr>

<th>StudentId</th><th>Name</th><th>Email</th>

<th>Mobile</th></tr>

<c:choose>

<c:when test=*"*${list.size()>0}*"*>

<c:forEach items=*"*${list}*"* var=*"std"*>

<tr><td>

${std.studentId}</td><td>

${std.name}</td><td>

${std.email}</td><td>

${std.mobile}</td></tr>

</c:forEach>

</c:when>

<c:otherwise>

<tr><td colspan=*"4"* align=*"center"*>

<c:out value=*"No Records Are Found"* ></c:out>

</td></tr>

</c:otherwise>

</c:choose></tr></table>

**DAOFactory.java**

**package** com.searchstudentsexample.factory;

**import** com.searchstudentsexample.dao.StudentDAO;

**public** **class** DAOFactory {

**private** **static** StudentDAO *studentDAO*;

**static**{

*studentDAO*=**new** StudentDAO();

}

**public** **static** StudentDAO getStudentDAO(){

**return** *studentDAO*;

}

}

**JdbcUtil.java**

**package** com.searchstudentsexample.util;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**public** **class** JdbcUtil {

**static**{

**try**{

Class.*forName*("oracle.jdbc.driver.OracleDriver");

}**catch**(ClassNotFoundException ce){

ce.printStackTrace();

}

}

**public** **static** Connection getConnection() **throws** SQLException{

String url="jdbc:oracle:thin:@localhost:1521:XE";

String un="system";

String pass="manager";

Connection con=DriverManager.*getConnection*(url,un,pass);

**return** con;

}

}

**URL related tags:**

**1)<c:import> and <c:param>**

This action can be used for importing the response of other pages into the current page at request processing time.

(this is Dynamic include)

**form1:**

<c:import url="absoulte or relative url" />

example:

**first.jsp**

<%@ page isELIgnored="false" %>

<%@ taglib prefix="c" ...... %>

<c:import url="second.jsp" />

<h1>This is first.jsp</h1>

**second.jsp**

<h1>This is second.jsp

**Execution instruction:**

give request to first.jsp

then u will get response

http://localhost:80/App/first.jsp

This is second.jsp

This is first.jsp

**form2:**

We can import the reources(html or jsps ) of other application also.

ie cross-context import is possible

<c:import url="/page1.jsp" context="/JSTLApp2" />

**Important note:**

Most of the web servers do not provide support FOR cross-context import

**form3:**

We can store the result of imported page into a variable specified by var attribute.

In the rest of the page where ever that result is required we can directly use that variable without performing import once again.

<c:import url="second.jsp" var="x" scope="session" />

${x}<br>

${x}<br>

${x}<br>

**form4:**

While performing import we can send html input form parameters(request parameters) also to target jsp for this we can use <c:param>tag.

These parameters are available as input form parameters in target jsp.

**first.jsp**

<%@ page isELIgnored="false" %>

<%@ taglib prefix="c" ... %>

<h1>welcome to 1st jsp it is importing response from second.jsp</h1>

<c:import url="second.jsp">

<c:param name="x" value="sathish" />

</c:import>

**second.jsp**

<h1> parameter value in second.jsp:${param.x}

**2.<c:redirect>**

This action can be used to redirect the request to another page.

This is exactly same as <jsp:forward>

**form1:**

<c:redirect url="second.jsp" />

**form2:**

<c:redirect url="/second.jsp" context="/App3" />

Cross context redirect is not supported by most of the webservers.

**form3:**

While performing redirection we can pass parameters also to the target resource for this we have to use <c:param> tag.

**first.jsp**

<%@ page isELIgnored="false" %>

<%@ taglib prefix="c" %>

<h1>hello this is first.jsp redirect to second.jsp<br>

<c:redirect url="second.jsp" >

<c:param name="x" value="sathish" />

</c:redirect>

**second.jsp**

<br> This is inside second.jsp and parameter given by 1st jsp is ${param.x}

**3.<c:url>**

We can use this standard action to rewrite urls to append session information and form parameters.

**form1:**

http://lh:80/App/first.jsp

**first.jsp**

|  |
| --- |
| <%@ page isELIgnored="false" %>  <%@ taglib prefix="c" .....%>  <c:url value="second.jsp" var="x" scope="request" />  <a href="${x}">clickhere</a> |

then value in x is ->*"/App/second.jsp"*

**second.jsp**

|  |
| --- |
| <h1> This is sathish jstl |

**form2:**

**first.jsp**

|  |
| --- |
| <%@ page isELIgnored="false" %>  <%@ taglib prefix="c" ....%>  <c:url value="second.jsp" var="x" >  <c:param name="course1" value="corejava" />  <c:param name="course2" value="advjava" />  </c:url>  <h1>The encoded url is:${x}  <br><a href="${x}">click here to see more</a> |

**the above anchor tag is same as**

<a href="/second.jsp?jsessionid=......&courrse1=corejava&course2=advjava">click here to see more</a>

**second.jsp**

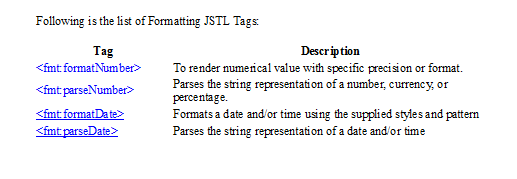
<h1>${param.course1} and${param.course2} are by sathishSir

## Formatting tags:

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

<%@ taglib prefix="fmt"

uri="http://java.sun.com/jsp/jstl/fmt" %>



We use <fmt:formatDate> tag to format the date or time information using provided styles and pattern.

The following JSP code displays the date and time information in various formats:

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

    pageEncoding="ISO-8859-1"%>

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

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

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

    "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>&lt;fmt:formatDate&gt; Demo</title>

</head>

<body>

<h1>&lt;fmt:formatDate&gt; Demo</h1>

<c:set var="today" value="<%=new java.util.Date()%>" />

<p>Time: <strong><fmt:formatDate type="time" value="${today}" /></strong></p>

<p>Date: <strong><fmt:formatDate type="date" value="${today}" /></strong></p>

<p>Date & Time: <strong><fmt:formatDate type="both" value="${today}" /></strong></p>

<p>Date & Time Short:

<strong>

    <fmt:formatDate type="both" dateStyle="short" timeStyle="short" value="${today}" />

</strong>

</p>

<p>Date & Time Medium:

<strong>

    <fmt:formatDate type="both" dateStyle="medium" timeStyle="medium" value="${today}" />

</strong></p>

<p>Date & Time Long:

<strong>

    <fmt:formatDate type="both" dateStyle="long" timeStyle="long" value="${today}" />

</strong></p>

<p>Date (yyyy-MM-dd):

<strong>

    <fmt:formatDate pattern="yyyy-MM-dd" value="${today}" />

</strong></p>

</body>

</html>

**The <fmt:parseDate> tag parses the string representation of date or time.**

# Syntax

<fmt:parseDate

   value="<string>"

   type="<string>"

   dateStyle="<string>"

   timeStyle="<string>"

   pattern="<string>"

   timeZone="<string>"

   parseLocale="<string>"

The following JSP code parses a string into date and displays its date value.

\* <html>

  <head>

    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

    <title>&lt;fmt:parseDate&gt; Demo</title>

  </head>

  <body>

    <h1>&lt;fmt:parseDate&gt; Demo</h1>

     <c:set var="currentDate" value="28-12-2012" />

     <fmt:parseDate value="${currentDate}" var="parsedCurrentDate" pattern="dd-MM-yyyy" />

     <p>Current date after parsing: <c:out value="${parsedCurrentDate}" /></p>

  </body>

</html>

**The <fmt:formatNumber> is used to format number values as number, currency or percentages.**

# Syntax

<fmt:formatNumber

  value="<string>"

  type="<string>"

  pattern="<string>"

  currencyCode="<string>"

  currencySymbol="<string>"

  groupingUsed="<string>"

  maxIntegerDigits="<string>"

The below JSP code takes account balances and displays it using various formats.

The Account Balance can be displayed using various ways:

<body>

 <c:set var="accountBalance" value="9500.60" />

      <p>as it is: <strong><fmt:formatNumber value="${accountBalance}" type="currency"/></strong></p>

      <p>max. integer digits 3:

      <strong>

          <fmt:formatNumber type="number" maxIntegerDigits="3" value="${accountBalance}" />

      </strong></p>

      <p>max. fraction digits 1:

      <strong>

          <fmt:formatNumber type="number" maxFractionDigits="3" value="${accountBalance}" />

      </strong></p>

      <p>no grouping:

      <strong>

          <fmt:formatNumber type="number" groupingUsed="false" value="${accountBalance}" />

      </strong></p>

      <p>percent with max. integer digits 3:

      <strong>

          <fmt:formatNumber type="percent" maxIntegerDigits="3" value="${accountBalance}" />

      </strong></p>

      <p>percent with min. fraction digits 10:

      <strong>

          <fmt:formatNumber type="percent" minFractionDigits="10" value="${accountBalance}" />

      </strong></p>

      <p>pattern ###.###E0:

      <strong>

          <fmt:formatNumber type="number" pattern="###.###E0" value="${accountBalance}" />

      </strong></p>

      <p>Account Balance in USA :

      <fmt:setLocale value="en\_US"/>

      <strong><fmt:formatNumber value="${accountBalance}" type="currency"/></p></strong>

       <p>Account Balance in Canada :

       <fmt:setLocale value="fr\_CA"/>

       <strong><fmt:formatNumber value="${accountBalance}" type="currency"/></p></strong>

   </body>

**Sample Invoice Page with JSTL:**

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

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

**<%@taglib prefix="c"**

**uri="http://java.sun.com/jsp/jstl/core" %>**

**<html>**

**<div align="center">**

**${sessionScope.medicalStore.medicalStoreName}**

**</div>**

**<div style="font-size:12px">**

**PhoneNumbers : ${sessionScope.medicalStore.phoneNumber1},${sessionScope.medicalStore.phoneNumber2}**

**<br/>**

**EmailId : ${sessionScope.medicalStore.emailId}**

**</div>**

**<div style="font-size:12px" align="right">**

**Address : H\_NO: ${sessionScope.medicalStore.h\_no}, Street : ${sessionScope.medicalStore.street}**

**<br/>**

**City : ${sessionScope.medicalStore.city}**

**</div>**

**<div style="font-size:12px">**

**Patient Name : ${sessionScope.customer.customerName}**

**</div>**

**<hr/>**

**<%**

**int sno=1;**

**%>**

**<table>**

**<tr><th>SNO</th><th>MedicineName</th>**

**<th>MedicineType</th>**

**<th>ExpDate</th><th>Rate</th>**

**<th>Quantity</th><th>DosageUnits</th><th>Amount</tr>**

**<c:forEach items="${sessionScope.setOfMedicines}" var="medicine">**

**<tr>**

**<td>**

**<%=sno++ %>**

**</td>**

**<td>**

**${medicine.medicineName}**

**</td>**

**<td>**

**${medicine.medicineType}**

**</td>**

**<td>**

**<fmt:formatDate pattern="MM-YYYY" value="${medicine.expDate}"/>**

**</td>**

**<td>**

**${medicine.rate}**

**</td>**

**<td>**

**${medicine.stock}**

**</td>**

**<td>**

**${medicine.dosageUnits}**

**</td>**

**<td>**

**<fmt:formatNumber value="${medicine.amount}" maxFractionDigits="2"/>**

**</td>**

**</tr>**

**<c:set var="amount"**

**value="${medicine.amount}"/>**

**<c:set var="totalAmount"**

**value="${amount+totalAmount}"/>**

**</c:forEach>**

**</table><br/>**

**<div style="font-size:12px;">**

**Total Amount : <fmt:formatNumber maxFractionDigits="2" value="${totalAmount}"/>**

**</div>**

**<div align="center">**

**<input type="button" value="Save & Print" onclick="save()"/>**

**<input type="button" value="Cancel" onclick="cancel()"/>**

**</div>**

**</html>**

**<script>**

**function cancel(){**

**$.ajax({**

**type:"get",**

**url:"cancel",**

**success:function(result){**

**window.location.reload(true);**

**}**

**});**

**}**

**function save(){**

**$.ajax({**

**type:"get",**

**url:"saveInvoiceDetails",**

**success:function(result){**

**if(result.length>2){**

**var response=JSON.parse(result);**

**if(response.status==1){**

**var billNum=response.data;**

**printInvoiceDetails(billNum);**

**}**

**else{**

**alert(response.message);**

**}**

**}**

**else{**

**alert("server problem");**

**}**

**}**

**});**

**}**

**function printInvoiceDetails(billNum){**

**$.ajax({**

**type:"get",**

**url:"printInvoiceDetails",**

**data:"billNum="+billNum,**

**async: false,**

**success : function(result) {**

**var display\_setting =**

**"location=no,directories=yes,menubar=no,";**

**display\_setting += "scrollbars=yes,width=750, height=600, left=100, top=25";**

**var document\_print = window.open("", "", display\_setting);**

**document\_print.document.open();**

**document\_print.document.write('<html><body onload="window.print();self.close()">'+result+'</body></html>');**

**document\_print.document.close();**

**window.location.reload(true);**

**return false;**

**}**

**});**

**}**

**</script>**

**-----------------------------------------------------------------------------------------------**

**JDBC-Transacations**

If your JDBC Connection is in *auto-commit* mode, which is by default, then every SQL statement is committed to the database upon its completion. That may be fine for simple applications, but there are three reasons why you may want to turn off auto-commit and manage your own transactions:

* To increase performance
* To maintain the integrity of business processes
* To use distributed transactions

Transactions enable you to control if, and when, changes are applied to the database. It treats a single SQL statement or a group of SQL statements as one logical unit, and if any statement fails, the whole transaction fails.

To enable manual- transaction support instead of the *auto-commit* mode that the JDBC driver uses by default, use the Connection object's **setAutoCommit()** method. If you pass a boolean false to setAutoCommit( ), you turn off auto-commit. You can pass a boolean true to turn it back on again.

**For example**, if you have a Connection object named conn, code the following to turn off auto-commit:

conn.setAutoCommit(false);

## Commit & Rollback

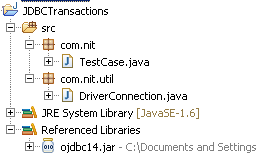
Once you are done with your changes and you want to commit the changes then call **commit()** method on connection object as follows:

conn.commit( );

Otherwise, to roll back updates to the database made using the Connection named conn, use the following code:

conn.rollback( );

**The following example illustrates the use of a commit and rollback object:**

****

**DriverConnection.java**

**package** com.nit.util;

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

**public** **class** DriverConnection {

**static**{

**try**{

Class.*forName*("oracle.jdbc.driver.OracleDriver");

}**catch** (Exception e) {

e.printStackTrace();

}

}

**public** **static** Connection getConn()**throws** SQLException{

**return** DriverManager.*getConnection*("jdbc:oracle:thin:@nit-11:1521:xe","system" , "manager");

}

}

**TestCase.java**

**package** com.nit;

**import** java.sql.Connection;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** com.nit.util.DriverConnection;

**public** **class** TestCase {

**public** **static** **void** main(String[] args) **throws** SQLException{

Connection con=**null**;

**try**{

con=DriverConnection.*getConn*();

**con.setAutoCommit(false);**

Statement st=con.createStatement();

**int** count=st.executeUpdate("INSERT INTO STUDENT VALUES(89,’NITSTUDENT2',85)");

System.*out*.println("Number of Rows are Inserted:"+i);

If(count>0){  
con.commit();

}

}

**catch**(SQLException e) {

con.rollback();

System.*out*.println(e);

}

}

}

**OUTPUT**

Number of Rows are Inserted:1

Check The Data in DataBase

SQL> SELECT \*FROM STUDENT;

SNO SNAME SMARKS

------- -------------- ----------

25 NIT 89

95 NITSTUDENT 99

**package** com.nit;

**import** java.sql.Connection;

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** com.nit.util.DriverConnection;

**public** **class** TestCase {

**public** **static** **void** main(String[] args) **throws** SQLException{

Connection con=**null**;

**try**{

con=DriverConnection.*getConn*();

**con.setAutoCommit(true);**

Statement st=con.createStatement();

**int** i=st.executeUpdate("INSERT INTO STUDENT VALUES(89,'NITSTUDENT2',85)");

System.*out*.println("Number of Rows are Inserted:"+i);

}

**catch**(SQLException e) {

//con.rollback();

System.*out*.println(e);

}

}

}

**OUTPUT**

Number of Rows are Inserted:1

Check The Data in DataBase

SQL> SELECT \*FROM STUDENT;

SNO SNAME SMARKS

---------- ------------------- ----------

25 NIT 89

95 NITSTUDENT 99

**89 NITSTUDENT2 85**