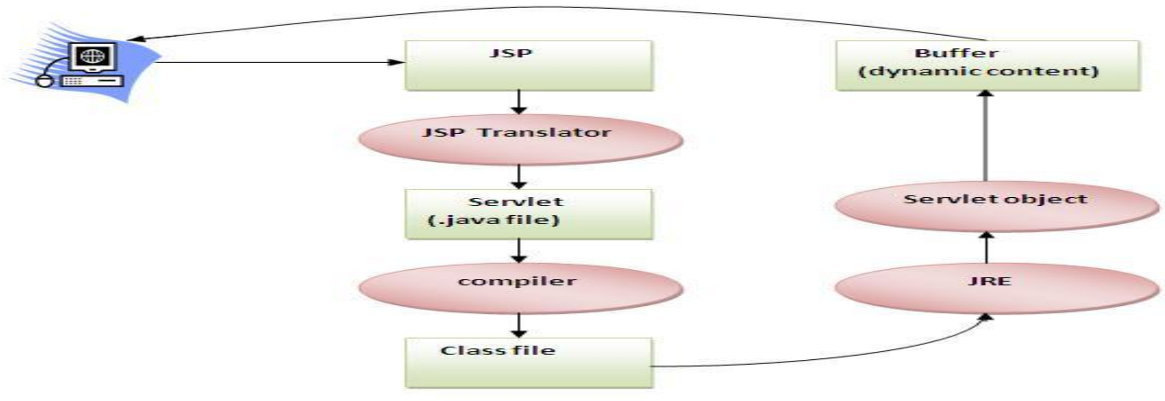
**JSP** technology is used to create web application just like Servlet technology.It can be thought of as an extension to servlet because it provides more functionality than servlet such as expression language, jstl etc.
A JSP page consists of HTML tags and JSP tags. The jsp pages are easier to maintain than servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom 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 servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.
**2) Easy to maintain**
JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.
**3) Fast Development: No need to recompile and redeploy**
If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
**4) Less code than Servlet**
In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. 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.

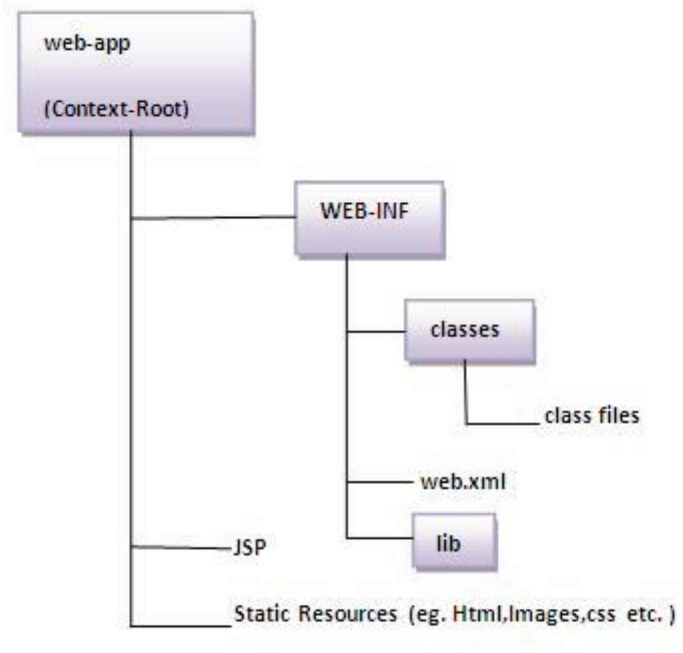
**Creating a simple JSP Page**
To create the first jsp page, write some html code as given below, and save it by .jsp extension. We have save this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the jsp page.

**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.

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

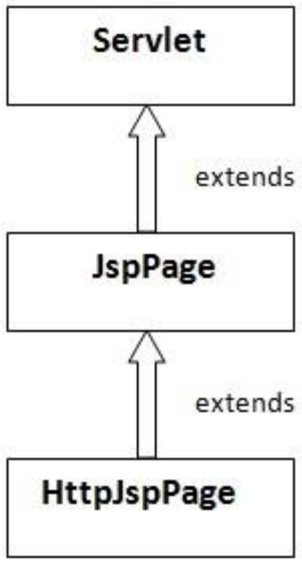
It will print **10** on the browser.
How to run a simple JSP Page ?
Follow the following steps to execute this JSP page:
Start the server
put the jsp file in a folder and deploy on the server
visit the browser by the url http://localhost:portno/contextRoot/jspfile e.g.
http://localhost:8888/myapplication/index.jsp

**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:
javax.servlet.jsp
javax.servlet.jsp.tagext
**javax.servlet.jsp package**
The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:
JspPage
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**
**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.
**public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

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

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

**SP 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.
**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:

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

**Simple Example of JSP scriptlet tag**
In this example, we are displaying a welcome message.

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

**Example of JSP scriptlet tag that prints the user name**
In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.
**index.html**

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

**welcome.jsp**

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

**JSP expression tag**

The code placed within 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**

```
<%=  statement %>
```

**Example of JSP expression tag**

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

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

</html>
```

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

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

**Example of JSP expression tag that prints the user name**

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

**index.html**

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

**welcome.jsp**

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

## JSP Declaration Tag

The JSP declaration tag is used to declare fields and methods.
The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.
So it doesn't get memory at each request.
Syntax of JSP declaration tag
The syntax of the declaration tag is as follows:
<%!  field or method declaration %>

**Difference between the jsp scriptlet tag and jsp declaration tag ?**

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

**Example of JSP declaration tag that declares field**

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

```
<html>
<body>

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

</body>
</html>
```

**Example of JSP declaration tag that declares method**

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

```
<html>
<body>

<%!
int cube(int n){
return n*n*n*;
}
%>

<%= "Cube of 3 is:"+cube(3) %>

</body>
</html>
```

## JSP Implicit Objects

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.
The available implicit objects are out, request, config, session, application etc.
A list of the 9 implicit objects is given below:

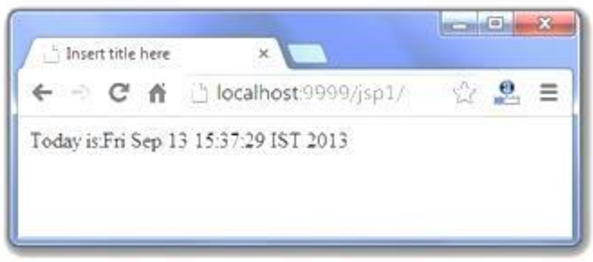
| Object | Type |
|---|---|
| out | JspWriter |
| request | HttpServletRequest |
| response | HttpServletResponse |
| config | ServletConfig |
| application | ServletContext |
| session | HttpSession |
| pageContext | PageContext |
| page | Object |
| exception | Throwable |

**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:
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

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

**Output**



```
Today is:Fri Sep 13 15:37:29 IST 2013
```

**JSP request implicit object**

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

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

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**

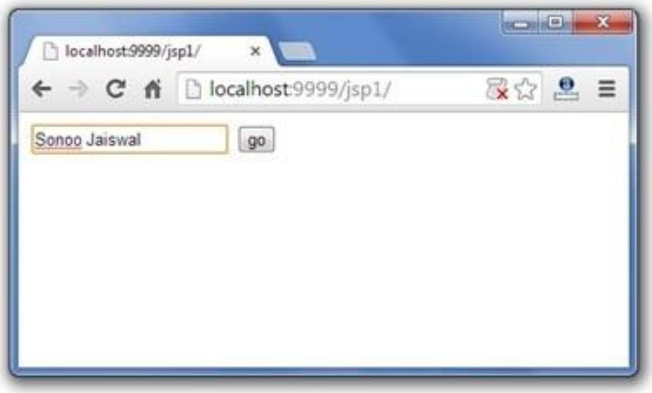
index.html

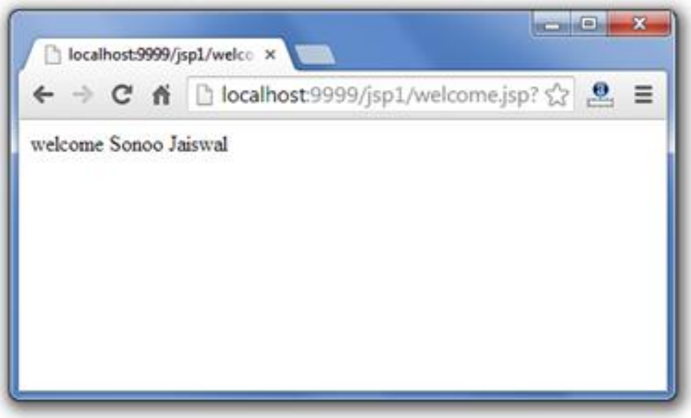
```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

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

**Output**

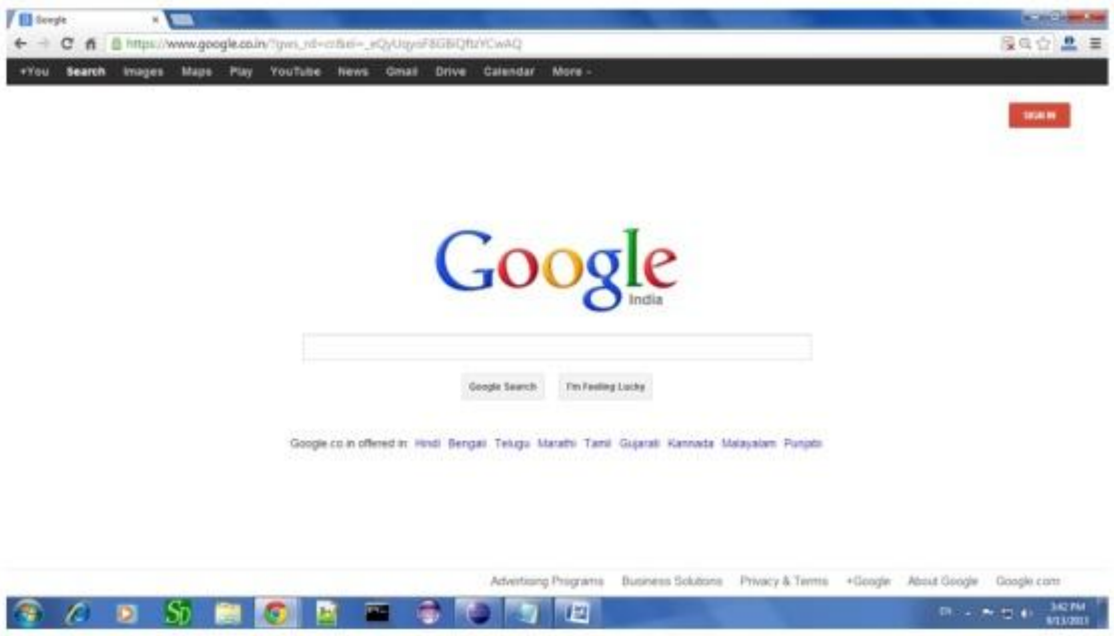




```
welcome Sonoo Jaiswal
```

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**
```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```
**welcome.jsp**
```
<%
response.sendRedirect("http://www.google.com");
%>
```
Output



4) JSP config implicit object

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

**Example of config implicit object:**

**index.html**

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

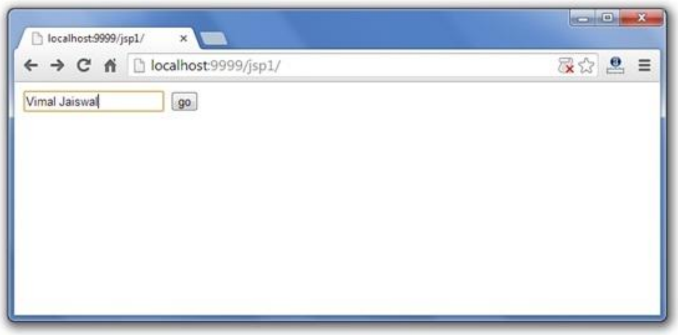
**web.xml file**

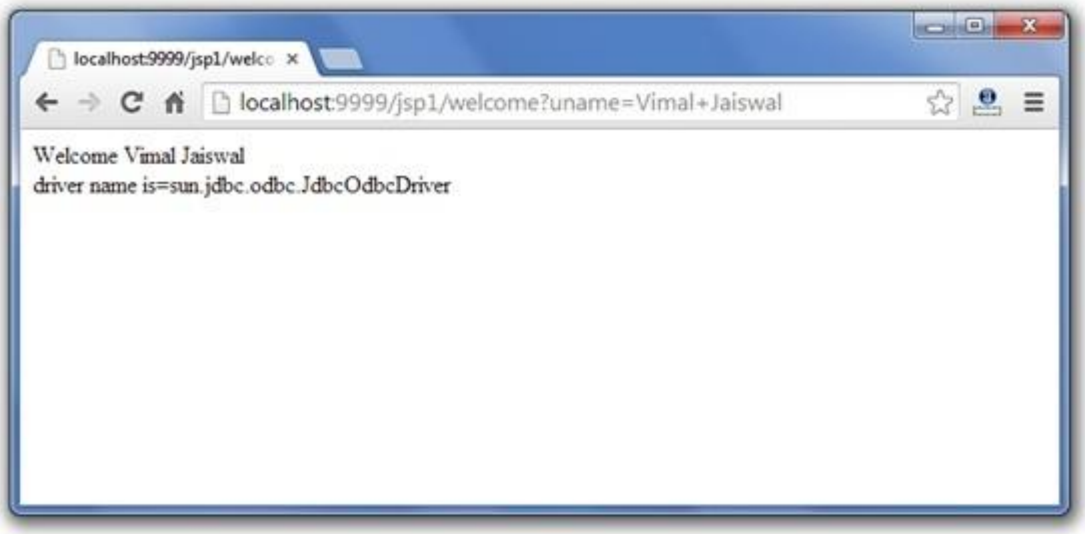
```
<web-app>

<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>

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

</servlet>

<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```

**welcome.jsp**

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

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

**Output**




localhost:9999/jsp1/welco
localhost:9999/jsp1/welcome?uname=Vimal+Jaiswal

```
Welcome Vimal Jaiswal
driver name is=sun.jdbc.odbc.JdbcOdbcDriver
```

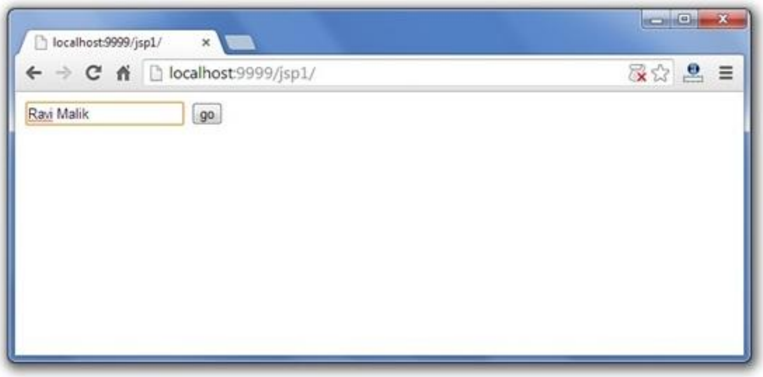
**6) session implicit object**
In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information.
**Example of session implicit object**
index.html

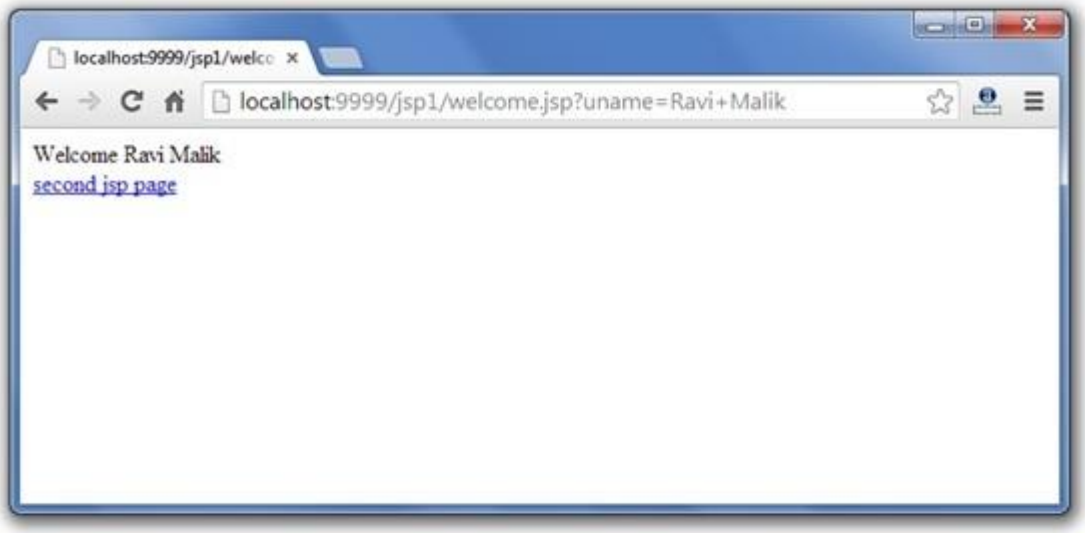
```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%

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

session.setAttribute("user",name);
```

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

%>
</body>
</html>
```
second.jsp
```
<html>
<body>
<%

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

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




localhost:9999/jsp1/welco
localhost:9999/jsp1/welcome.jsp?uname=Ravi+Malik
Welcome Ravi Malik
second jsp page

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

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

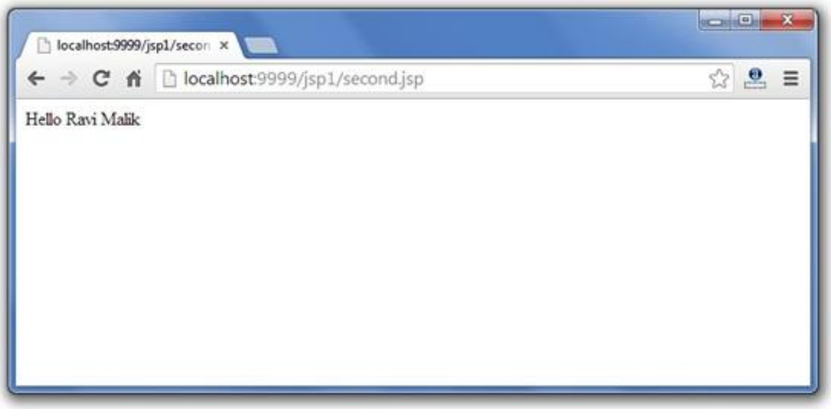
welcome.jsp

```
<html>
<body>
<%

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

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

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

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

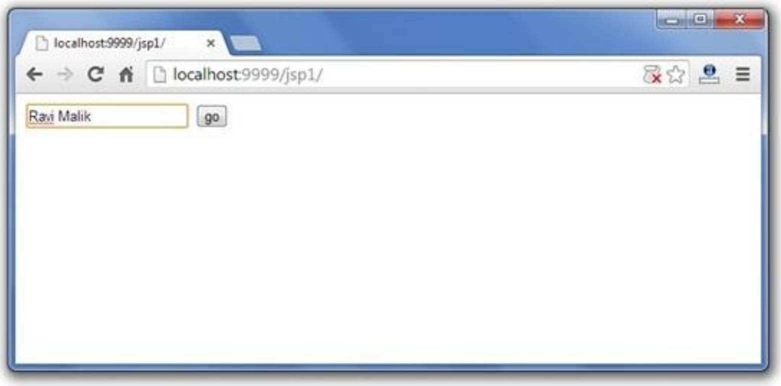
second.jsp

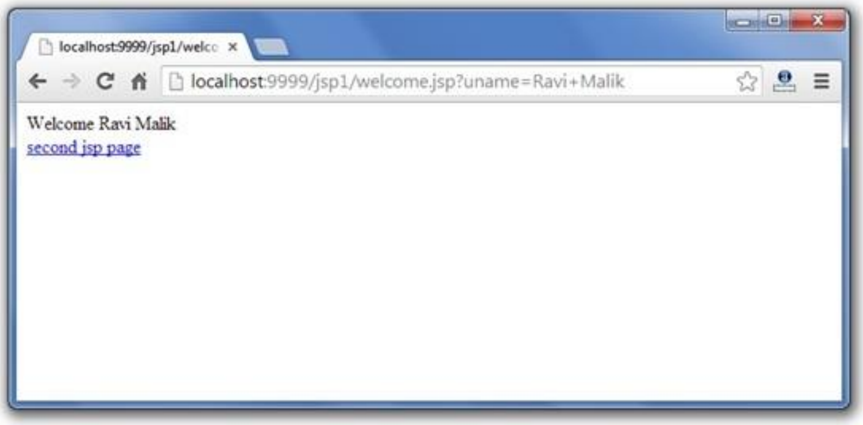
```
<html>
<body>
<%

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

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

Output

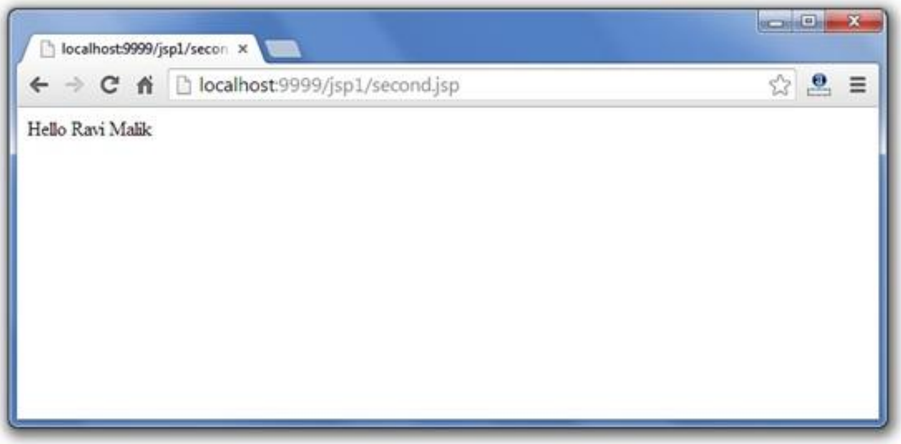


Hello Ravi Malik




localhost:9999/jsp1/welco
localhost:9999/jsp1/welcome.jsp?uname=Ravi+Malik

Welcome Ravi Malik
second jsp page



Hello Ravi Malik

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

```
<%@ page isErrorPage="true" %>
<html>
<body>
Sorry following exception occured:<%= exception %>
</body>  </html>
```

**JSP directives**

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

There are three types of directives:

page directive
include directive
taglib directive

Syntax of JSP Directive

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

**JSP page directive**

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

Syntax of JSP page directive

```
<%@ 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**

```
<html>
<body>

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

</body>
</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**

```
<html>
<body>

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

```
Today is: <%= new java.util.Date() %>

</body>
</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**

```
<html>
<body>

<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>

</body>
</html>
```

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

```
public String getServletInfo() {
  return "composed by Sonoo Jaiswal";
}
```

**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**

```
<html>
<body>

<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>

</body>
</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.

```
<%@ 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:

```
public class SimplePage_jsp extends HttpJspBase
  implements SingleThreadModel{
.......
}
```

---

**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**

```
//index.jsp
<html>
<body>

<%@ page errorPage="myerrorpage.jsp" %>

 <%= 100/0 %>

</body>
</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**

```
//myerrorpage.jsp
<html>
<body>

<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>
The exception is: <%= exception %>

</body>
</html>
```

```
<%@ 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:

```
public class SimplePage_jsp extends HttpJspBase
  implements SingleThreadModel{
.......
}
```

**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**

```
//index.jsp
<html>
<body>

<%@ page errorPage="myerrorpage.jsp" %>

 <%= 100/0 %>

</body>
</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**

```
//myerrorpage.jsp
<html>
<body>

<%@ page isErrorPage="true" %>

 Sorry an exception occured!<br/>
The exception is: <%= exception %>

</body>
</html>
```

**Jsp Include Directive**

The include directive is used to include the contents of any resource it may be jsp file, html file or text 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

```
<%@ 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.

```
<html>
<body>

<%@ include file="header.html" %>

Today is: <%= java.util.Calendar.getInstance().getTime() %>

</body>
</html>
```

## Exception Handling in JSP

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:

By **errorPage** and **isErrorPage** attributes of page directive

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

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

process.jsp

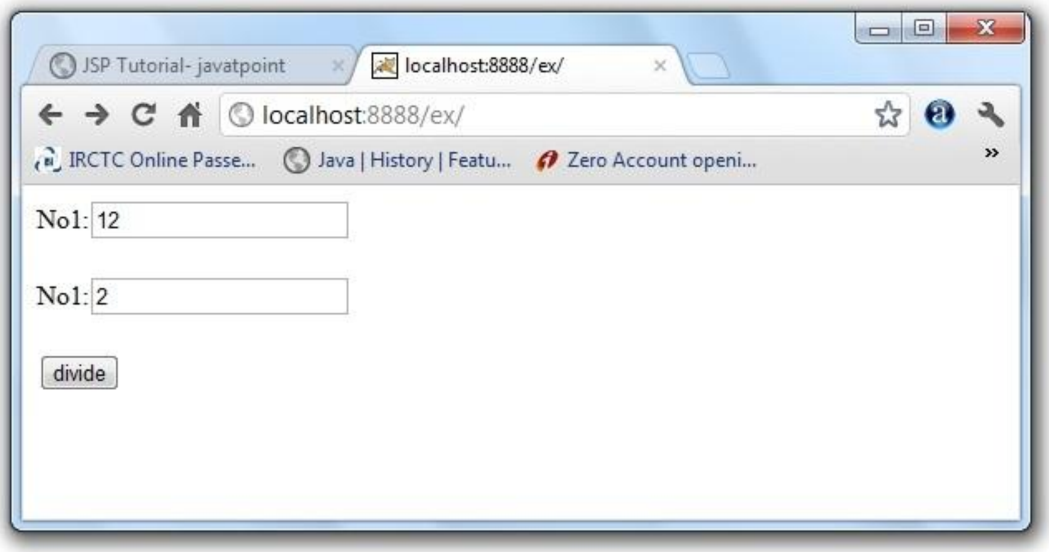
```
<%@ page errorPage="error.jsp" %>
<%

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

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

%>
```

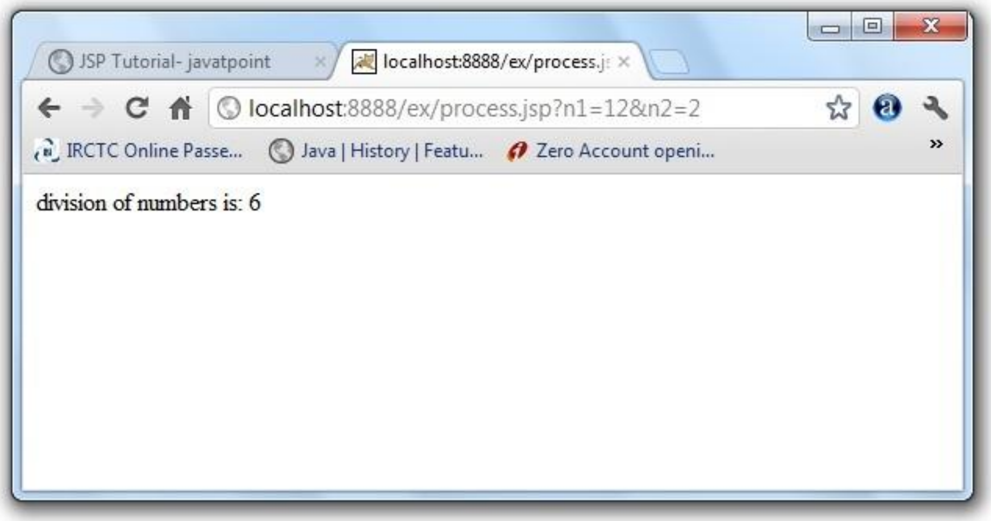
error.jsp

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

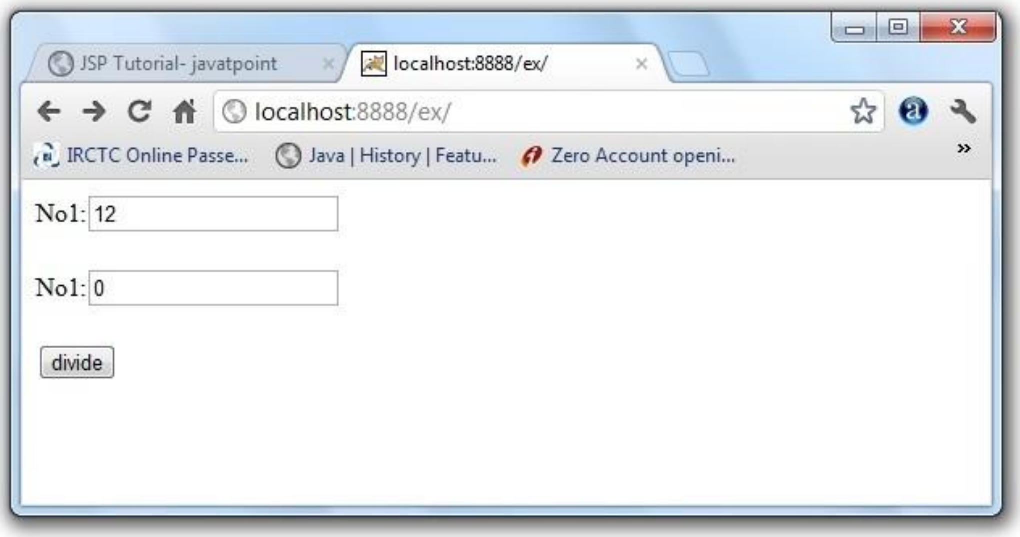
```
<h3>Sorry an exception occured!</h3>

Exception is: <%= exception %>
```

**Output of this example:**



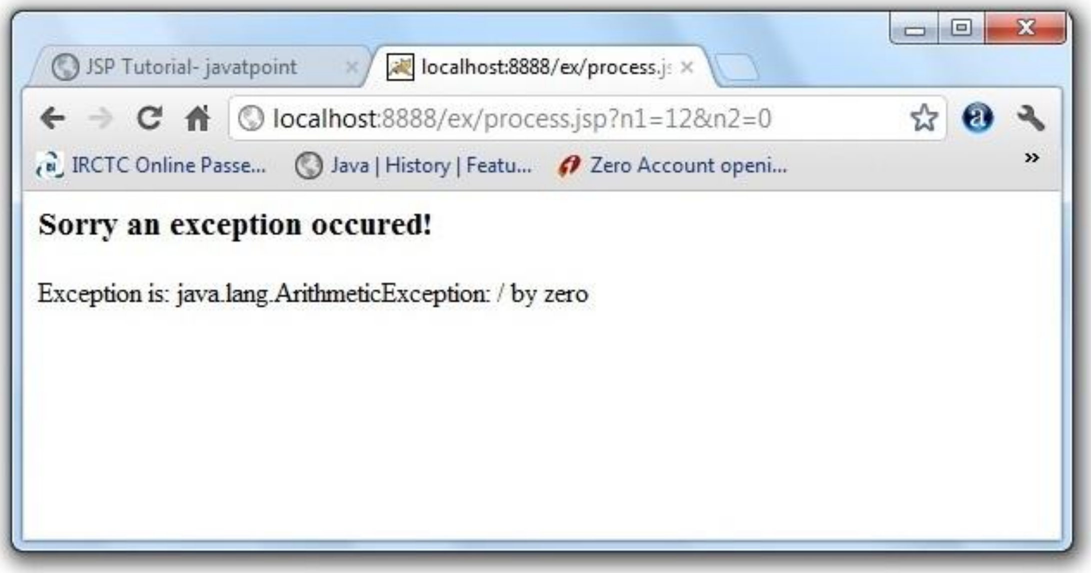

JSP Tutorial- javatpoint
localhost:8888/ex/process.j
localhost:8888/ex/process.jsp?n1=12&n2=2
IRCTC Online Passe...
Java | History | Featu...
Zero Account openi...

division of numbers is: 6



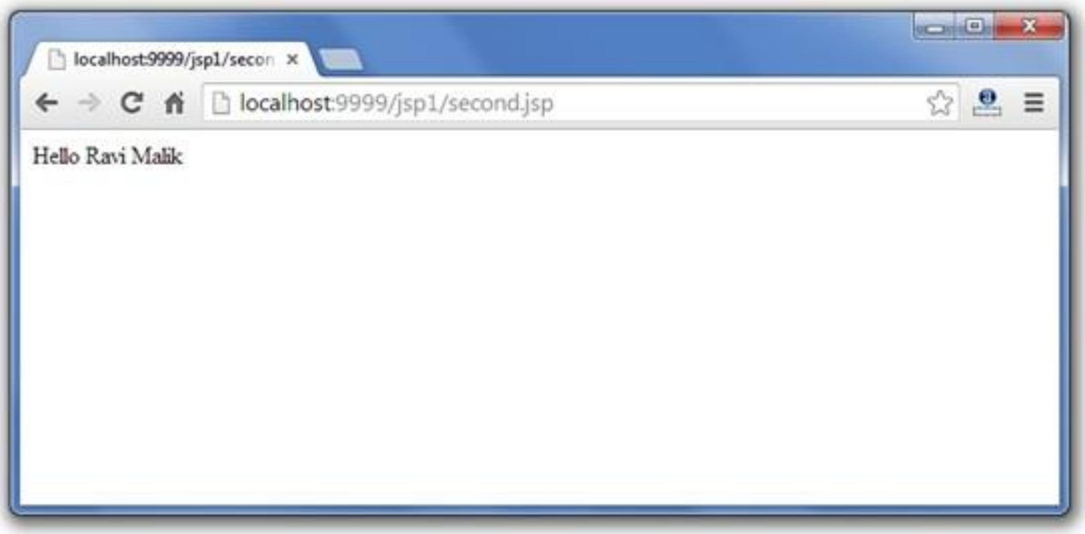
No1: 12

No1: 0


JSP Tutorial- javatpoint
localhost:8888/ex/process.j
localhost:8888/ex/process.jsp?n1=12&n2=0
IRCTC Online Passe...
Java | History | Featu...
Zero Account openi...

**Sorry an exception occured!**

Exception is: java.lang.ArithmeticException: / by zero



Hello Ravi Malik

**JSP Action Tag**
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

```
<jsp:forward page="relativeURL | <%= expression %>" />
```
Syntax of jsp:forward action tag with parameter
```
<jsp:forward page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:forward>
```

**Example of jsp:forward action tag without parameter**
In this example, we are simply forwarding the request to the printdate.jsp file.
index.jsp
```
<html>
<body>
<h2>this is index page</h2>

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

**Example of jsp:forward action tag with parameter**
In this example, we are forwarding the request to the printdate.jsp file with parameter and printdate.jsp file prints the parameter value with date and time.
index.jsp
```
<html>
<body>
<h2>this is index page</h2>

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

</body>
</html>
```
printdate.jsp
```
<html>
<body>

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

</body>
</html>
```

**jsp:include action tag**

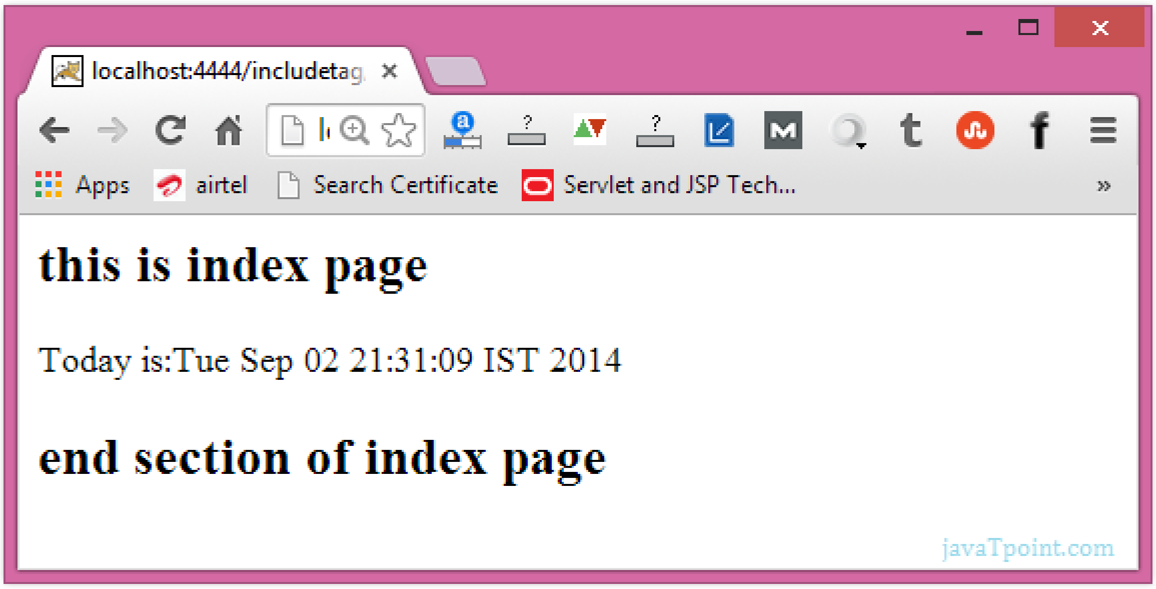
The **jsp:include action tag** is used to include the content of another resource it may be jsp, html or servlet.
The jsp include action tag includes the resource at request time so it is**better for dynamic pages** because there might be changes in future.
The jsp:include tag can be used to include static as well as dynamic pages.
**Advantage of jsp:include action tag**
**Code reusability** : We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.
**Difference between jsp include directive and include action**

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

---

Syntax of jsp:include action tag without parameter
```
<jsp:include page="relativeURL | <%= expression %>" />
```
Syntax of jsp:include action tag with parameter
```
<jsp:include page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:include>
```

**Example of jsp:include action tag without parameter**
In this example, index.jsp file includes the content of another printdate.jsp file.
*File: index.jsp*
```
<h2>this is index page</h2>

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

<h2>end section of index page</h2>
```
*File: printdate.jsp*
```
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
```



this is index page

Today is:Tue Sep 02 21:31:09 IST 2014

end section of index page

**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**

```
//Employee.java

package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;

public Employee(){}

public void setId(int id){this.id=id;}

public int getId(){return id;}

public void setName(String name){this.name=name;}

public String getName(){return name;}

}
```

**How to access the java bean class?**

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

```
package mypack;
public class Test{
```

```
public static void main(String args[]){

Employee e=new Employee();//object is created

e.setName("Arjun");//setting value to the object

System.out.println(e.getName());

}}
```

*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

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

Syntax of jsp:useBean action tag

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

**Attributes and Usage of jsp:useBean action tag**

**id:** is used to identify the bean in the specified scope.

**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.

**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.

**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.

**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.

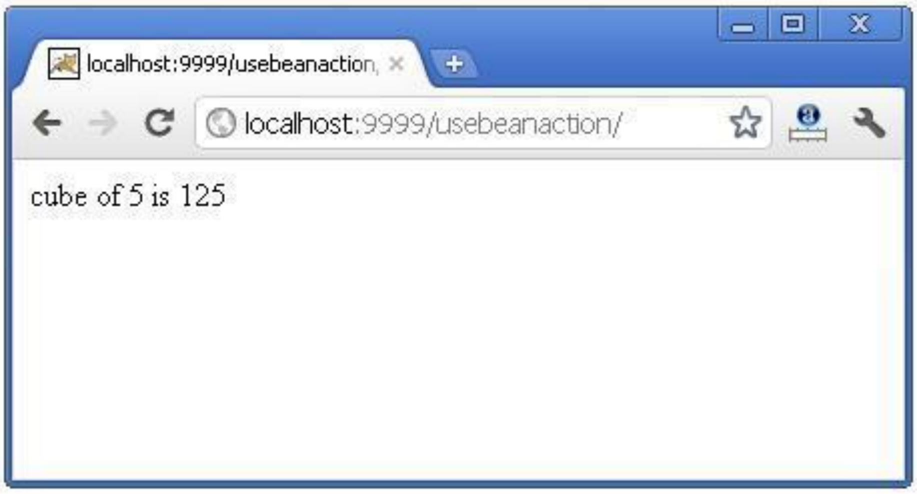
*For the example of setProperty, getProperty and useBean tags, visit next page.*

Calculator.java (a simple Bean class)

```
package com;
public class Calculator{
public int cube(int n){return n*n*n;}
}
```

index.jsp file

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

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



```
cube of 5 is 125
```

**jsp:setProperty and jsp:getProperty action tags**

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

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

Syntax of jsp:setProperty action tag

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

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

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

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

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

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

```
<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

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

Simple example of jsp:getProperty action tag

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

---

**Example of bean development in JSP**

In this example there are 3 pages:

index.html for input of values

welocme.jsp file that sets the incoming values to the bean object and prints the one value

User.java bean class that have setter and getter methods

index.html

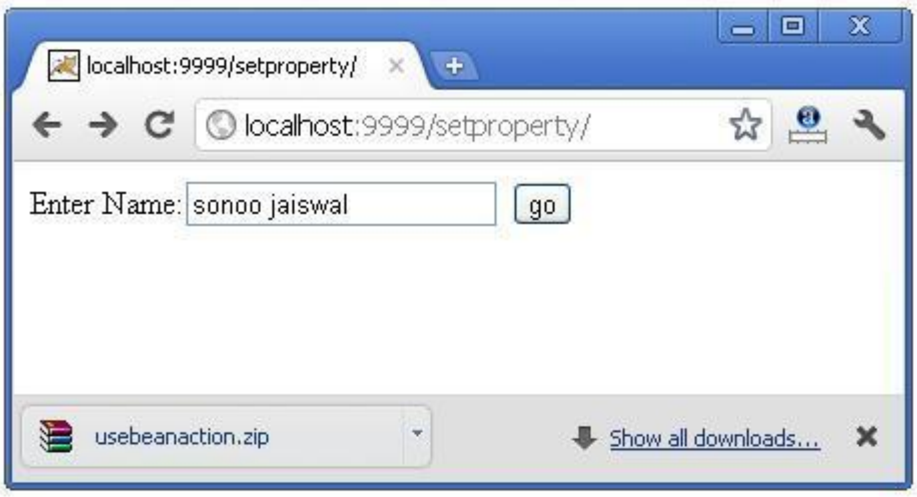
```
<form action="process.jsp" method="post">
Name:<input type="text" name="name"><br>
Password:<input type="password" name="password"><br>
Email:<input type="text" name="email"><br>
<input type="submit" value="register">
</form>
```

process.jsp

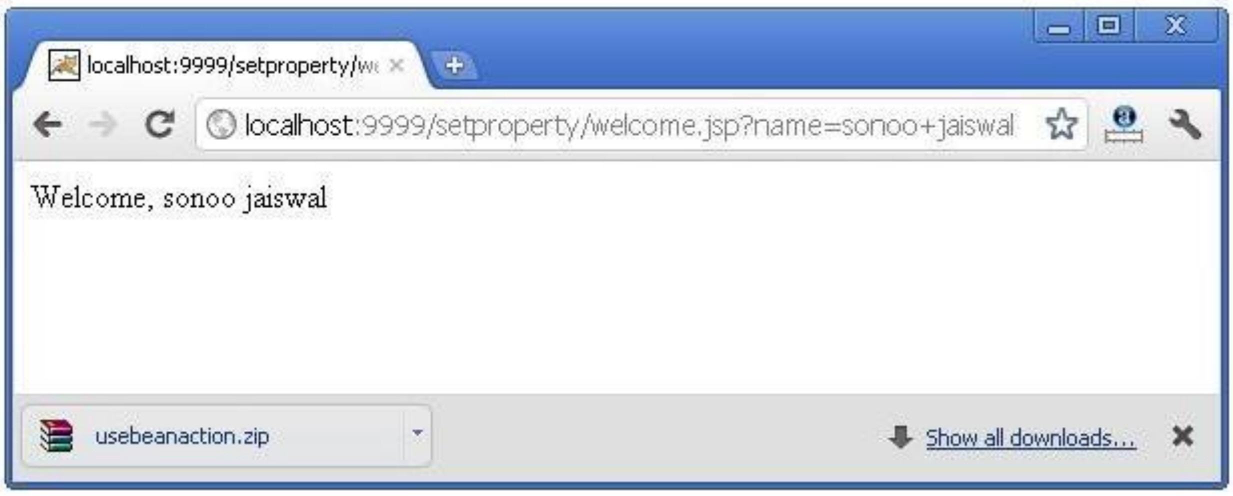
```
<jsp:useBean id="u" class="org.sssit.User"></jsp:useBean>
<jsp:setProperty property="*" name="u"/>

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

User.java

```
package org.sssit;

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



Enter Name: sonoo jaiswal go


localhost:9999/setproperty/w
localhost:9999/setproperty/welcome.jsp?name=sonoo+jaiswal
usebeanaction.zip
Show all downloads...

Welcome, sonoo jaiswal

**Displaying applet in JSP (jsp:plugin action tag)**
The jsp:plugin action tag is used to embed applet in the jsp file. The jsp:plugin action tag downloads plugin at client side to execute an applet or bean.
Syntax of jsp:plugin action tag
**<jsp:plugin** type= "applet | bean" code= "nameOfClassFile"
codebase= "directoryNameOfClassFile"
**</jsp:plugin>**

---

## <u>Expression Language (EL) in JSP</u>

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)
${ 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**

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

**process.jsp**
```
Welcome, ${ param.name }
```

**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**

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

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

**process.jsp**

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