JSP



**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 thanservlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tag etc.

Advantage of JSP over Servlet

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

**1) Extension to Servlet**

JSP technology is the extension to servlet technology.We can use all the features of servletin 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 withpresentation 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 servletcode needs to be updated andrecompiled if we have to change the look and feel of the application.

**4) Less code than Servlet**

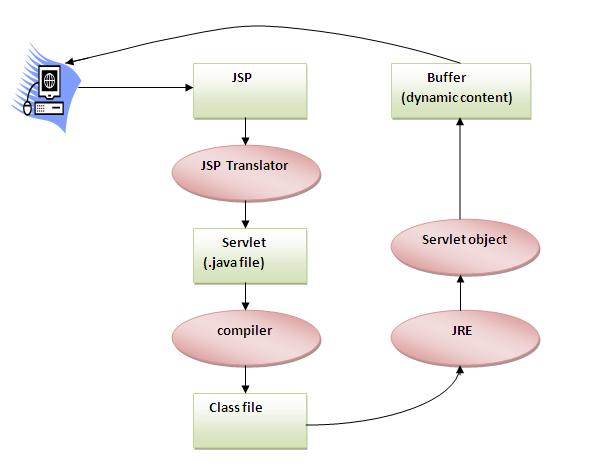
In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc.that reduces thecode. Moreover, we can use EL, implicit objects etc.

Life cycle of a JSP Page

The JSP pages follows these phases:

* Translation of JSP Page
* Compilation of JSP Page
* Classloading (class file is loaded by the classloader)
* Instantiation (Object of the Generated Servlet is created).
* Initialization ( jspInit() method is invoked by the container).
* Reqeust processing ( \_jspService() method is invoked by the container).
* Destroy ( jspDestroy() method is invoked by the container).

**Note: jspInit(), \_jspService() and jspDestroy() are the life cycle methods of JSP.**



As depicted in the above diagram, JSP page is translated into servlet by the help of JSP translator. The JSP translator is a part of webserver that is responsible to translate the JSP page into servlet. Afterthat Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happens in servlet is performed on JSP later like initialization, committing response to the browser and destroy.

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 save 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, here we are using the scriptlet tag to put java code in the JSP page. We will learn scriptlet tag later.

1. <html>
2. <body>
3. <% out.print(2\*5); %>
4. </body>
5. </html>

It will print **10** on the browser.

How to run a simple JSP Page ?

Follow the following steps to execute this JSP page:

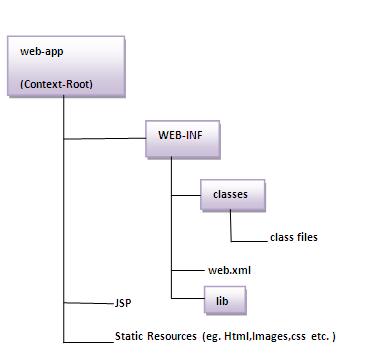
* Start the server
* put the jsp file in a folder and deploy on the server
* visit the browser by the url http://localhost:portno/contextRoot/jspfile e.g. http://localhost:8888/myapplication/index.jsp

Do I need to follow directory structure to run a simple JSP ?

No, there is no need of directory structure if you don't have class files or tld files. For example, put jsp files in a folder directly and deploy that folder.It will be running fine.But if you are using bean class, Servlet or tld file then directory structure is required.

Directory structure of JSP

The directory structure of JSP page is same as servlet. We contains the jsp page outside the WEB-INF folder or in any directory.



# 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:

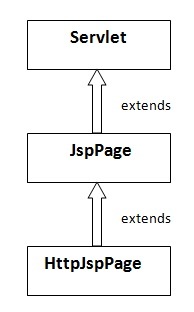
1. JspPage
2. HttpJspPage

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 JSPbefore 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 pagecomes to the container. It is used to process the request. The underscore \_ signifies that you cannot override this method.

We will learn all other classes and interfaces later.

# JSP Scriptlet tag (Scripting elements)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

## 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

### JSP scriptlet tag

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

1. <%  java source code %>

### Example of JSP scriptlet tag

In this example, we are displaying a welcome message.

1. **<html>**
2. **<body>**
3. **<**% out.print("welcome to jsp"); %**>**
4. **</body>**
5. **</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*

1. **<html>**
2. **<body>**
3. **<form** action="welcome.jsp"**>**
4. **<input** type="text" name="uname"**>**
5. **<input** type="submit" value="go"**><br/>**
6. **</form>**
7. **</body>**
8. **</html>**

*File: welcome.jsp*

1. <html>
2. <body>
3. <%
4. String name=request.getParameter("uname");
5. out.print("welcome "+name);
6. %>
7. </form>
8. </body>
9. </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.

1. **<html>**
2. **<body>**
3. **<**%= "welcome to jsp" %**>**
4. **</body>**
5. **</html>**

**Note: Do not end your statement with semicolon in case of expression tag.**

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

*index.jsp*

1. **<html>**
2. **<body>**
3. Current Time: **<**%= java.util.Calendar.getInstance().getTime() %**>**
4. **</body>**
5. **</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*

1. **<html>**
2. **<body>**
3. **<form** action="welcome.jsp"**>**
4. **<input** type="text" name="uname"**><br/>**
5. **<input** type="submit" value="go"**>**
6. **</form>**
7. **</body>**
8. **</html>**

*File: welcome.jsp*

1. **<html>**
2. **<body>**
3. **<**%= "Welcome "+request.getParameter("uname") %**>**
4. **</body>**
5. **</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:

1. **<**%!  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**

1. **<html>**
2. **<body>**
3. **<**%! int data=50; %**>**
4. **<**%= "Value of the variable is:"+data %**>**
5. **</body>**
6. **</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**

1. **<html>**
2. **<body>**
3. **<**%!
4. int cube(int n){
5. return n\*n\*n\*;
6. }
7. %**>**
8. **<**%= "Cube of 3 is:"+cube(3) %**>**
9. **</body>**
10. **</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.

A list of the 9 implicit objects is given below:

|  |  |
| --- | --- |
| **Object** | **Type** |
| Out | JspWriter |
| Request | HttpServletRequest |
| Response | HttpServletResponse |
| Config | ServletConfig |
| Application | ServletContext |
| Session | HttpSession |
| pageContext | PageContext |
| Page | Object |
| Exception | Throwable |

**1) JSP out implicit object**

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

1. PrintWriter out=response.getWriter();

But in JSP, you don't need to write this code.

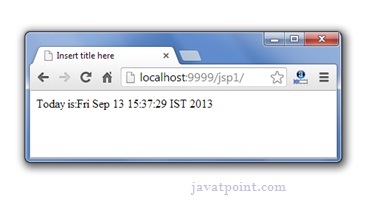
Example of out implicit object

In this example we are simply displaying date and time.

**index.jsp**

1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>

**Output**



Upcoming topics in JSP implicit Objects

JSP request implicit object

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsprequest 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.

Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

Example of JSP request implicit object

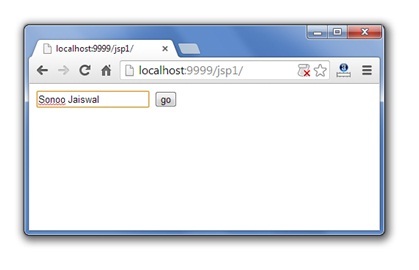
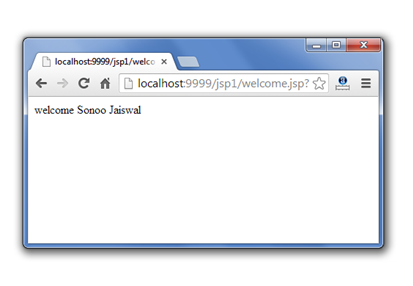
**index.html**

1. **<form** action="welcome.jsp"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

**welcome.jsp**

1. <%
2. String name=request.getParameter("uname");
3. out.print("welcome "+name);
4. %>

**Output**

3) 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.

Let's see the example of response implicit object where we are redirecting the response to the Google.

Example of response implicit object

**index.html**

1. **<form** action="welcome.jsp"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

**welcome.jsp**

1. **<**%
2. response.sendRedirect("http://www.google.com");
3. %**>**

**Output**

4) JSP config implicit object

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

Generally, it is used to get initialization parameter from the web.xml file.

Example of config implicit object:

**index.html**

1. **<form** action="welcome"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

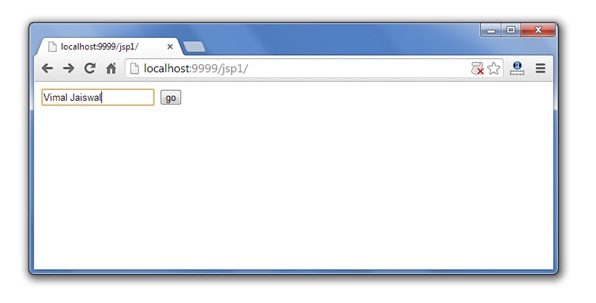
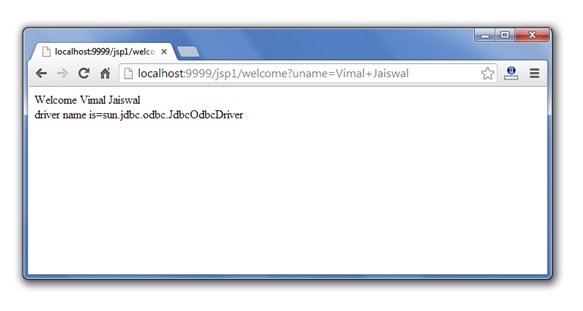
**web.xml file**

1. **<web-app>**
3. **<servlet>**
4. **<servlet-name>**sonoojaiswal**</servlet-name>**
5. **<jsp-file>**/welcome.jsp**</jsp-file>**
7. **<init-param>**
8. **<param-name>**dname**</param-name>**
9. **<param-value>**sun.jdbc.odbc.JdbcOdbcDriver**</param-value>**
10. **</init-param>**
12. **</servlet>**
14. **<servlet-mapping>**
15. **<servlet-name>**sonoojaiswal**</servlet-name>**
16. **<url-pattern>**/welcome**</url-pattern>**
17. **</servlet-mapping>**
19. **</web-app>**

**welcome.jsp**

1. **<**%
2. out.print("Welcome "+request.getParameter("uname"));
4. String driver=config.getInitParameter("dname");
5. out.print("driver name is="+driver);
6. %**>**

**Output**

5) 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.

This initialization parameter can be used by all jsp pages.

Example of application implicit object:

**index.html**

1. **<form** action="welcome"**>**
2. **<input** type="text" name="uname"**>**
3. **<input** type="submit" value="go"**><br/>**
4. **</form>**

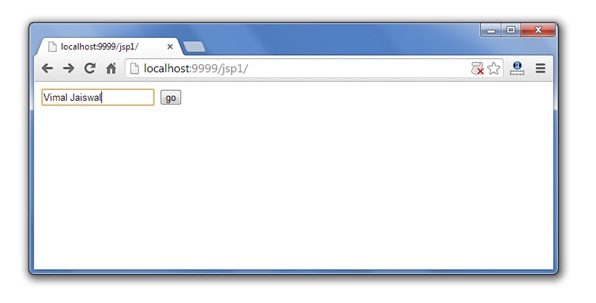
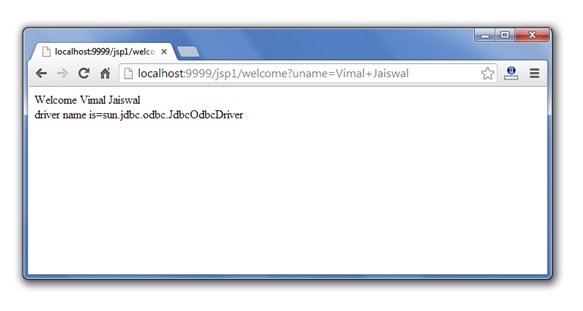
**web.xml file**

1. **<web-app>**
3. **<servlet>**
4. **<servlet-name>**sonoojaiswal**</servlet-name>**
5. **<jsp-file>**/welcome.jsp**</jsp-file>**
6. **</servlet>**
8. **<servlet-mapping>**
9. **<servlet-name>**sonoojaiswal**</servlet-name>**
10. **<url-pattern>**/welcome**</url-pattern>**
11. **</servlet-mapping>**
13. **<context-param>**
14. **<param-name>**dname**</param-name>**
15. **<param-value>**sun.jdbc.odbc.JdbcOdbcDriver**</param-value>**
16. **</context-param>**
18. **</web-app>**

**welcome.jsp**

1. **<**%
3. out.print("Welcome "+request.getParameter("uname"));
5. String driver=application.getInitParameter("dname");
6. out.print("driver name is="+driver);
8. %**>**

**Output**

6) 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**

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

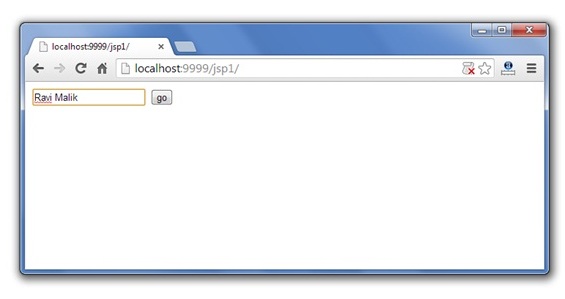
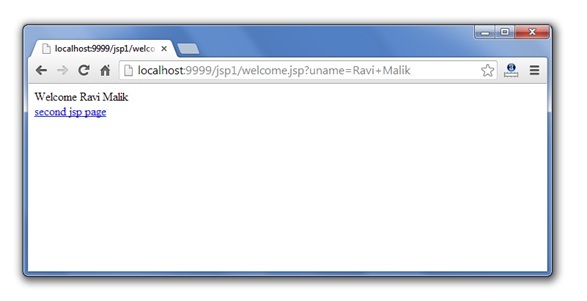
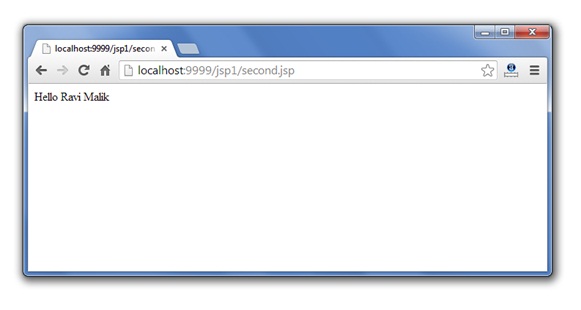
**welcome.jsp**

1. <html>
2. <body>
3. <%
5. String name=request.getParameter("uname");
6. out.print("Welcome "+name);
8. session.setAttribute("user",name);
10. <a href="second.jsp">second jsp page</a>
12. %>
13. </body>
14. </html>

**second.jsp**

1. <html>
2. <body>
3. <%
5. String name=(String)session.getAttribute("user");
6. out.print("Hello "+name);
8. %>
9. </body>
10. </html>

**Output**

7) 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**

1. <html>
2. <body>
3. <form action="welcome.jsp">
4. <input type="text" name="uname">
5. <input type="submit" value="go"><br/>
6. </form>
7. </body>
8. </html>

**welcome.jsp**

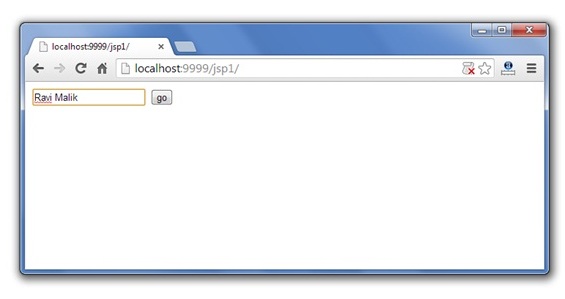
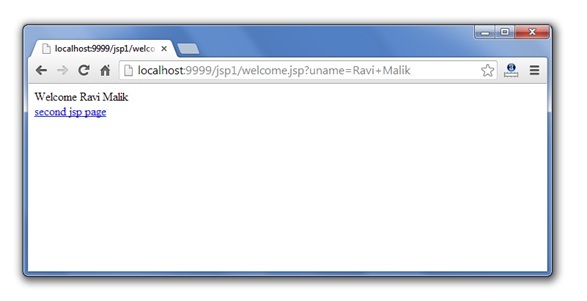
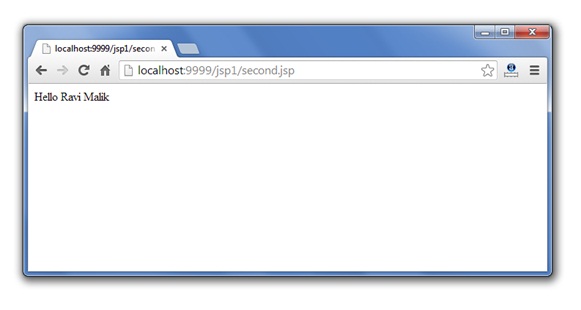
1. <html>
2. <body>
3. <%
5. String name=request.getParameter("uname");
6. out.print("Welcome "+name);
8. pageContext.setAttribute("user",name,PageContext.SESSION\_SCOPE);

11. %>
12. <a href="second.jsp">second jsp page</a>
13. </body>
14. </html>

**second.jsp**

1. <html>
2. <body>
3. <%
5. String name=(String)pageContext.getAttribute("user",PageContext.SESSION\_SCOPE);
6. out.print("Hello "+name);
8. %>
9. </body>
10. </html>

**Output**

8) 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"); %> |

9) 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. Let's see a simple example: |

Example of exception implicit object:

**error.jsp**

1. <%@ page isErrorPage="true" %>
2. <html>
3. <body>
5. Sorry following exception occured:<%= exception %>
7. </body>
8. </html>

To get the full example, click here [full example of exception handling in jsp](http://www.javatpoint.com/exception-handling-in-jsp). But, it will be better to learn it after the JSP Directives.

JSP directives

1. [JSP directives](http://www.javatpoint.com/jsp-page-directive)
   1. [page directive](http://www.javatpoint.com/jsp-page-directive#page)
   2. [Attributes of page directive](http://www.javatpoint.com/jsp-page-directive#pageattr)

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

There are three types of directives:

* page directive
* include directive
* taglib directive

**Syntax of JSP Directive**

1. <%@ directive attribute="value" %>

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

**Syntax of JSP page directive**

1. <%@ page 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 to import keyword in java class or interface. |

Example of import attribute

1. <html>
2. <body>
4. <%@ page **import**="java.util.Date" %>
5. Today is: <%= **new** Date() %>
7. </body>
8. </html>

2)contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The default value is "text/html;charset=ISO-8859-1".

Example of contentType attribute

1. <html>
2. <body>
4. <%@ page contentType=application/msword %>
5. Today is: <%= **new** java.util.Date() %>
7. </body>
8. </html>

3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

4)info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

Example of info attribute

1. <html>
2. <body>
4. <%@ page info="composed by Sonoo Jaiswal" %>
5. Today is: <%= **new** java.util.Date() %>
7. </body>
8. </html>

The web container will create a method getServletInfo() in the resulting servlet.For example:

1. **public** String getServletInfo() {
2. **return** "composed by Sonoo Jaiswal";
3. }

5)buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

Example of buffer attribute

1. <html>
2. <body>
4. <%@ page buffer="16kb" %>
5. Today is: <%= **new** java.util.Date() %>
7. </body>
8. </html>

6)language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

7)isELIgnored

|  |
| --- |
| We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is false i.e. Expression Language is enabled by default. We see Expression Language later. |

1. <%@ page isELIgnored="true" %>//Now EL will be ignored

8)isThreadSafe

|  |
| --- |
| Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it.If you make the value of isThreadSafe attribute like: |

<%@ page isThreadSafe="false" %>

The web container in such a case, will generate the servlet as:

1. **public** **class** SimplePage\_jsp **extends** HttpJspBase
2. **implements** SingleThreadModel{
3. .......
4. }

9)errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

Example of errorPage attribute

1. //index.jsp
2. <html>
3. <body>
5. <%@ page errorPage="myerrorpage.jsp" %>
7. <%= 100/0 %>
9. </body>
10. </html>

10)isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

**Note: The exception object can only be used in the error page.**

Example of isErrorPage attribute

1. //myerrorpage.jsp
2. <html>
3. <body>
5. <%@ page isErrorPage="true" %>
7. Sorry an exception occured!<br/>
8. The exception is: <%= exception %>
10. </body>
11. </html>

Jsp Include Directive

1. [Include directive](http://www.javatpoint.com/jsp-include-directive)
2. [Advantage of Include directive](http://www.javatpoint.com/jsp-include-directive#includeadv)
3. [Example of include directive](http://www.javatpoint.com/jsp-include-directive#includeex)

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

Advantage of Include directive

Code Reusability

**Syntax of include directive**

1. <%@ include file="resourceName" %>

Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an header.html file.

1. <html>
2. <body>
4. <%@ include file="header.html" %>
6. Today is: <%= java.util.Calendar.getInstance().getTime() %>
8. </body>
9. </html>

**Note: The include directive includes the original content, so the actual page size grows at runtime.**

JSP Taglib directive

1. [JSP Taglib directive](http://www.javatpoint.com/jsp-taglib-directive)
2. [Example of JSP Taglib directive](http://www.javatpoint.com/jsp-taglib-directive#taglibex)

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. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

**Syntax JSP Taglib directive**

1. <%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>

Example of JSP Taglib directive

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

1. <html>
2. <body>
4. <%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
6. <mytag:currentDate/>
8. </body>
9. </html>

Exception Handling in JSP

1. [Exception Handling in JSP](http://www.javatpoint.com/exception-handling-in-jsp)
2. [Example of exception handling in jsp by the elements of page directive](http://www.javatpoint.com/exception-handling-in-jsp#jspexcepex1)
3. [Example of exception handling in jsp by specifying the error-page element in web.xml file](http://www.javatpoint.com/exception-handling-in-jsp#jspexcepex2)

The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception any time in your web application. So handling exceptions is a safer side for the web developer. In 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

Example of exception handling in jsp by the elements of page directive

In this case, you must define and create a page to handle the exceptions, as in the error.jsp page. The pages where may occur exception, define the errorPage attribute of page directive, as in the process.jsp page.

There are 3 files:

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

**index.jsp**

1. <form action="process.jsp">
2. No1:<input type="text" name="n1" /><br/><br/>
3. No1:<input type="text" name="n2" /><br/><br/>
4. <input type="submit" value="divide"/>
5. </form>

**process.jsp**

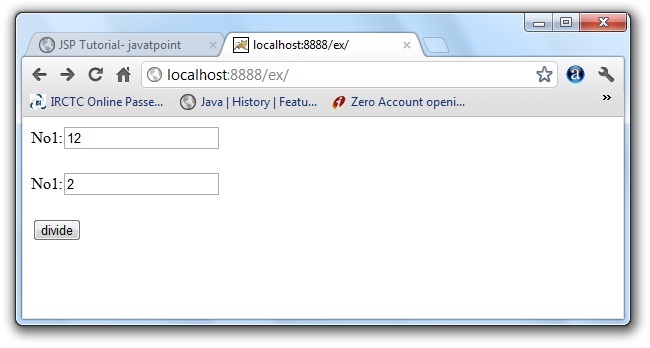
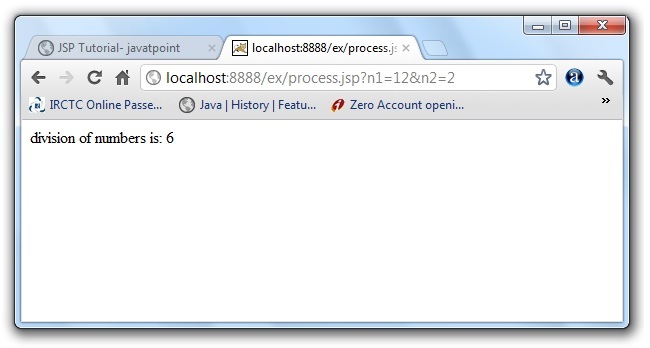
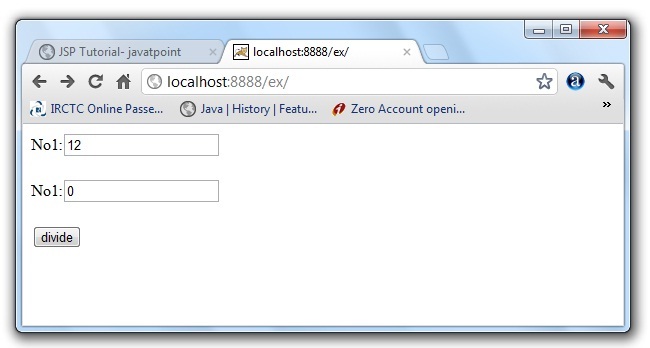
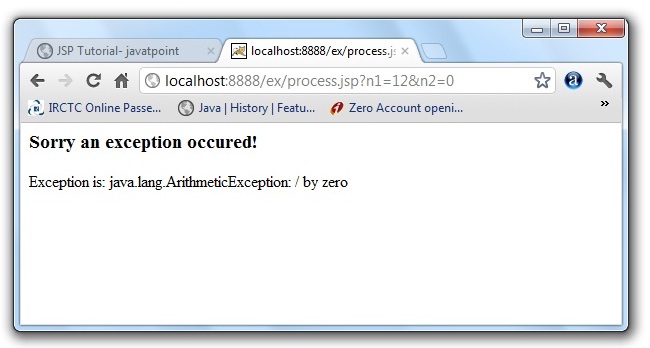
1. <%@ page errorPage="error.jsp" %>
2. <%
4. String num1=request.getParameter("n1");
5. String num2=request.getParameter("n2");
7. **int** a=Integer.parseInt(num1);
8. **int** b=Integer.parseInt(num2);
9. **int** c=a/b;
10. out.print("division of numbers is: "+c);
12. %>

**error.jsp**

1. <%@ page isErrorPage="true" %>
3. <h3>Sorry an exception occured!</h3>
5. Exception is: <%= exception %>

[download this example](http://www.javatpoint.com/src/jsp/ex1.zip)

Output of this example:

Example of exception handling in jsp by specifying the error-page element in web.xml file

This approach is better because you don't need to specify the errorPage attribute in each jsp page. Specifying the single entry in the web.xml file will handle the exception. In this case, either specify exception-type or error-code with the location element. If you want to handle all the exception, you will have to specify the java.lang.Exception in the exception-type element. Let's see the simple example:

There are 4 files:

* web.xml file for specifying the error-page element
* index.jsp for input values
* process.jsp for dividing the two numbers and displaying the result
* error.jsp for displaying the exception

**1) web.xml file if you want to handle any exception**

1. <web-app>
3. <error-page>
4. <exception-type>java.lang.Exception</exception-type>
5. <location>/error.jsp</location>
6. </error-page>
8. </web-app>

This approach is better if you want to handle any exception. If you know any specific error code and you want to handle that exception, specify the error-code element instead of exception-type as given below:

**1) web.xml file if you want to handle the exception for a specific error code**

1. <web-app>
3. <error-page>
4. <error-code>500</error-code>
5. <location>/error.jsp</location>
6. </error-page>
8. </web-app>

**2) index.jsp file is same as in the above example**

**3) process.jsp**

|  |
| --- |
| Now, you don't need to specify the errorPage attribute of page directive in the jsp page. |

1. <%@ page errorPage="error.jsp" %>
2. <%
4. String num1=request.getParameter("n1");
5. String num2=request.getParameter("n2");
7. **int** a=Integer.parseInt(num1);
8. **int** b=Integer.parseInt(num2);
9. **int** c=a/b;
10. out.print("division of numbers is: "+c);
12. %>

**4) error.jsp file is same as in the above example**

JSP Action Tags

1. [JSP Action Tags](http://www.javatpoint.com/jsp-action-tags-forward-action)
2. [jsp:forward action tag](http://www.javatpoint.com/jsp-action-tags-forward-action#forward)
3. [Example of jsp:forward action tag without parameter](http://www.javatpoint.com/jsp-action-tags-forward-action#forwardex1)
4. [Example of jsp:forward action tag with parameter](http://www.javatpoint.com/jsp-action-tags-forward-action#forwardex2)

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. |

The jsp:useBean, jsp:setProperty and jsp:getProperty tags are used for bean development. So we will see these tags in bean developement.

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**

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

**Syntax of jsp:forward action tag with parameter**

1. <jsp:forward page="relativeURL | <%= expression %>">
2. <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
3. </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**

1. <html>
2. <body>
3. <h2>**this** is index page</h2>
5. <jsp:forward page="printdate.jsp" />
6. </body>
7. </html>

**printdate.jsp**

1. <html>
2. <body>
3. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
4. </body>
5. </html>

[download this example](http://www.javatpoint.com/src/jsp/forwardaction.zip)

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**

1. <html>
2. <body>
3. <h2>**this** is index page</h2>
5. <jsp:forward page="printdate.jsp" >
6. <jsp:param name="name" value="javatpoint.com" />
7. </jsp:forward>
9. </body>
10. </html>

**printdate.jsp**

1. <html>
2. <body>
4. <% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
5. <%= request.getParameter("name") %>
7. </body>
8. </html>

jsp:include action tag

1. [jsp:include action tag](http://www.javatpoint.com/jsp-include-action)
2. [Syntax of jsp:include action tag](http://www.javatpoint.com/jsp-include-action#includesyn)
3. [Example of jsp:include action tag without parameter](http://www.javatpoint.com/jsp-include-action#include)

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**

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

**Syntax of jsp:include action tag with parameter**

1. <jsp:include page="relativeURL | <%= expression %>">
2. <jsp:param name="parametername" value="parametervalue | <%=expression%>" />
3. </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*

1. <h2>**this** is index page</h2>
3. <jsp:include page="printdate.jsp" />
5. <h2>end section of index page</h2>

*File: printdate.jsp*

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

[download this example](http://www.javatpoint.com/src/jsp/includeaction.zip)

# Java Bean

A Java Bean is a java class that should follow 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 Java Bean?

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

### Simple example of java bean class

1. //Employee.java
3. **package** mypack;
4. **public** **class** Employee **implements** java.io.Serializable{
5. **private** **int** id;
6. **private** String name;
8. **public** Employee(){}
10. **public** **void** setId(**int** id){**this**.id=id;}
12. **public** **int** getId(){**return** id;}
14. **public** **void** setName(String name){**this**.name=name;}
16. **public** String getName(){**return** name;}
18. }

### How to access the java bean class?

|  |
| --- |
| To access the java bean class, we should use getter and setter methods. |

1. **package** mypack;
2. **public** **class** Test{
3. **public** **static** **void** main(String args[]){
5. Employee e=**new** Employee();//object is created
7. e.setName("Arjun");//setting value to the object
9. System.out.println(e.getName());
11. }}

#### Note: There are two ways to provide values to the object, one way is by constructor and second is by setter method.

# jsp:useBean action tag

1. [jsp:useBean action tag](http://www.javatpoint.com/jsp-useBean-action)
2. [Syntax of jsp:useBean action tag](http://www.javatpoint.com/jsp-useBean-action#jspusesyn)
3. [Attributes and Usage of jsp:useBean action tag](http://www.javatpoint.com/jsp-useBean-action#jspuseattr)
4. [Simple example of jsp:useBean action tag](http://www.javatpoint.com/jsp-useBean-action#jspuseex1)

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

1. <jsp:useBean id= "instanceName" scope= "page | request | session | application"
2. **class**= "packageName.className" type= "packageName.className"
3. beanName="packageName.className | <%= expression >" >
4. </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

In this example, we are simply invoking the method of the Bean class.

**For the example of setProperty, getProperty and useBean tags, visit next page.**

**Calculator.java (a simple Bean class)**

1. **package** com.javatpoint;
2. **public** **class** Calculator{
4. **public** **int** cube(**int** n){**return** n\*n\*n;}
6. }

**index.jsp file**

1. <jsp:useBean id="obj" **class**="com.javatpoint.Calculator"/>
3. <%
4. **int** m=obj.cube(5);
5. out.print("cube of 5 is "+m);
6. %>

[download this example](http://www.javatpoint.com/src/jsp/usebeanaction.zip)

# jsp:setProperty and jsp:getProperty action tags

1. [jsp:setProperty and jsp:getProperty action tags](http://www.javatpoint.com/jsp-setProperty-and-jsp-getProperty-action-tag)
2. [Syntax of jsp:setProperty action tag](http://www.javatpoint.com/jsp-setProperty-and-jsp-getProperty-action-tag#jspsetsyn)
3. [Example of jsp:setProperty](http://www.javatpoint.com/jsp-setProperty-and-jsp-getProperty-action-tag#jspsetex1)
4. [jsp:getProperty action tag](http://www.javatpoint.com/jsp-setProperty-and-jsp-getProperty-action-tag#jspget)
5. [Syntax of jsp:getProperty action tag](http://www.javatpoint.com/jsp-setProperty-and-jsp-getProperty-action-tag#jspgetsyn)
6. [Example of jsp:getProperty action tag](http://www.javatpoint.com/jsp-setProperty-and-jsp-getProperty-action-tag#jspgetex1)
7. [Example of bean development in JSP](http://www.javatpoint.com/jsp-setProperty-and-jsp-getProperty-action-tag#jspbeanex1)

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

1. <jsp:setProperty name="instanceOfBean" property= "\*"   |
2. property="propertyName" param="parameterName"  |
3. property="propertyName" value="{ string | <%= expression %>}"
4. />

### 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

1. <form action="process.jsp" method="post">
2. Name:<input type="text" name="name"><br>
3. Password:<input type="password" name="password"><br>
4. Email:<input type="text" name="email"><br>
5. <input type="submit" value="register">
6. </form>

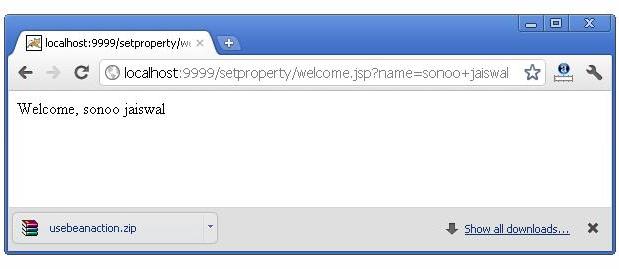
#### process.jsp

1. <jsp:useBean id="u" **class**="org.sssit.User"></jsp:useBean>
2. <jsp:setProperty property="\*" name="u"/>
4. Record:<br>
5. <jsp:getProperty property="name" name="u"/><br>
6. <jsp:getProperty property="password" name="u"/><br>
7. <jsp:getProperty property="email" name="u" /><br>

#### User.java

1. **package** org.sssit;
3. **public** **class** User {
4. **private** String name,password,email;
5. //setters and getters
6. }

[download this example](http://www.javatpoint.com/src/jsp/setproperty.zip)

#### Reusing Bean in Multiple Jsp Pages

Let's see the simple example, that prints the data of bean object in two jsp pages.

#### index.jsp

Same as above.

#### User.java

Same as above.

#### process.jsp

1. <jsp:useBean id="u" **class**="org.sssit.User" scope="session"></jsp:useBean>
2. <jsp:setProperty property="\*" name="u"/>
4. Record:<br>
5. <jsp:getProperty property="name" name="u"/><br>
6. <jsp:getProperty property="password" name="u"/><br>
7. <jsp:getProperty property="email" name="u" /><br>
9. <a href="second.jsp">Visit Page</a>

#### second.jsp

1. <jsp:useBean id="u" **class**="org.sssit.User" scope="session"></jsp:useBean>
2. Record:<br>
3. <jsp:getProperty property="name" name="u"/><br>
4. <jsp:getProperty property="password" name="u"/><br>
5. <jsp:getProperty property="email" name="u" /><br>

#### Using variable value in setProperty tag

In some case, you may get some value from the database, that is to be set in the bean object, in such case, you need to use expression tag. For example:

#### process.jsp

1. <jsp:useBean id="u" **class**="org.sssit.User"></jsp:useBean>
2. <%
3. String name="arjun";
4. %>
5. <jsp:setProperty property="name" name="u" value="<%=name %>"/>
7. Record:<br>

# Displaying applet in JSP (jsp:plugin action tag)

1. [Displaying applet in JSP](http://www.javatpoint.com/displaying-applet-in-jsp)
2. [Syntax of jsp:plugin action tag](http://www.javatpoint.com/displaying-applet-in-jsp#jsppluginsyn)
3. [Example of displaying applet in JSP](http://www.javatpoint.com/displaying-applet-in-jsp#jsppluginex1)

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

## Syntax of jsp:plugin action tag

1. **<jsp:plugin** type= "applet | bean" code= "nameOfClassFile"
2. codebase= "directoryNameOfClassFile"
3. **</jsp:plugin>**

### Example of displaying applet in JSP

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

### index.jsp

1. **<html>**
2. **<head>**
3. **<meta** http-equiv="Content-Type" content="text/html; charset=UTF-8"**>**
4. **<title>**Mouse Drag**</title>**
5. **</head>**
6. **<body** bgcolor="khaki"**>**
7. **<h1>**Mouse Drag Example**</h1>**
9. **<jsp:plugin** align="middle" height="500" width="500"
10. type="applet"  code="MouseDrag.class" name="clock" codebase="."**/>**
12. **</body>**
13. **</html>**

Expression Language (EL) in JSP

1. [Expression Language (EL) in JSP](http://www.javatpoint.com/EL-expression-in-jsp)
2. [Implicit Objects in Expression Language](http://www.javatpoint.com/EL-expression-in-jsp#elimplicit)
3. [Simple example of Expression Language that prints the name of the user](http://www.javatpoint.com/EL-expression-in-jsp#elex1)
4. [Example of Expression Language that prints the value set in the session scope](http://www.javatpoint.com/EL-expression-in-jsp#elex2)
5. [Precedence of Operators in EL](http://www.javatpoint.com/EL-expression-in-jsp#elprecedence)
6. [Reserve words in EL](http://www.javatpoint.com/EL-expression-in-jsp#elwords)

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.

**Syntax for Expression Language (EL)**

1. ${ expression }

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. |

Simple example of Expression Language that prints the name of the user

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**

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

**process.jsp**

1. Welcome, ${ param.name }

[download this example](http://www.javatpoint.com/src/jsp/el1.zip)

Example of Expression Language that prints the value set in the session scope

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

**index.jsp**

1. <h3>welcome to index page</h3>
2. <%
3. session.setAttribute("user","sonoo");
4. %>
6. <a href="process.jsp">visit</a>

**process.jsp**

1. Value is ${ sessionScope.user }

[download this example](http://www.javatpoint.com/src/jsp/el2.zip)

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

1. [MVC in JSP](http://www.javatpoint.com/MVC-in-jsp)
2. [Example of following MVC in JSP](http://www.javatpoint.com/MVC-in-jsp#mvcex)

**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

#### If you new to MVC, please visit [Model1 vs Model2](http://www.javatpoint.com/model-1-and-model-2-mvc-architecture) first.

MVC Example in JSP

In this example, we are using servlet as a controller, jsp as a view component, Java Bean class as a model.

In this example, we have created 5 pages:

* **index.jsp** a page that gets input from the user.
* **ControllerServlet.java** a servlet that acts as a controller.
* **login-success.jsp** and **login-error.jsp** files acts as view components.
* **web.xml** file for mapping the servlet.

*File: index.jsp*

1. <form action="ControllerServlet" method="post">
2. Name:<input type="text" name="name"><br>
3. Password:<input type="password" name="password"><br>
4. <input type="submit" value="login">
5. </form>

*File: ControllerServlet*

1. **package** com.javatpoint;
2. **import** java.io.IOException;
3. **import** java.io.PrintWriter;
4. **import** javax.servlet.RequestDispatcher;
5. **import** javax.servlet.ServletException;
6. **import** javax.servlet.http.HttpServlet;
7. **import** javax.servlet.http.HttpServletRequest;
8. **import** javax.servlet.http.HttpServletResponse;
9. **public** **class** ControllerServlet **extends** HttpServlet {
10. **protected** **void** doPost(HttpServletRequest request, HttpServletResponse response)
11. **throws** ServletException, IOException {
12. response.setContentType("text/html");
13. PrintWriter out=response.getWriter();
15. String name=request.getParameter("name");
16. String password=request.getParameter("password");
18. LoginBean bean=**new** LoginBean();
19. bean.setName(name);
20. bean.setPassword(password);
21. request.setAttribute("bean",bean);
23. **boolean** status=bean.validate();
25. **if**(status){
26. RequestDispatcher rd=request.getRequestDispatcher("login-success.jsp");
27. rd.forward(request, response);
28. }
29. **else**{
30. RequestDispatcher rd=request.getRequestDispatcher("login-error.jsp");
31. rd.forward(request, response);
32. }
34. }
36. @Override
37. **protected** **void** doGet(HttpServletRequest req, HttpServletResponse resp)
38. **throws** ServletException, IOException {
39. doPost(req, resp);
40. }
41. }

*File: LoginBean.java*

1. **package** com.javatpoint;
2. **public** **class** LoginBean {
3. **private** String name,password;
5. **public** String getName() {
6. **return** name;
7. }
8. **public** **void** setName(String name) {
9. **this**.name = name;
10. }
11. **public** String getPassword() {
12. **return** password;
13. }
14. **public** **void** setPassword(String password) {
15. **this**.password = password;
16. }
17. **public** **boolean** validate(){
18. **if**(password.equals("admin")){
19. **return** **true**;
20. }
21. **else**{
22. **return** **false**;
23. }
24. }
25. }

*File: login-success.jsp*

1. <%@page **import**="com.javatpoint.LoginBean"%>
3. <p>You are successfully logged in!</p>
4. <%
5. LoginBean bean=(LoginBean)request.getAttribute("bean");
6. out.print("Welcome, "+bean.getName());
7. %>

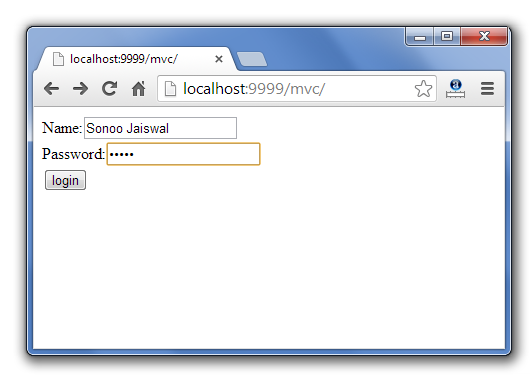
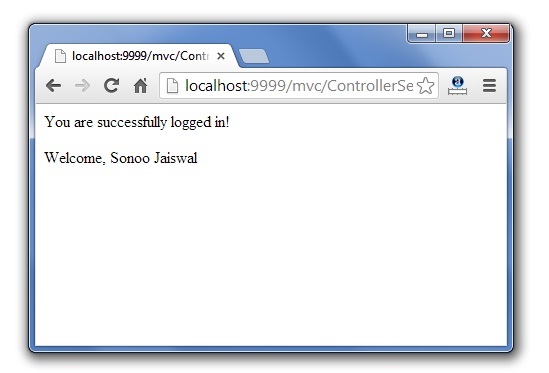
*File: login-error.jsp*

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

*File: web.xml*

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<web-app** xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"
4. xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"
5. id="WebApp\_ID" version="3.0"**>**
7. **<servlet>**
8. **<servlet-name>**s1**</servlet-name>**
9. **<servlet-class>**com.javatpoint.ControllerServlet**</servlet-class>**
10. **</servlet>**
11. **<servlet-mapping>**
12. **<servlet-name>**s1**</servlet-name>**
13. **<url-pattern>**/ControllerServlet**</url-pattern>**
14. **</servlet-mapping>**
15. **</web-app>**

[download this example (developed using eclipse IDE)](http://www.javatpoint.com/src/jsp/mvceclipse.zip)

** **

# Next TopCustom Tags in JSP

1. [Custom Tags in JSP](http://www.javatpoint.com/custom-tags)
2. [Advantages of Custom Tags](http://www.javatpoint.com/custom-tags)
3. [Syntax to use custom tag](http://www.javatpoint.com/custom-tags#syn)
4. [JSP Custom Tag API](http://www.javatpoint.com/custom-tags#api)
   1. [JspTag interface](http://www.javatpoint.com/custom-tags#JspTag)
   2. [Tag interface](http://www.javatpoint.com/custom-tags#Tag)
   3. [IteratorTag interface](http://www.javatpoint.com/custom-tags#IteratorTag)
   4. [TagSupport class](http://www.javatpoint.com/custom-tags#TagSupport)

**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.

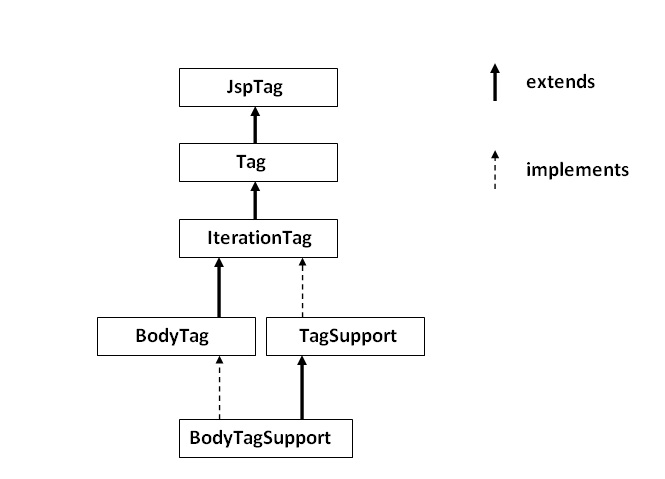
**Syntax to use custom tag**

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

1. **<prefix:tagname** attr1=value1....attrn=valuen **/>**
2. **<prefix:tagname** attr1=value1....attrn=valuen **>**
3. body code
4. **</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](http://www.javatpoint.com/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](http://www.javatpoint.com/attributes-in-jsp-custom-tag)

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

[Iteration using Custom Tag](http://www.javatpoint.com/Iteration-using-jsp-custom-tag)

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

[Custom URI in Custom Tag](http://www.javatpoint.com/custom-uri-in-jsp-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

1. [Example of JSP Custom Tag](http://www.javatpoint.com/example-of-jsp-custom-tag)
   1. [Create the Tag handler class](http://www.javatpoint.com/example-of-jsp-custom-tag#step1)
   2. [Create the TLD file](http://www.javatpoint.com/example-of-jsp-custom-tag#step2)
   3. [Create the JSP file](http://www.javatpoint.com/example-of-jsp-custom-tag#step3)

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**

**Understanding flow of custom tag in jsp**

**1) Create the Tag handler class**

To create the Tag Handler, we are inheriting the **TagSupport class** and overriding its method **doStartTag()**.To write data for the jsp, we need to use the **JspWriter class**.

The **PageContext** class provides **getOut()** method that returns the instance of JspWriter class. TagSupport class provides instance of pageContext bydefault.

*File: MyTagHandler.java*

1. **package** com.javatpoint.sonoo;
2. **import** java.util.Calendar;
3. **import** javax.servlet.jsp.JspException;
4. **import** javax.servlet.jsp.JspWriter;
5. **import** javax.servlet.jsp.tagext.TagSupport;
6. **public** **class** MyTagHandler **extends** TagSupport{
8. **public** **int** doStartTag() **throws** JspException {
9. JspWriter out=pageContext.getOut();//returns the instance of JspWriter
10. **try**{
11. out.print(Calendar.getInstance().getTime());//printing date and time using JspWriter
12. }**catch**(Exception e){System.out.println(e);}
13. **return** SKIP\_BODY;//will not evaluate the body content of the tag
14. }
15. }

**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*

1. **<?xml** version="1.0" encoding="ISO-8859-1" **?>**
2. <!DOCTYPE taglib
3. PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
4. "http://java.sun.com/j2ee/dtd/web-jsptaglibrary\_1\_2.dtd"**>**
6. **<taglib>**
8. **<tlib-version>**1.0**</tlib-version>**
9. **<jsp-version>**1.2**</jsp-version>**
10. **<short-name>**simple**</short-name>**
11. **<uri>**http://tomcat.apache.org/example-taglib**</uri>**
13. **<tag>**
14. **<name>**today**</name>**
15. **<tag-class>**com.javatpoint.sonoo.MyTagHandler**</tag-class>**
16. **</tag>**
17. **</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*

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

[download this example](http://www.javatpoint.com/src/jsp/cu1.zip)

**Output**

Attributes in JSP Custom Tag

1. [Attributes in JSP Custom Tag](http://www.javatpoint.com/attributes-in-jsp-custom-tag)
2. [Example to use attribute in JSP Custom Tag](http://www.javatpoint.com/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:

1. **<m:cube** number="4"**></m:cube>**

Here **m** is the prefix, **cube** is the tag name and **number** is the attribute.

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**

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

**CubeNumber.java**

1. **package** com.javatpoint.taghandler;
2. **import** javax.servlet.jsp.JspException;
3. **import** javax.servlet.jsp.JspWriter;
4. **import** javax.servlet.jsp.tagext.TagSupport;
6. **public** **class** CubeNumber **extends** TagSupport{
7. **private** **int** number;
9. **public** **void** setNumber(**int** number) {
10. **this**.number = number;
11. }
13. **public** **int** doStartTag() **throws** JspException {
14. JspWriter out=pageContext.getOut();
15. **try**{
16. out.print(number\*number\*number);
17. }**catch**(Exception e){e.printStackTrace();}
19. **return** SKIP\_BODY;
20. }
21. }

**mytags.tld**

1. **<?xml** version="1.0" encoding="ISO-8859-1" **?>**
2. <!DOCTYPE taglib
3. PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
4. "http://java.sun.com/j2ee/dtd/web-jsptaglibrary\_1\_2.dtd"**>**
6. **<taglib>**
7. **<tlib-version>**1.0**</tlib-version>**
8. **<jsp-version>**1.2**</jsp-version>**
9. **<short-name>**simple**</short-name>**
10. **<uri>**http://tomcat.apache.org/example-taglib**</uri>**
11. **<description>**A simple tab library for the examples**</description>**
13. **<tag>**
14. **<name>**cube**</name>**
15. **<tag-class>**com.javatpoint.taghandler.CubeNumber**</tag-class>**
16. **<attribute>**
17. **<name>**number**</name>**
18. **<required>**true**</required>**
19. **</attribute>**
20. **</tag>**
21. **</taglib>**

**Output**

1. Cube of 4 is: 64

## 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**

1. **package** com.javatpoint;
2. **import** javax.servlet.jsp.JspException;
3. **import** javax.servlet.jsp.JspWriter;
4. **import** javax.servlet.jsp.tagext.TagSupport;
5. **import** java.sql.\*;
7. **public** **class** PrintRecord **extends** TagSupport{
8. **private** String id;
9. **private** String table;
11. **public** **void** setId(String id) {
12. **this**.id = id;
13. }
14. **public** **void** setTable(String table) {
15. **this**.table = table;
16. }
18. **public** **int** doStartTag()**throws** JspException{
19. JspWriter out=pageContext.getOut();
20. **try**{
21. Class.forName("oracle.jdbc.driver.OracleDriver");
22. Connection con=DriverManager.getConnection(
23. "jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
24. PreparedStatement ps=con.prepareStatement("select \* from "+table+" where id=?");
25. ps.setInt(1,Integer.parseInt(id));
26. ResultSet rs=ps.executeQuery();
27. **if**(rs!=**null**){
28. ResultSetMetaData rsmd=rs.getMetaData();
29. **int** totalcols=rsmd.getColumnCount();
30. //column name
31. out.write("<table border='1'>");
32. out.write("<tr>");
33. **for**(**int** i=1;i<=totalcols;i++){
34. out.write("<th>"+rsmd.getColumnName(i)+"</th>");
35. }
36. out.write("</tr>");
37. //column value
39. **if**(rs.next()){
40. out.write("<tr>");
41. **for**(**int** i=1;i<=totalcols;i++){
42. out.write("<td>"+rs.getString(i)+"</td>");
43. }
44. out.write("</tr>");
46. }**else**{
47. out.write("Table or Id doesn't exist");
48. }
49. out.write("</table>");
51. }
52. con.close();
53. }**catch**(Exception e){System.out.println(e);}
54. **return** SKIP\_BODY;
55. }
56. }

**m.tld**

1. **<?xml** version="1.0" encoding="ISO-8859-1" **?>**
2. <!DOCTYPE taglib
3. PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
4. "http://java.sun.com/j2ee/dtd/web-jsptaglibrary\_1\_2.dtd"**>**
6. **<taglib>**
8. **<tlib-version>**1.2**</tlib-version>**
9. **<jsp-version>**2.0**</jsp-version>**
10. **<short-name>**c**</short-name>**
11. **<uri>**javatpoint**</uri>**
13. **<tag>**
14. **<name>**printRecord**</name>**
15. **<tag-class>**com.javatpoint.PrintRecord**</tag-class>**
16. **<attribute>**
17. **<name>**id**</name>**
18. **<required>**true**</required>**
19. **</attribute>**
20. **<attribute>**
21. **<name>**table**</name>**
22. **<required>**true**</required>**
23. **</attribute>**
25. **</tag>**
26. **</taglib>**

**index.jsp**

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

#### Output

Iteration using JSP Custom Tag

1. [Iteration using JSP Custom Tag](http://www.javatpoint.com/Iteration-using-jsp-custom-tag)
2. [Example of Iteration using JSP Custom Tag](http://www.javatpoint.com/Iteration-using-jsp-custom-tag)

We can iterate the body content of any tag using the **doAfterBody()**method of **IterationTag interface**.

Here we are going to use the TagSupport class which implements the IterationTag interface. For iterating the body content, we need to use the **EVAL\_BODY\_AGAIN** constant in the doAfterBody() method.

Example of Iteration using JSP Custom Tag

In this example, we are going to use the attribute in the custom tag, which returns the power of any given number. We have created three files here

* index.jsp
* PowerNumber.java
* mytags.tld

**index.jsp**

1. <%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>
3. 3 ^ 5 = <m:power number="3" power="5">
4. body
5. </m:power>

**PowerNumber.java**

1. **package** com.javatpoint.taghandler;
3. **import** javax.servlet.jsp.JspException;
4. **import** javax.servlet.jsp.JspWriter;
5. **import** javax.servlet.jsp.tagext.TagSupport;
7. **public** **class** PowerNumber **extends** TagSupport{
8. **private** **int** number;
9. **private** **int** power;
10. **private** **static** **int** counter;
11. **private** **static** **int** result=1;
13. **public** **void** setPower(**int** power) {
14. **this**.power = power;
15. }
17. **public** **void** setNumber(**int** number) {
18. **this**.number = number;
19. }
21. **public** **int** doStartTag() **throws** JspException {
22. **return** EVAL\_BODY\_INCLUDE;
23. }
25. **public** **int** doAfterBody() {
26. counter++;
27. result \*= number;
28. **if** (counter==power)
29. **return** SKIP\_BODY;
30. **else**
31. **return** EVAL\_BODY\_AGAIN;
32. }
34. **public** **int** doEndTag() **throws** JspException {
35. JspWriter out=pageContext.getOut();
36. **try**{
37. out.print(result);
38. }**catch**(Exception e){e.printStackTrace();}
40. **return** EVAL\_PAGE;
41. }
42. }

**mytags.tld**

1. <?xml version="1.0" encoding="ISO-8859-1" ?>
2. <!DOCTYPE taglib
3. PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
4. "http://java.sun.com/j2ee/dtd/web-jsptaglibrary\_1\_2.dtd">
6. <taglib>
7. <tlib-version>1.0</tlib-version>
8. <jsp-version>1.2</jsp-version>
9. <**short**-name>simple</**short**-name>
10. <uri>http://tomcat.apache.org/example-taglib</uri>
11. <description>A simple tab library **for** the examples</description>
13. <tag>
14. <name>power</name>
15. <tag-**class**>com.javatpoint.taghandler.PowerNumber</tag-**class**>
17. <attribute>
18. <name>number</name>
19. <required>**true**</required>
20. </attribute>
22. <attribute>
23. <name>power</name>
24. <required>**true**</required>
25. </attribute>
27. </tag>
28. </taglib>

Looping using Iteration Tag (creating tag for loop)

Let's create a loop tag that iterates the body content of this tag.

*File: index.jsp*

1. <!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
2. <html>
3. <head>
4. <title>Insert title here</title>
5. </head>
6. <body>
8. <%@taglib prefix="m" uri="sssuri" %>
9. <m:loop end="5" start="1">
10. <p>My Name is khan</p>
11. </m:loop>
13. </body>
14. </html>

*File: mytags.tld*

1. <?xml version="1.0" encoding="ISO-8859-1" ?>
2. <!DOCTYPE taglib
3. PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
4. "http://java.sun.com/j2ee/dtd/web-jsptaglibrary\_1\_2.dtd">
5. <taglib>
6. <tlib-version>1.0</tlib-version>
7. <jsp-version>1.2</jsp-version>
8. <**short**-name>abc</**short**-name>
10. <uri>sssuri</uri>
11. <tag>
12. <name>loop</name>
13. <tag-**class**>com.javatpoint.customtag.Loop</tag-**class**>
15. <attribute>
16. <name>start</name>
17. <required>**true**</required>
18. </attribute>
20. <attribute>
21. <name>end</name>
22. <required>**true**</required>
23. </attribute>
24. </tag>
26. </taglib>

*File: Loop.java*

1. **package** com.javatpoint.customtag;
2. **import** javax.servlet.jsp.JspException;
3. **import** javax.servlet.jsp.tagext.TagSupport;
5. **public** **class** Loop **extends** TagSupport{
6. **private** **int** start=0;
7. **private** **int** end=0;
9. **public** **void** setStart(**int** start) {
10. **this**.start = start;
11. }
12. **public** **void** setEnd(**int** end) {
13. **this**.end = end;
14. }
16. @Override
17. **public** **int** doStartTag() **throws** JspException {
18. **return** EVAL\_BODY\_INCLUDE;
19. }
21. @Override
22. **public** **int** doAfterBody() **throws** JspException {
23. **if**(start<end){
24. start++;
25. **return** EVAL\_BODY\_AGAIN;
26. }**else**{
27. **return** SKIP\_BODY;
28. }
30. }

33. }

*File: web.xml*

1. **<?xml** version="1.0" encoding="UTF-8"**?>**
2. **<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/ns/javaee/web-app\_2\_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0"**>**
4. **<jsp-config>**
5. **<taglib>**
6. **<taglib-uri>**sssuri**</taglib-uri>**
7. **<taglib-location>**/WEB-INF/mytags.tld**</taglib-location>**
8. **</taglib>**
9. **</jsp-config>**
11. **</web-app>**

[download this example (developed using Eclipse ide)](http://www.javatpoint.com/src/jsp/customit2.zip)

**Output**

Custom URI in JSP Custom Tag

1. [Custom URI in JSP Custom Tag](http://www.javatpoint.com/custom-uri-in-jsp-custom-tag)
2. [Example to use Custom URI in JSP Custom Tag](http://www.javatpoint.com/custom-uri-in-jsp-custom-tag#ex)

We can use the custom URI, to tell the web container about the tld file. In such case, we need to define the taglib element in the web.xml. The web container gets the information about the tld file from the web.xml file for the specified URI.

Example to use custom URI in JSP Custom Tag

In this example, we are going to use the custom uri in the JSP file. For this application, we need to focus on 4 files.

* index.jsp
* web.xml
* mytags.tld
* PrintDate.java

**index.jsp**

1. <%@ taglib uri="mytags" prefix="m" %>
2. Today is: <m:today></m:today>

**web.xml**

1. <?xml version="1.0" encoding="UTF-8"?>
2. <!DOCTYPE web-app
3. PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
4. "http://java.sun.com/dtd/web-app\_2\_3.dtd">
6. <web-app>
8. <jsp-config>
9. <taglib>
10. <taglib-uri>mytags</taglib-uri>
11. <taglib-location>/WEB-INF/mytags.tld</taglib-location>
12. </taglib>
13. </jsp-config>
15. </web-app>

**mytags.tld**

1. <?xml version="1.0" encoding="ISO-8859-1" ?>
2. <!DOCTYPE taglib
3. PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
4. "http://java.sun.com/j2ee/dtd/web-jsptaglibrary\_1\_2.dtd">
6. <taglib>
7. <tlib-version>1.0</tlib-version>
8. <jsp-version>1.2</jsp-version>
9. <**short**-name>simple</**short**-name>
10. <uri>mytags</uri>
11. <description>A simple tab library **for** the examples</description>
13. <tag>
14. <name>today</name>
15. <tag-**class**>com.javatpoint.taghandler.PrintDate</tag-**class**>
16. </tag>
17. </taglib>

**PrintDate.java**

1. **package** com.javatpoint.taghandler;
3. **import** javax.servlet.jsp.JspException;
4. **import** javax.servlet.jsp.JspWriter;
5. **import** javax.servlet.jsp.tagext.TagSupport;
7. **public** **class** PrintDate **extends** TagSupport{
9. **public** **int** doStartTag() **throws** JspException {
10. JspWriter out=pageContext.getOut();
11. **try**{
12. out.print(java.util.Calendar.getInstance().getTime());
13. }**catch**(Exception e){e.printStackTrace();}
15. **return** SKIP\_BODY;
16. }

19. }

[download this example](http://www.javatpoint.com/src/jsp/customuri.zip)

# JSP CRUD Example

We can easily create CRUD Example in JSP. Here, we are using DAO files for database and JSTL for traversing records.

### Download jstl.jar and mysql-connector.jar

[Download jstl1.2.jar file](http://www.javatpoint.com/jsppages/src/jstl-1.2.jar)  
[Download mysql-connector.jar](http://www.javatpoint.com/src/jdbc/mysql-connector.jar)

### Download SQL File to Import in MySQL

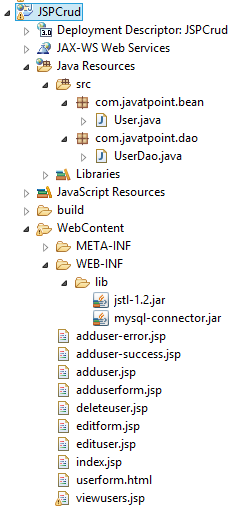
[Download SQL File](http://www.javatpoint.com/jsppages/src/register.sql)

### Download Project

[download CRUD project in JSP](http://www.javatpoint.com/jsppages/src/jspcrudapp.zip)

## CRUD Example

**Directory Structure in Eclipse**

   
**index.jsp**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
5. <title>JSP CRUD Example</title>
6. </head>
7. <body>
8. <h1>JSP CRUD Example</h1>
9. <a href="adduserform.jsp">Add User</a>
10. <a href="viewusers.jsp">View Users</a>
12. </body>
13. </html>

**adduserform.jsp**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
5. <title>Add User Form</title>
6. </head>
7. <body>
9. <jsp:include page="userform.html"></jsp:include>
11. </body>
12. </html>

**userform.html**

1. **<a** href="viewusers.jsp"**>**View All Records**</a><br/>**
3. **<h1>**Add New User**</h1>**
4. **<form** action="adduser.jsp" method="post"**>**
5. **<table>**
6. **<tr><td>**Name:**</td><td><input** type="text" name="name"**/></td></tr>**
7. **<tr><td>**Password:**</td><td>**
8. **<input** type="password" name="password"**/></td></tr>**
9. **<tr><td>**Email:**</td><td><input** type="email" name="email"**/></td></tr>**
10. **<tr><td>**Sex:**</td><td>**
11. **<input** type="radio" name="sex" value="male"**/>**Male
12. **<input** type="radio" name="sex" value="female"**/>**Female **</td></tr>**
13. **<tr><td>**Country:**</td><td>**
14. **<select** name="country" style="width:155px"**>**
15. **<option>**India**</option>**
16. **<option>**Pakistan**</option>**
17. **<option>**Afghanistan**</option>**
18. **<option>**Berma**</option>**
19. **<option>**Other**</option>**
20. **</select>**
21. **</td></tr>**
22. **<tr><td** colspan="2"**><input** type="submit" value="Add User"**/></td></tr>**
23. **</table>**
24. **</form>**

**adduser.jsp**

1. <%@page **import**="com.javatpoint.dao.UserDao"%>
2. <jsp:useBean id="u" **class**="com.javatpoint.bean.User"></jsp:useBean>
3. <jsp:setProperty property="\*" name="u"/>
5. <%
6. **int** i=UserDao.save(u);
7. **if**(i>0){
8. response.sendRedirect("adduser-success.jsp");
9. }**else**{
10. response.sendRedirect("adduser-error.jsp");
11. }
12. %>

**User.java**

1. **package** com.javatpoint.bean;
2. **public** **class** User {
3. **private** **int** id;
4. **private** String name,password,email,sex,country;
5. //generate getters and setters
6. }

**UserDao.java**

1. **package** com.javatpoint.dao;
2. **import** java.sql.\*;
3. **import** java.util.ArrayList;
4. **import** java.util.List;
5. **import** com.javatpoint.bean.User;
6. **public** **class** UserDao {
8. **public** **static** Connection getConnection(){
9. Connection con=**null**;
10. **try**{
11. Class.forName("com.mysql.jdbc.Driver");
12. con=DriverManager.getConnection("jdbc:mysql://localhost:3306/test","","");
13. }**catch**(Exception e){System.out.println(e);}
14. **return** con;
15. }
16. **public** **static** **int** save(User u){
17. **int** status=0;
18. **try**{
19. Connection con=getConnection();
20. PreparedStatement ps=con.prepareStatement(
21. "insert into register(name,password,email,sex,country) values(?,?,?,?,?)");
22. ps.setString(1,u.getName());
23. ps.setString(2,u.getPassword());
24. ps.setString(3,u.getEmail());
25. ps.setString(4,u.getSex());
26. ps.setString(5,u.getCountry());
27. status=ps.executeUpdate();
28. }**catch**(Exception e){System.out.println(e);}
29. **return** status;
30. }
31. **public** **static** **int** update(User u){
32. **int** status=0;
33. **try**{
34. Connection con=getConnection();
35. PreparedStatement ps=con.prepareStatement(
36. "update register set name=?,password=?,email=?,sex=?,country=? where id=?");
37. ps.setString(1,u.getName());
38. ps.setString(2,u.getPassword());
39. ps.setString(3,u.getEmail());
40. ps.setString(4,u.getSex());
41. ps.setString(5,u.getCountry());
42. ps.setInt(6,u.getId());
43. status=ps.executeUpdate();
44. }**catch**(Exception e){System.out.println(e);}
45. **return** status;
46. }
47. **public** **static** **int** delete(User u){
48. **int** status=0;
49. **try**{
50. Connection con=getConnection();
51. PreparedStatement ps=con.prepareStatement("delete from register where id=?");
52. ps.setInt(1,u.getId());
53. status=ps.executeUpdate();
54. }**catch**(Exception e){System.out.println(e);}
56. **return** status;
57. }
58. **public** **static** List<User> getAllRecords(){
59. List<User> list=**new** ArrayList<User>();
61. **try**{
62. Connection con=getConnection();
63. PreparedStatement ps=con.prepareStatement("select \* from register");
64. ResultSet rs=ps.executeQuery();
65. **while**(rs.next()){
66. User u=**new** User();
67. u.setId(rs.getInt("id"));
68. u.setName(rs.getString("name"));
69. u.setPassword(rs.getString("password"));
70. u.setEmail(rs.getString("email"));
71. u.setSex(rs.getString("sex"));
72. u.setCountry(rs.getString("country"));
73. list.add(u);
74. }
75. }**catch**(Exception e){System.out.println(e);}
76. **return** list;
77. }
78. **public** **static** User getRecordById(**int** id){
79. User u=**null**;
80. **try**{
81. Connection con=getConnection();
82. PreparedStatement ps=con.prepareStatement("select \* from register where id=?");
83. ps.setInt(1,id);
84. ResultSet rs=ps.executeQuery();
85. **while**(rs.next()){
86. u=**new** User();
87. u.setId(rs.getInt("id"));
88. u.setName(rs.getString("name"));
89. u.setPassword(rs.getString("password"));
90. u.setEmail(rs.getString("email"));
91. u.setSex(rs.getString("sex"));
92. u.setCountry(rs.getString("country"));
93. }
94. }**catch**(Exception e){System.out.println(e);}
95. **return** u;
96. }
97. }

**adduser-success.jsp**

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"**>**
5. **<title>**Add User Success**</title>**
6. **</head>**
7. **<body>**
9. **<p>**Record successfully saved!**</p>**
10. **<jsp:include** page="userform.html"**></jsp:include>**
12. **</body>**
13. **</html>**

**adduser-error.jsp**

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"**>**
5. **<title>**Add User Error**</title>**
6. **</head>**
7. **<body>**
9. **<p>**Sorry, an error occurred!**</p>**
10. **<jsp:include** page="userform.html"**></jsp:include>**
12. **</body>**
13. **</html>**

**viewusers.jsp**

1. <!DOCTYPE html**>**
3. **<html>**
4. **<head>**
5. **<meta** http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"**>**
6. **<title>**View Users**</title>**
7. **</head>**
8. **<body>**
10. **<**%@page import="com.javatpoint.dao.UserDao,com.javatpoint.bean.\*,java.util.\*"%**>**
11. **<**%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%**>**
13. **<h1>**Users List**</h1>**
15. **<**%
16. List**<User>** list=UserDao.getAllRecords();
17. request.setAttribute("list",list);
18. %**>**
20. **<table** border="1" width="90%"**>**
21. **<tr><th>**Id**</th><th>**Name**</th><th>**Password**</th><th>**Email**</th>**
22. **<th>**Sex**</th><th>**Country**</th><th>**Edit**</th><th>**Delete**</th></tr>**
23. **<c:forEach** items="${list}" var="u"**>**
24. **<tr><td>**${u.getId()}**</td><td>**${u.getName()}**</td><td>**${u.getPassword()}**</td>**
25. **<td>**${u.getEmail()}**</td><td>**${u.getSex()}**</td><td>**${u.getCountry()}**</td>**
26. **<td><a** href="editform.jsp?id=${u.getId()}"**>**Edit**</a></td>**
27. **<td><a** href="deleteuser.jsp?id=${u.getId()}"**>**Delete**</a></td></tr>**
28. **</c:forEach>**
29. **</table>**
30. **<br/><a** href="adduserform.jsp"**>**Add New User**</a>**
32. **</body>**
33. **</html>**

**editform.jsp**

1. <!DOCTYPE html**>**
2. **<html>**
3. **<head>**
4. **<meta** http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"**>**
5. **<title>**Edit Form**</title>**
6. **</head>**
7. **<body>**
8. **<**%@page import="com.javatpoint.dao.UserDao,com.javatpoint.bean.User"%**>**
10. **<**%
11. String id=request.getParameter("id");
12. User u=UserDao.getRecordById(Integer.parseInt(id));
13. %**>**
15. **<h1>**Edit Form**</h1>**
16. **<form** action="edituser.jsp" method="post"**>**
17. **<input** type="hidden" name="id" value="<%=u.getId() %>"**/>**
18. **<table>**
19. **<tr><td>**Name:**</td><td>**
20. **<input** type="text" name="name" value="<%= u.getName()%>"**/></td></tr>**
21. **<tr><td>**Password:**</td><td>**
22. **<input** type="password" name="password" value="<%= u.getPassword()%>"**/></td></tr>**
23. **<tr><td>**Email:**</td><td>**
24. **<input** type="email" name="email" value="<%= u.getEmail()%>"**/></td></tr>**
25. **<tr><td>**Sex:**</td><td>**
26. **<input** type="radio" name="sex" value="male"**/>**Male
27. **<input** type="radio" name="sex" value="female"**/>**Female **</td></tr>**
28. **<tr><td>**Country:**</td><td>**
29. **<select** name="country"**>**
30. **<option>**India**</option>**
31. **<option>**Pakistan**</option>**
32. **<option>**Afghanistan**</option>**
33. **<option>**Berma**</option>**
34. **<option>**Other**</option>**
35. **</select>**
36. **</td></tr>**
37. **<tr><td** colspan="2"**><input** type="submit" value="Edit User"**/></td></tr>**
38. **</table>**
39. **</form>**
41. **</body>**
42. **</html>**

**edituser.jsp**

1. **<**%@page import="com.javatpoint.dao.UserDao"%**>**
2. **<jsp:useBean** id="u" class="com.javatpoint.bean.User"**></jsp:useBean>**
3. **<jsp:setProperty** property="\*" name="u"**/>**
4. **<**%
5. int i=UserDao.update(u);
6. response.sendRedirect("viewusers.jsp");
7. %**>**

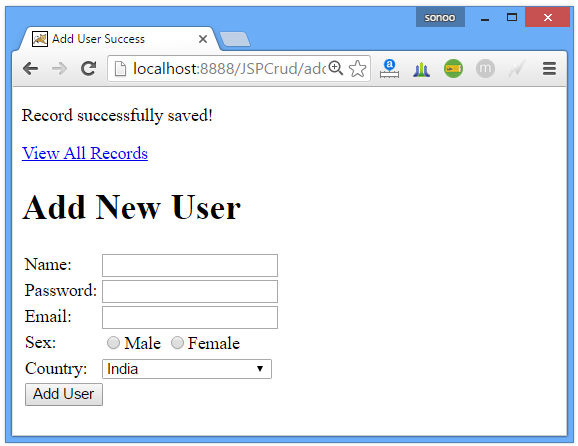
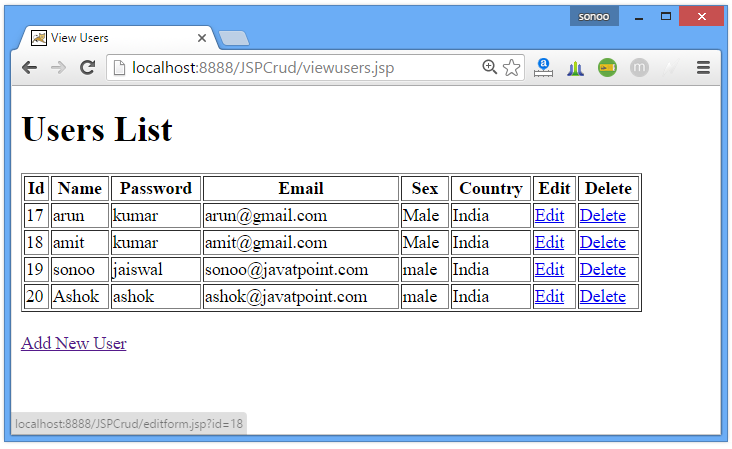
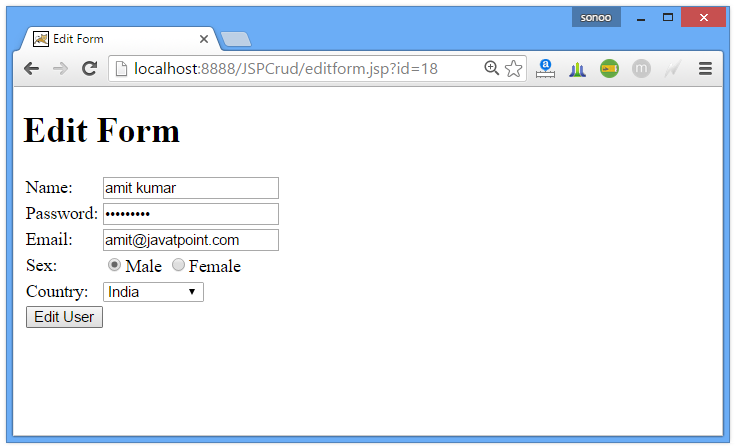
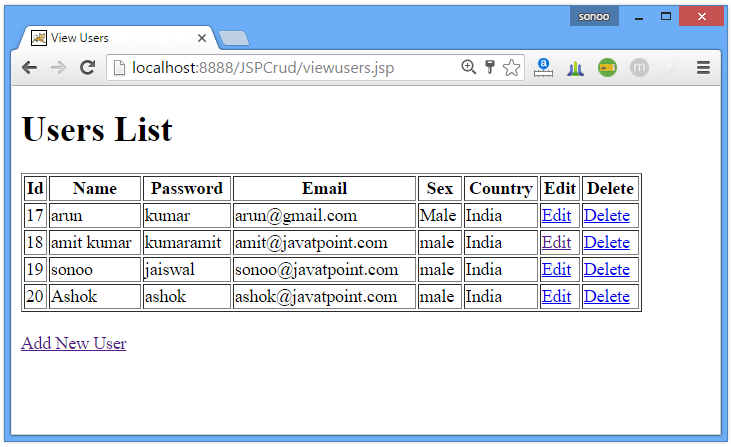
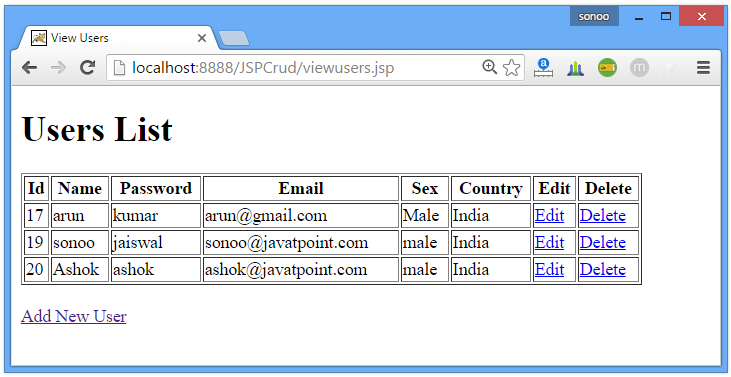
**deleteuser.jsp**

1. **<**%@page import="com.javatpoint.dao.UserDao"%**>**
2. **<jsp:useBean** id="u" class="com.javatpoint.bean.User"**></jsp:useBean>**
3. **<jsp:setProperty** property="\*" name="u"**/>**
4. **<**%
5. UserDao.delete(u);
6. response.sendRedirect("viewusers.jsp");
7. %**>**

### Download Project

[download CRUD project in JSP](http://www.javatpoint.com/jsppages/src/jspcrudapp.zip)

### Output

**Next Topic**[Registration Form in JSP](http://www.javatpoint.com/registration-form-in-jsp)

Registration Form in JSP

1. [Registration Form in JSP](http://www.javatpoint.com/registration-form-in-jsp)
2. [Example of Registration Form in JSP](http://www.javatpoint.com/registration-form-in-jsp#jspregisex)

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 create the table in the Oracle database: |

1. CREATE TABLE  "USER432"
2. (    "NAME" VARCHAR2(4000),
3. "EMAIL" VARCHAR2(4000),
4. "PASS" VARCHAR2(4000)
5. )
6. /

We have created the table named user432 here.

**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.

1. <form action="process.jsp">
2. <input type="text" name="uname" value="Name..." onclick="this.value=''"/><br/>
3. <input type="text" name="uemail"  value="Email ID..." onclick="this.value=''"/><br/>
4. <input type="password" name="upass"  value="Password..." onclick="this.value=''"/><br/>
5. <input type="submit" value="register"/>
6. </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.

1. <%@page **import**="bean.RegisterDao"%>
2. <jsp:useBean id="obj" **class**="bean.User"/>
4. <jsp:setProperty property="\*" name="obj"/>
6. <%
7. **int** status=RegisterDao.register(obj);
8. **if**(status>0)
9. out.print("You are successfully registered");
11. %>

**User.java**

|  |
| --- |
| It is the bean class that have 3 properties uname, uemail and upass with its setter and getter methods. |

1. **package** bean;
3. **public** **class** User {
4. **private** String uname,upass,uemail;
6. **public** String getUname() {
7. **return** uname;
8. }
10. **public** **void** setUname(String uname) {
11. **this**.uname = uname;
12. }
14. **public** String getUpass() {
15. **return** upass;
16. }
18. **public** **void** setUpass(String upass) {
19. **this**.upass = upass;
20. }
22. **public** String getUemail() {
23. **return** uemail;
24. }
26. **public** **void** setUemail(String uemail) {
27. **this**.uemail = uemail;
28. }
30. }

**Provider.java**

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

1. **package** bean;
3. **public** **interface** Provider {
4. String DRIVER="oracle.jdbc.driver.OracleDriver";
5. String CONNECTION\_URL="jdbc:oracle:thin:@localhost:1521:xe";
6. String USERNAME="system";
7. String PASSWORD="oracle";
9. }

**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.

1. **package** bean;
2. **import** java.sql.\*;
3. **import** **static** bean.Provider.\*;
5. **public** **class** ConnectionProvider {
6. **private** **static** Connection con=**null**;
7. **static**{
8. **try**{
9. Class.forName(DRIVER);
10. con=DriverManager.getConnection(CONNECTION\_URL,USERNAME,PASSWORD);
11. }**catch**(Exception e){}
12. }
14. **public** **static** Connection getCon(){
15. **return** con;
16. }
18. }

**RegisterDao.java**

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

1. **package** bean;
3. **import** java.sql.\*;
5. **public** **class** RegisterDao {
7. **public** **static** **int** register(User u){
8. **int** status=0;
9. **try**{
10. Connection con=ConnectionProvider.getCon();
11. PreparedStatement ps=con.prepareStatement("insert into user432 values(?,?,?)");
12. ps.setString(1,u.getUname());
13. ps.setString(2,u.getUemail());
14. ps.setString(3,u.getUpass());
16. status=ps.executeUpdate();
17. }**catch**(Exception e){}
19. **return** status;
20. }
22. }

Login and Logout Example in JSP

1. [Login and Logout Example in JSP](http://www.javatpoint.com/login-form-in-jsp)
2. [Example of Login Form in JSP](http://www.javatpoint.com/login-form-in-jsp#jsploginex)

|  |
| --- |
| In this example of creating login form, we have used the DAO (Data Access Object), Factory method 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\_URL, USERNAME and PASSWORD * **ConnectionProvider.java**, a class that is responsible to return the object of Connection. It uses the 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 the 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:

1. CREATE TABLE  "USER432"
2. (    "NAME" VARCHAR2(4000),
3. "EMAIL" VARCHAR2(4000),
4. "PASS" VARCHAR2(4000)
5. )
6. /

We assume that there are many records in this table.

**index.jsp**

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

1. <a href="login.jsp">login</a>|
2. <a href="logout.jsp">logout</a>|
3. <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.

1. <%@ include file="index.jsp" %>
2. <hr/>
4. <h3>Login Form</h3>
5. <%
6. String profile\_msg=(String)request.getAttribute("profile\_msg");
7. **if**(profile\_msg!=**null**){
8. out.print(profile\_msg);
9. }
10. String login\_msg=(String)request.getAttribute("login\_msg");
11. **if**(login\_msg!=**null**){
12. out.print(login\_msg);
13. }
14. %>
15. <br/>
16. <form action="loginprocess.jsp" method="post">
17. Email:<input type="text" name="email"/><br/><br/>
18. Password:<input type="password" name="password"/><br/><br/>
19. <input type="submit" value="login"/>"
20. </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.

1. <%@page **import**="bean.LoginDao"%>
2. <jsp:useBean id="obj" **class**="bean.LoginBean"/>
4. <jsp:setProperty property="\*" name="obj"/>
6. <%
7. **boolean** status=LoginDao.validate(obj);
8. **if**(status){
9. out.println("You r successfully logged in");
10. session.setAttribute("session","TRUE");
11. }
12. **else**
13. {
14. out.print("Sorry, email or password error");
15. %>
16. <jsp:include page="index.jsp"></jsp:include>
17. <%
18. }
19. %>

**LoginBean.java**

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

1. **package** bean;
3. **public** **class** LoginBean {
4. **private** String email,pass;
6. **public** String getEmail() {
7. **return** email;
8. }
10. **public** **void** setEmail(String email) {
11. **this**.email = email;
12. }
14. **public** String getPass() {
15. **return** pass;
16. }
18. **public** **void** setPass(String pass) {
19. **this**.pass = pass;
20. }

23. }

**Provider.java**

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

1. **package** bean;
3. **public** **interface** Provider {
4. String DRIVER="oracle.jdbc.driver.OracleDriver";
5. String CONNECTION\_URL="jdbc:oracle:thin:@localhost:1521:xe";
6. String USERNAME="system";
7. String PASSWORD="oracle";
9. }

**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.

1. **package** bean;
2. **import** java.sql.\*;
3. **import** **static** bean.Provider.\*;
5. **public** **class** ConnectionProvider {
6. **private** **static** Connection con=**null**;
7. **static**{
8. **try**{
9. Class.forName(DRIVER);
10. con=DriverManager.getConnection(CONNECTION\_URL,USERNAME,PASSWORD);
11. }**catch**(Exception e){}
12. }
14. **public** **static** Connection getCon(){
15. **return** con;
16. }
18. }

**LoginDao.java**

This class varifies the emailid and password.

1. **package** bean;
2. **import** java.sql.\*;
3. **public** **class** LoginDao {
5. **public** **static** **boolean** validate(LoginBean bean){
6. **boolean** status=**false**;
7. **try**{
8. Connection con=ConnectionProvider.getCon();
10. PreparedStatement ps=con.prepareStatement(
11. "select \* from user432 where email=? and pass=?");
13. ps.setString(1,bean.getEmail());
14. ps.setString(2, bean.getPass());
16. ResultSet rs=ps.executeQuery();
17. status=rs.next();
19. }**catch**(Exception e){}
21. **return** status;
23. }
24. }

Uploading file to the server using JSP

1. [Uploading file to the server using JSP](http://www.javatpoint.com/uploading-file-to-the-server-in-jsp)
2. [MultipartRequest class](http://www.javatpoint.com/uploading-file-to-the-server-in-jsp#multipart)
3. [Constructors of MultipartRequest class](http://www.javatpoint.com/uploading-file-to-the-server-in-jsp#multipartc)
4. [Example of File Upload in JSP](http://www.javatpoint.com/uploading-file-to-the-server-in-jsp#jspuploadex)

There are many ways to upload the file to the server. One of the way is by the MultipartRequest class. For using this class you need to have the cos.jar file. In this example, we are providing the cos.jar file alongwith the code.

MultipartRequest class

|  |
| --- |
| It is a utility class to handle the multipart/form-data request. There are many constructors defined in the MultipartRequest class. |

**Commonly used Constructors of MultipartRequest class**

* **MultipartRequest(HttpServletRequest request, String saveDirectory)** uploads the file upto 1MB.
* **MultipartRequest(HttpServletRequest request, String saveDirectory, int maxPostSize)** uploads the file upto specified post size.

### MultipartRequest(HttpServletRequest request, String saveDirectory, int maxPostSize, String encoding) uploads the file upto specifiedExample of File Upload in JSP

In this example, we are creating two files only, index.jsp and fileupload.jsp.

**index.jsp**

To upload the file to the server, there are two requirements:

1. You must use the post request.
2. encodeType should be multipart/form-data that gives information to the server that you are going to upload the file.
3. <form action="upload.jsp" method="post" enctype="multipart/form-data">
4. Select File:<input type="file" name="fname"/><br/>
5. <input type="image" src="MainUpload.png"/>
6. </form>

**upload.jsp**

We are uploading the incoming file to the location d:/new, you can specify your location here.

1. <%@ page **import**="com.oreilly.servlet.MultipartRequest" %>
2. <%
3. MultipartRequest m = **new** MultipartRequest(request, "d:/new");
4. out.print("successfully uploaded");
6. %>

If size of the file is greater than 1MB, you should specify the post size.

* post size with given encoding.

Example of Downloading file from the server using JSP

In this example, we are going to download the jsp file. But you may download any file. For downloading the file from the server, you should specify the content type named APPLICATION/OCTET-STREAM.

**index.jsp**

This file provides a link to download the jsp file.

1. <a href="download.jsp">download the jsp file</a>

**download.jsp**

In this example, we are downloading the file home.jsp which is located in the e: drive. You may change this location accordingly.

1. <%
2. String filename = "home.jsp";
3. String filepath = "e:\\";
4. response.setContentType("APPLICATION/OCTET-STREAM");
5. response.setHeader("Content-Disposition","attachment; filename=\"" + filename + "\"");
7. java.io.FileInputStream fileInputStream=**new** java.io.FileInputStream(filepath + filename);
9. **int** i;
10. **while** ((i=fileInputStream.read()) != -1) {
11. out.write(i);
12. }
13. fileInputStream.close();
14. %>