



JSP (JAVA Server Pages)

 Uday Sharma

 mrudaysharma4600@gmail.com

- **JSP (Java Server Pages):-**

- ✓ It stands for Java Server Pages.
- ✓ It is a server-side technology,
- ✓ It is used for creating web application.
- ✓ It is used to create dynamic web content.
- ✓ JSP tags are used to insert Java code into HTML pages.
- ✓ It is an advanced version of Servlets.
- ✓ In this, java code can be inserted in HTML/XML pages or both.
- ✓ JSP is first converted into servlet by JSP container before processing the client's request.

- **Advantages JSP over Servlets:-**

- ✓ Easy to maintain.
- ✓ No recompilation or redeployment is required.
- ✓ JSP has accessed to entire API of java.
- ✓ JSP are extended version of Servlet.

- **Features of JSP:-**

- ✓ **Coding in JSP is easy:** - As it is just adding JAVA code to HTML/XML.
- ✓ **Reduction in the length of Code:** - In JSP we use action tags, custom tags etc.
- ✓ **Connection to Database is easier:** - It is easier to connect website to database and allows to read or write data easily to the database.
- ✓ **Make Interactive websites:** - In this we can create dynamic web pages which helps user to interact in real time environment.
- ✓ **Portable, Powerful, flexible and easy to maintain:** - as these are browser and server independent.
- ✓ **No Redeployment and No Re-Compilation:** - It is dynamic, secure and platform independent so no need to re-compilation.
- ✓ **Extension to Servlet:** - as it has all features of servlets, implicit objects and custom tags

- **Advantages of using JSP: -**

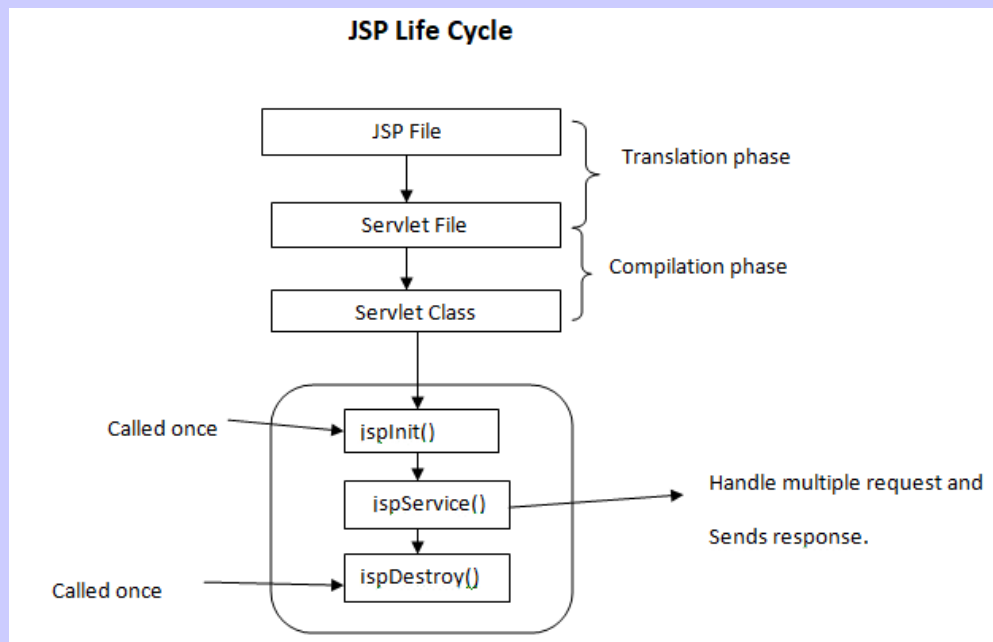
- ✓ It does not require advanced knowledge of JAVA.
- ✓ It is capable of handling exceptions.
- ✓ Easy to use and learn.
- ✓ It can tags which are easy to use and understand.
- ✓ Implicit objects are there which reduces the length of code.
- ✓ It is suitable for both JAVA and non-JAVA programmer

- **Disadvantages of using JSP: -**

- Difficult to debug for errors.
- First time access leads to wastage of time
- Its output is HTML which lacks features.

- **JSP life cycle: -**

- A Java Server Page life cycle is defined as the process that started with its creation which later translated to a servlet and afterward servlet lifecycle comes into play. This is how the process goes on until its destruction.
- JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.
- A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

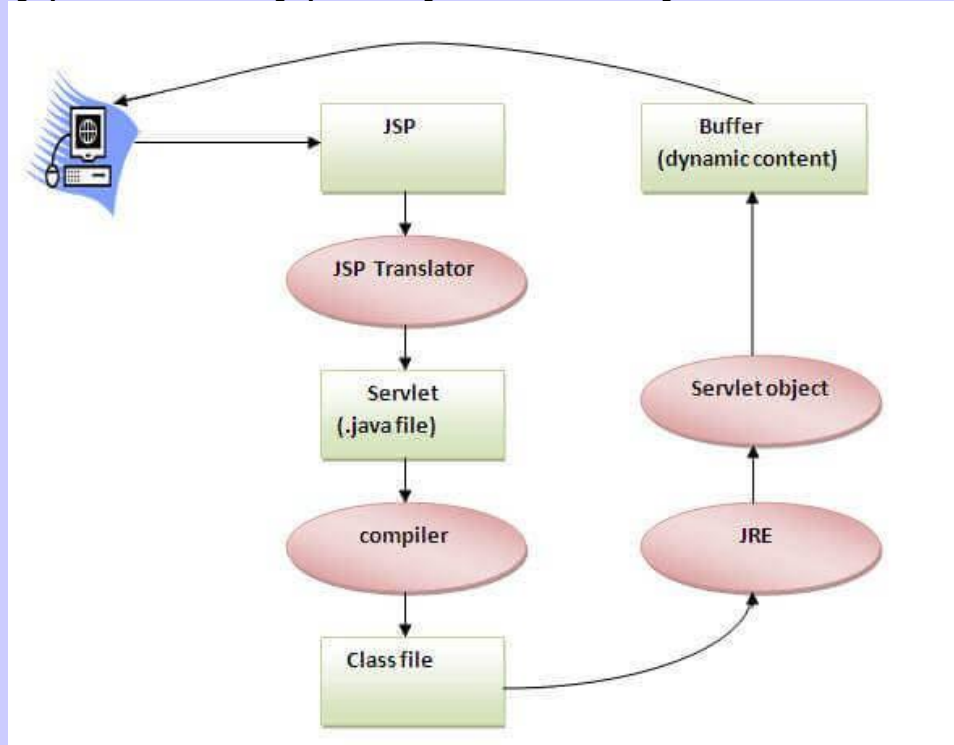


• ***The Lifecycle of a JSP Page :-***

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).

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 the web server which is responsible for translating the

JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are 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 saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

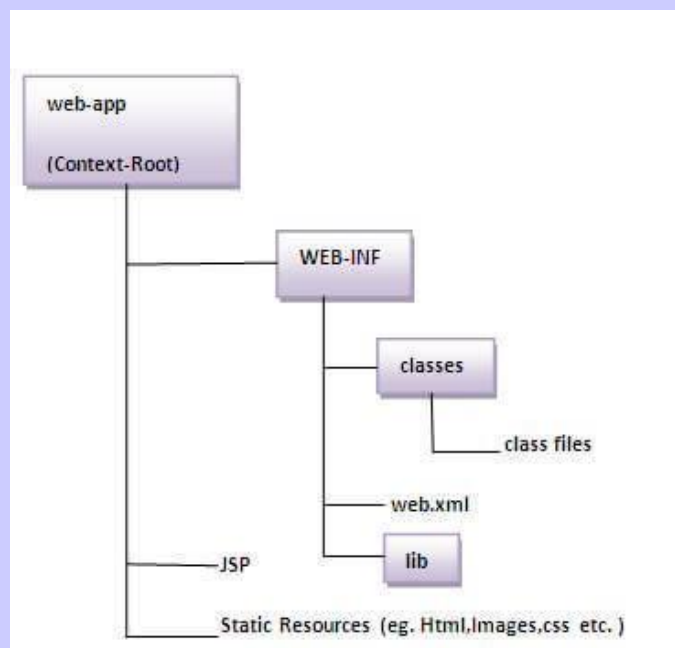
index.jsp

Example of scriptlet tag later.

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

- **The Directory structure of JSP:-**

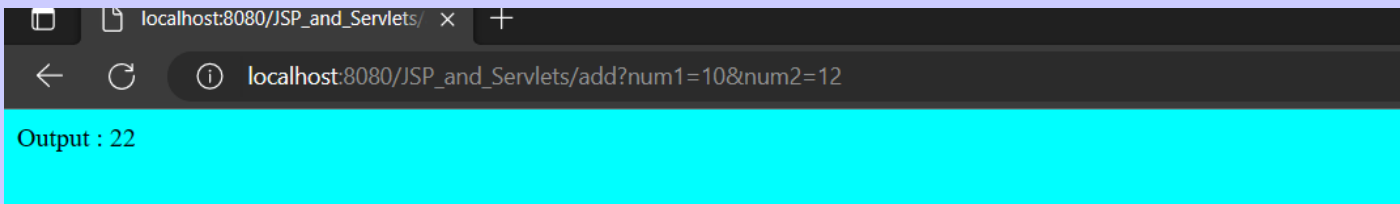
- The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



- **JSP ANNOTATION:-**

After entering the values of num1 and num2 is 10 and 12...

```
AddServlets.java x index.html x Square_servlet.java
1 package com.udaysharma;
2 import java.io.IOException;
12 @WebServlet("/add")
13 public class AddServlets extends HttpServlet {
14     public void doGet(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException {
15         // we need to convert the String into integer
16         int i=Integer.parseInt(req.getParameter("num1"));
17         int j=Integer.parseInt(req.getParameter("num2"));
18         // performing the add operation
19         int k = i+j;
20         PrintWriter out=resp.getWriter();
21         out.println("<html><body bgcolor='cyan'>");
22         out.println("Output : "+k);
23         out.println("</body></html>");
24     }
25 }
```



• JSP Tags:-

- **Scriptlet:-** it provides the facility to insert java code inside the **jsp**.
- **Directive:-** if we want to import the package or some external package we use directive. In the other words we can also say that the jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.
JSP DIRECTIVES:
 - ✓ @page
 - ✓ @include
- **Declaration:** The JSP declaration tag is used to declare fields and methods.
- **Expression:** The code placed within JSP expression tag is written to the output stream of the response.

• SP Scriptlet tag (Scripting elements):-

1. Scripting elements
2. JSP scriptlet tag
3. Simple Example of JSP scriptlet tag
4. Example of JSP scriptlet tag that prints the user name

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:

<% java source code %>

• Example of JSP scriptlet tag:-

In this example, we are displaying a welcome message.

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

• JSP expression tag:-

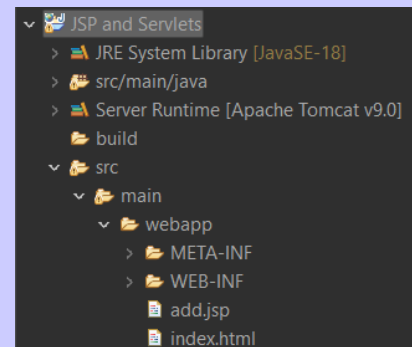
The code placed within JSP expression tag is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

• Syntax of JSP expression tag:-

<%= statement %>

• 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 %>**

We using the jsp file then first we have to create a jsp file .

So,in this we are not using the servlet:-

Code:- this example contains both the scriptlet ,Declaration and expression tag.

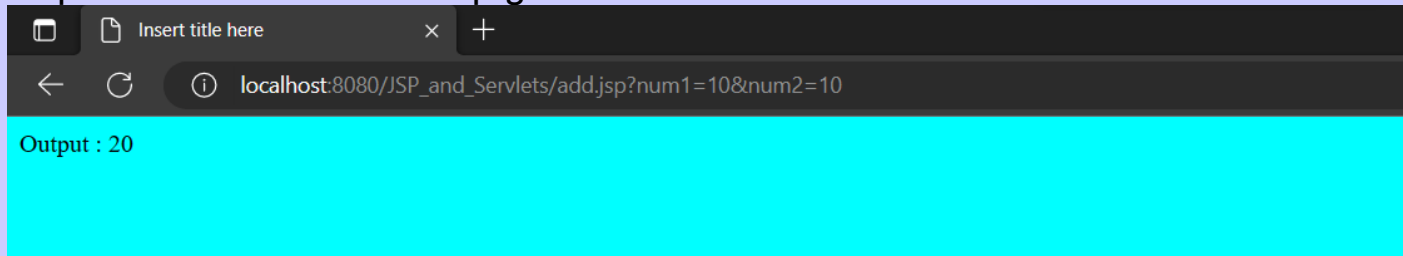
add.jsp:-

```
index.html  add.jsp ×
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6   <meta charset="UTF-8">
7   <title>Insert title here</title>
8 </head>
9 <body bgcolor="cyan">
10 <%
11     int i=Integer.parseInt(request.getParameter("num1"));
12     int j=Integer.parseInt(request.getParameter("num2"));
13     int k=i+j;
14     out.println("Output : "+k);
15     %>
16 </body>
17 </html>
```

index.html:-

```
index.html ×  add.jsp
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="UTF-8">
5   <title>Into of JSP and Servlets</title>
6 </head>
7 <body>
8   <h1>My name is uday Sharma am doing Btech CSE and Now Learning JSP and Servlets</h1>
9   <form action="add.jsp" method='Get'>
10     Enter the 1st number <input type="Text" name="num1"><br>
11     Enter the 2nd number <input type="Text" name="num2"><br>
12     <input type="submit">
13   </form>
14 </body>
15 </html>
```

Output:- on the client side server page or web browser.



- If any of the **data** which are **not** in the **service** of the page we can **write inside** of the **declaration tag** denoted as **<%! -----%>** .

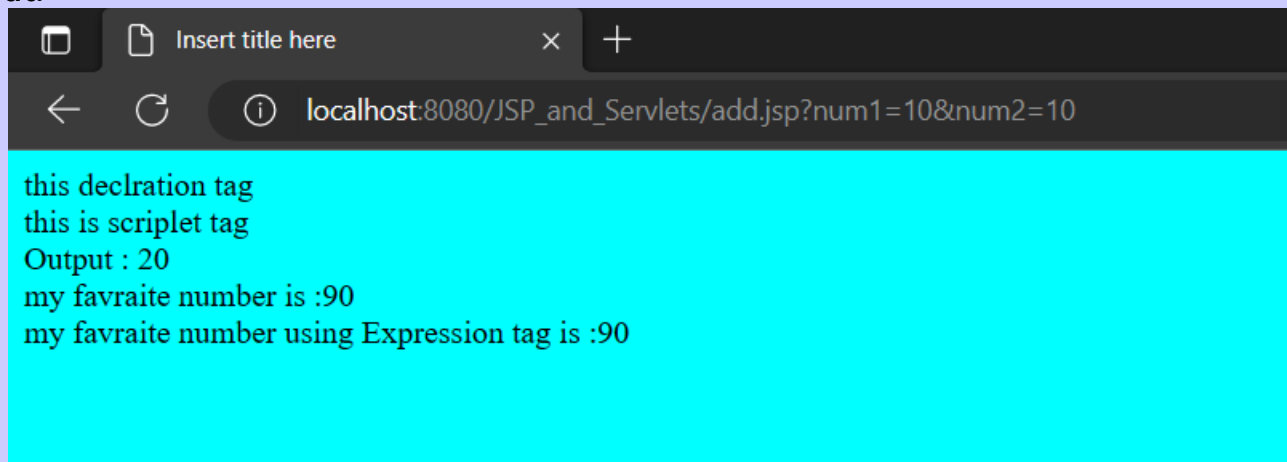
Ex:-

<%! Int values=90 %> and we can also make a method inside of these tag which go in the page.

- <%@ page import="java.util.*"%> this tag is called **directive tag** ..
- If we want to print something we can use <%= > this tag is called expression tag using to print the data

```
index.html  add.jsp x
1  <%@page import="java.util.*"%>
2  <%@ page language="java" contentType="text/html; charset=UTF-8"
3      pageEncoding="UTF-8"%>
4  <!DOCTYPE html>
5  <html>
6  <head>
7  <meta charset="UTF-8">
8  <title>Insert title here</title>
9  </head>
10 <body bgcolor="cyan">
11 this decleration tag<br>
12 <%!int coef=90 ;%>
13 this is scriplet tag<br>
14     <%
15         int i=Integer.parseInt(request.getParameter("num1"));
16         int j=Integer.parseInt(request.getParameter("num2"));
17         int k=i+j;
18         out.println("Output : "+k);
19     %>
20     <br>
21     my favraite number is :<%out.println(coef); %><br>
22     my favraite number using Expression tag is :<%= (coef) %>
23 </body>
24 </html>
```

Output:-



- ***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.

- **JSP directives:-**

1. **page directive:-**

2. **Attributes of page directive:-**

The **jsp directives** are messages that tell 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

```
language="scripting language"  
extends="className"  
import="importList"  
session="true|false"  
autoFlush="true|false"  
contentType="ctinfo"  
errorPage="error_url"  
isErrorPage="true|false"  
info="information"  
isELIgnored="true|false"  
isThreadSafe="true|false"
```

Example:-

```
<@page language="java" import="java.util.*,java.sql.*"  
info="Contact page" extends="com.telusko.User" contentType="text/  
html" isELIgnored="false" isThreadSafe="false" session="false"  
@Include
```

```
<%@ include file="filename" %>  
<%@ include file="header.jsp" %>
```


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>
3. <%@ page import="java.util.Date" %>
4. Today is: <%= new Date() %>
5. </body>
6. </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>
3. <%@ page contentType=application/msword %>
4. Today is: <%= new java.util.Date() %>
5. </body>
6. </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>
3. <%@ page info="composed by Sonoo Jaiswal" %>
4. Today is: <%= new java.util.Date() %>
5. </body>
6. </html>

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

1. **public** String getServletInfo() {
2. **return** "composed Uday Sharma ";
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>
3. <%@ page buffer="16kb" %>
4. Today is: <%= new java.util.Date() %>
5. </body>
6. </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>`
4. `<%@ page errorPage="myerrorpage.jsp" %>`
5. `<%= 100/0 %>`
6. `</body>`
7. `</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>`
4. `<%@ page isErrorPage="true" %>`
5. `Sorry an exception occurred!
`
6. `The exception is: <%= exception %>`
7. `</body>`
8. `</html>`

- sp Include Directive

1. Include directive
2. Advantage of Include directive
3. Example of 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**

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

Add.jsp:-

```
index.html  add.jsp ×  home.jsp
1 <%@page import="java.util.*"%>
2 <%@ page language="java" contentType="text/html; charset=UTF-8"
3     pageEncoding="UTF-8" %>
4
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <meta charset="UTF-8">
9 <title>Insert title here</title>
10 </head>
11 <body bgcolor="cyan">
12 this decleration tag<br>
13 <%!int coef=90 ;%>
14 this is scriplet tag<br>
15 <%
16     int i=Integer.parseInt(request.getParameter("num1"));
17     int j=Integer.parseInt(request.getParameter("num2"));
18     int k=i+j;
19     out.println("Output : "+k);
20     %>
21 <br>
22 my favraite number is :<%out.println(coef); %><br>
23 my favraite number using Expression tag is :<%=coef)%>
24 </body>
25 </html>
```

- **Home.jsp:-** in this jsp page we include the file “add.jsp” so this add.jsp will ccompute first:-

```

index.html  add.jsp  home.jsp ×
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8" import="java.sql.Statement,java.util.Random"%>
3 <%@ page import="java.util.ArrayList" %>
4 <%@include file="add.jsp" %>
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <meta charset="UTF-8">
9 <title>this is the home page of jsp</title>
10 </head>
11 <body>
12 <p>this is home page</p>
13 </body>
14 </html>

```

- **Index.html:-** using it to run on the **web browser**_save it and give the action to **home.jsp** with contains **add.jsp**:-

```

index.html ×  add.jsp  home.jsp
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Into of JSP and Servlets</title>
6 </head>
7 <body>
8 <h1>My name is uday Sharma am doing Btect CSE and Now Learning JSP and Servlets</h1>
9 <form action="home.jsp" method='Get'>
10 Enter the 1st number <input type="Text" name="num1"><br>
11 Enter the 2nd number <input type="Text" name="num2"><br>
12 <input type="submit">
13 </form>
14 </body>
15 </html>

```

- After save and restart the server and enter the 1st and 2nd number value 10 and enter submit button

Into of JSP and Servlets × +

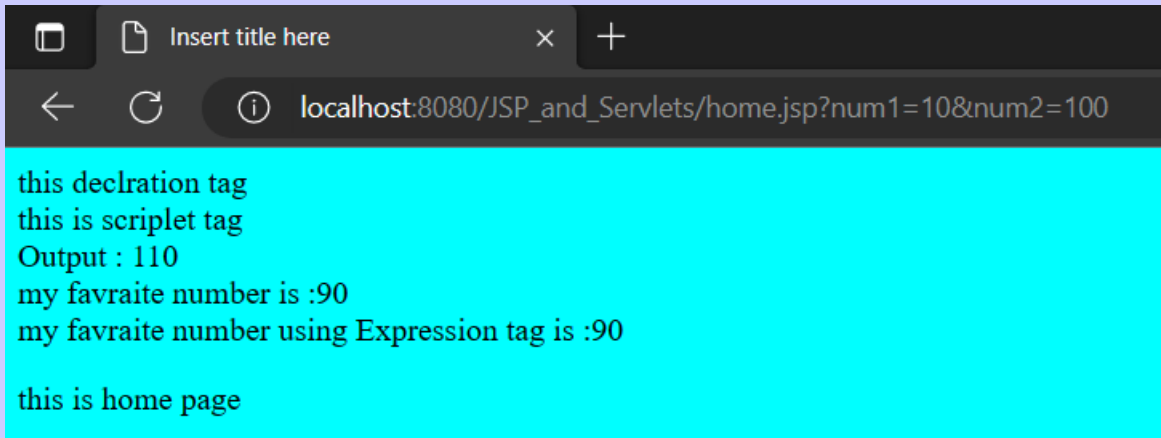
localhost:8080/JSP_and_Servlets/index.html

My name is uday Sharma am doing Btect CSE and Now Learning JSP and Servlets

Enter the 1st number

Enter the 2nd number

- After **submitting** we land on the **new webpage** and in **url** we can see we are on **home.jsp** page mean our home.jsp file which includes add.jsp run properly



- **JSP Taglib directive:-**

1. **JSP Taglib directive**
2. **Example of JSP Taglib directive**

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. 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.

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

Implicit Objects in JSP:-

Code:-

```
Builton Object (can be used in Scriptlet and Expression)

request (HttpServletRequest)

response (HttpServletResponse)

pageContext (PageContext)

out (JspWriter) ~ PrintWriter object

session (HttpSession)

application (ServletContext)

config (ServletConfig)
```

```

1 <%@page import="java.util.*" %>
2
3 <%@ page language="java" contentType="text/html; charset=UTF-8"
4   pageEncoding="UTF-8"%>
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <meta charset="UTF-8">
9 <title>Tags in Java</title>
10 </head>
11 <body>
12
13 <%!           //Declaration Tag - means to declare something
14   int a = 100;
15   %>
16
17 <h1>Hello to JSP</h1>
18
19 <%           //Scriptlet Tag - use to insert java code into jsp
20   out.println(9+1);
21   %>
22
23 My fav no is: <%= a%>           //Expression tag - means to print declared expressions//
24
25 </body>
26 </html>

```

• JSP Directives

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

✓ **@page:** The page directive is used to provide instructions to the container or to import libraries or external libraries.

✓ **Syntax:**

<%@ page attribute="value" %>

✓ **@include:** it is used to include a file during translation phase.

✓ **Syntax:**

<%@ include file="file name" %>

Code:- in the below example we using the @page in the jsp file to some of the main library:-

Tags.jsp:-

```

1 <%@page import="java.util.*" %>
2
3 <%@ page language="java" contentType="text/html; charset=UTF-8"
4   pageEncoding="UTF-8"%>
5 <!DOCTYPE html>
6 <html>
7 <head>
8 <meta charset="UTF-8">
9 <title>Tags in Java</title>
10 </head>
11 <body>
12
13 <%!           //Declaration Tag - means to declare something
14   int a = 100;
15   %>
16
17 <h1>Hello to JSP</h1>
18
19 <%           //Scriptlet Tag - use to insert java code into jsp
20   out.println(9+1);
21   %>
22
23 My fav no is: <%= a%>           //Expression tag - means to print declared expressions//
24
25 </body>
26 </html>

```

Home.jsp:-

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3
4 <%@ include file = "Tags.jsp" %>
5
6 <!DOCTYPE html>
7 <html>
8 <head>
9 <meta charset="UTF-8">
10 <title>Insert title here</title>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

- **JSP implicit objects:-**

- There are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages.
- The available implicit objects are out, request, config, session, application etc.

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

- **JSP out implicit object:-**

In servlet we write:

PrintWriter out = response.getWriter();

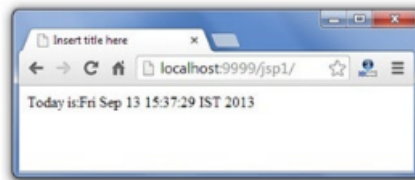
Example of out implicit object

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



- ***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.

Example of JSP request implicit object

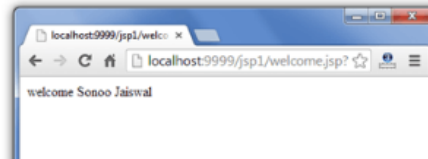
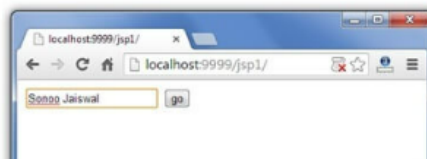
index.html

```
<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



- ***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.

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



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>
```

- **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**.

welcome.jsp

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

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

Output



- **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>
```

- **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:

- o **page**
- o **request**
- o **session**

application

- **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">

6. </form>
7. </body>
8. </html>

- welcome.jsp:-

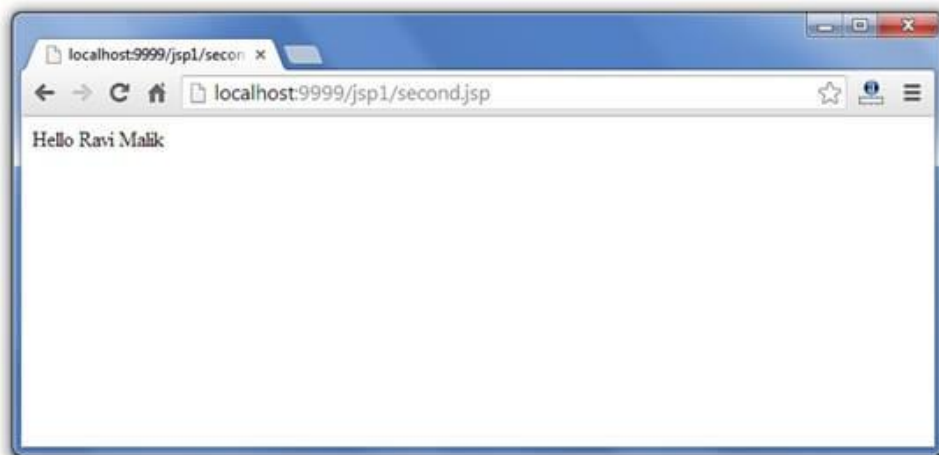
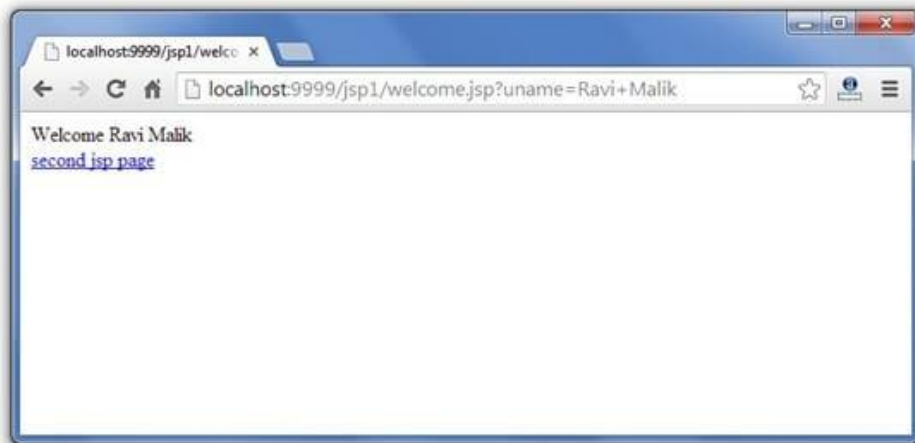
1. <html>
2. <body>
3. <%
4. String name=request.getParameter("uname");
5. out.print("Welcome "+name);
6. pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
7. second jsp page
8. %>
9. </body>
10. </html>

- second.jsp:-

1. <html>
2. <body>
3. <%
- 4.
5. String name=(String)pageContext.getAttribute("user",PageContext.SESSION_SCOPE);
6. out.print("Hello "+name);
7. %>
8. </body>
9. </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:

```
<% (HttpServletRequest)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>`
4. `Sorry following exception occurred:<%= exception %>`
5. `</body>`
6. `</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:

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

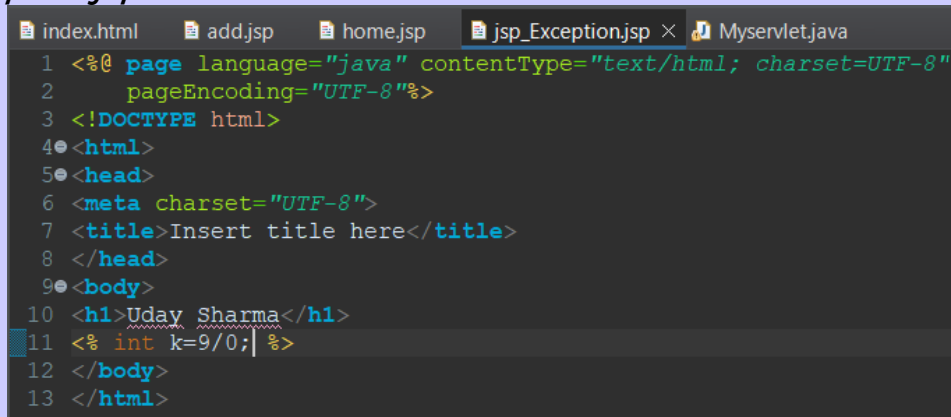
- **Example of exception handling in jsp by the elements of page directive**

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.

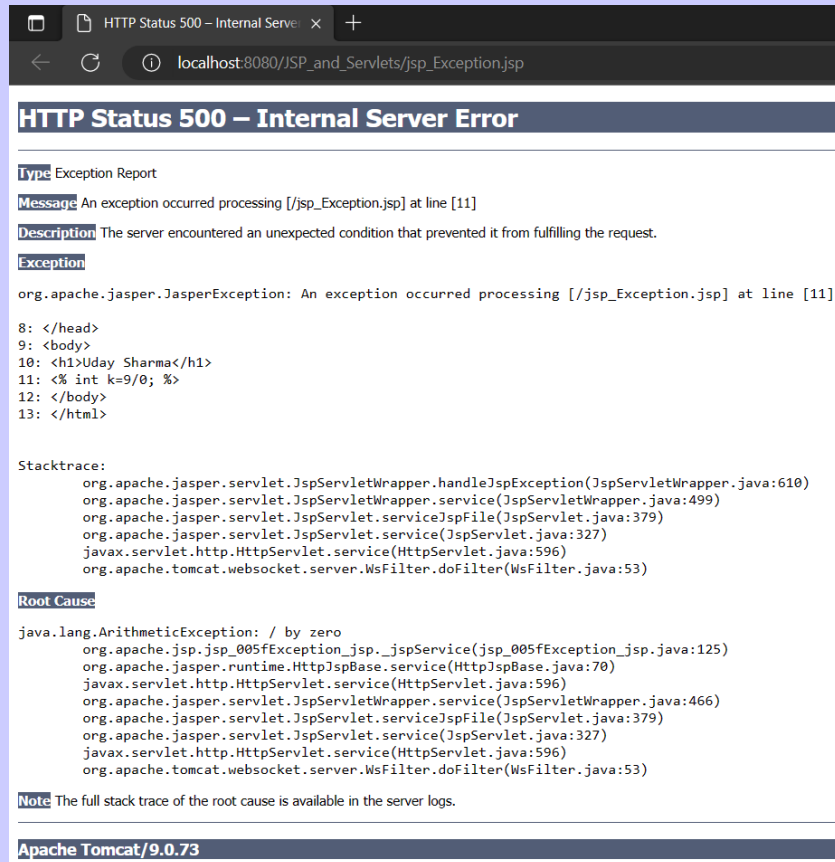
There are 3 files:

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

Code:- jsp_Exception.jsp:-



```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body>
10 <h1>Uday Sharma</h1>
11 <% int k=9/0;| %>
12 </body>
13 </html>
```

Using an example of Exception in Jsp :-

Index.html:-

```
index.html x home.jsp jsp_Exception.jsp error.jsp add.jsp
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta charset="UTF-8">
5 <title>Into of JSP and Servlets</title>
6 </head>
7 <body>
8 <h1>My name is uday Sharma am doing Btect CSE and Now Learning JSP and Servlets</h1>
9 <form action="jsp_Exception.jsp" method='Get'>
10 Enter the 1st number <input type="Text" name="num1"><br>
11 Enter the 2nd number <input type="Text" name="num2"><br>
12 <input type="submit">
13 </form>
14 </body>
15 </html>
```

- Jsp_Exception.jsp:-
- We using the errorpage to goto the errorpage:-

```
index.html home.jsp jsp_Exception.jsp × error.jsp add.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8" errorPage="error.jsp"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body>
10 <h1>Uday Sharma</h1>
11 <%
12     int i=Integer.parseInt(request.getParameter("num1"));
13     int j=Integer.parseInt(request.getParameter("num2"));
14     int k=i/j;
15     out.println("Output : "+k);
16 %>
17 </body>
18 </html>
```

- **error.jsp:-**
- Declaring this page as a **error page:-**

```
index.html home.jsp jsp_Exception.jsp error.jsp × add.jsp
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8" isErrorPage="true"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body bgcolor="Red">
10 <h1>Error</h1>
11 Exception<%=exception %>
12 </body>
13 </html>
```

- Output:- after save and restart the server we and enter the values of the 1st and 2nd number .

Insert title here × Into of JSP and Servlets × +

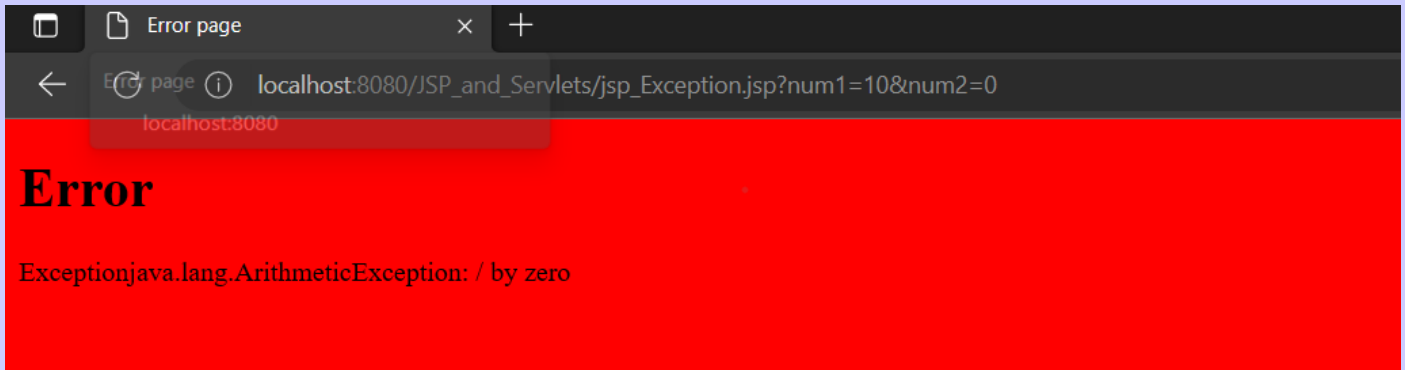
localhost:8080/JSP_and_Servlets/index.html

My name is uday Sharma am doing Btect CSE and Now Learning JSP and Servlets

Enter the 1st number

Enter the 2nd number

- We enter the value of 10 and 0 so the data goes the **jsp_Exception.jsp** file. And we can see in the **url** we are in the **error page** because 10/0 gives error. Because the **jsp_Exception.jsp** file the includes the error page:-

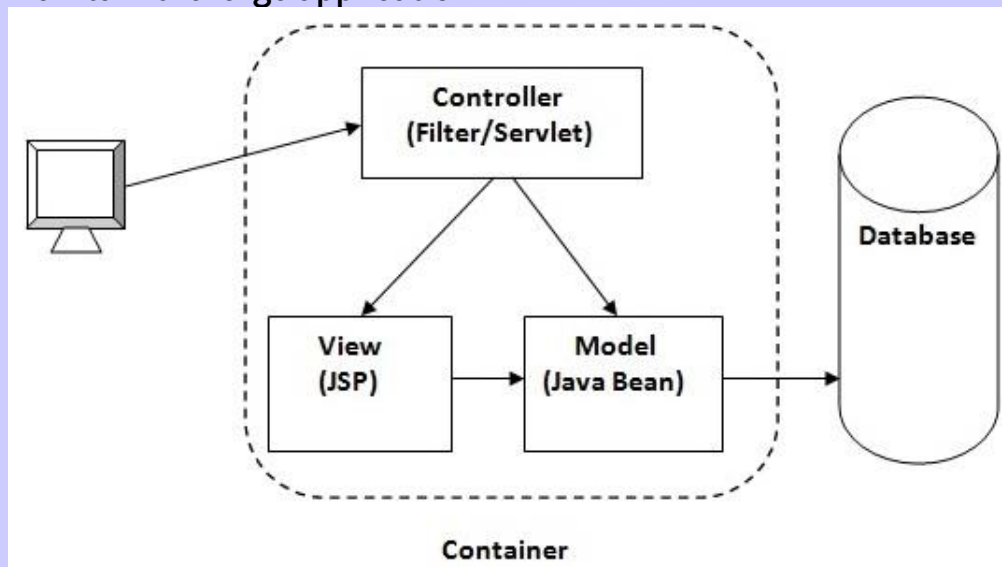


• MVC in JSP:-

1. MVC in JSP

2. Example of following MVC in JSP

- **MVC** stands for Model View and Controller. It is a **design pattern** that separates the business logic, presentation logic and data.
- **Controller** acts as an interface between View and Model. Controller intercepts all the incoming requests.
- **Model** represents the state of the application i.e. data. It can also have business logic.
- **View** represents the presentation 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 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.

Example:- Making a Login page:-

- **JSTL (JSP Standard Tag Library):-**

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development. For using the **jstl library** we need the required **JAR file** . so, we have to **download the jar file** “Jstl.jar file”.

- **Advantage of JSTL :-**

1. **Fast Development** JSTL provides many tags that simplify the JSP.
2. **Code Reusability** We can use the JSTL tags on various pages.
3. **No need to use scriptlet tag** It avoids the use of scriptlet tag.

- **JSTL Tags :-**

There JSTL mainly provides five types of tags:

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

- For creating JSTL application, you need to load the **jstl.jar** file. After download the jar file we need the to built the class path in our java project.

- **DemoServlet.java:-**

```
DemoServlet.java × display.jsp
1 package com.udaysharma;
2
3 import java.io.IOException;
11 @WebServlet("/DemoServlet")
12 public class DemoServlet extends HttpServlet {
13     protected void doGet(HttpServletRequest request, HttpServletResponse response ) throws ServletException, IOException {
14         // JSTL--> JSP Standard tag library
15         String name="uday";
16         request.setAttribute("label", name);
17         RequestDispatcher rd=request.getRequestDispatcher("display.jsp");
18         rd.forward(request, response);
19     }
20 }
21
```

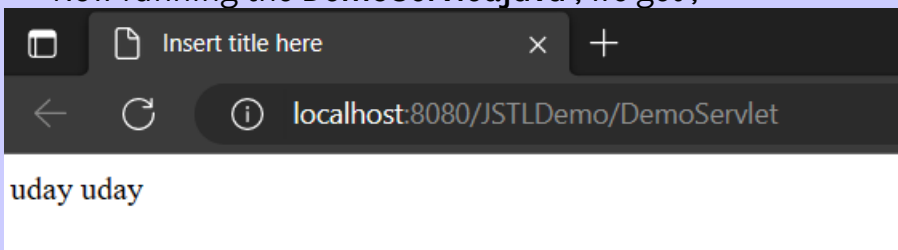
- **Display.jsp:-**

```

DemoServlet.java  display.jsp x
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body>
10 <%
11 String name=request.getAttribute("label").toString();
12 out.println(name);
13 %>
14 ${label}
15 </body>
16 </html>

```

- Now running the **DemoServlet.java** , we get :-



Note:- If we want to use the JSTL library we have to build the path in the WEB-INF=>Lib=>jstl.jar 1.2

Or Copy the file.

• JSTL Core Tags:-

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

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

- JSTL Core Tags List:-

Tags	Description
<u>c:out</u>	It display the result of an expression, similar to the way <%=...%> tag work.
<u>c:import</u>	It Retrives relative or an absolute URL and display the contents to either a String in 'var',a Reader in 'varReader' or the page.(fetch the website)
<u>c:set</u>	It sets the result of an expression under evaluation in a 'scope' variable.
<u>c:remove</u>	It is used for removing the specified scoped variable from a particular scope.
<u>c:catch</u>	It is used for Catches any Throwable exceptions that occurs in the body.
<u>c:if</u>	It is conditional tag used for testing the condition and display the body content only if the expression evaluates is true.
<u>c:choose, c:when, c:otherwise</u>	It is the simple conditional tag that includes its body content if the evaluated condition is true.

<u>c:forEach</u>	It is the basic iteration tag. It repeats the nested body content for fixed number of times or over collection.
<u>c:forTokens</u>	It iterates over tokens which is separated by the supplied delimiters.
<u>c:param</u>	It adds a parameter in a containing 'import' tag's URL.
<u>c:redirect</u>	It redirects the browser to a new URL and supports the context-relative URLs.
<u>c:url</u>	It creates a URL with optional query parameters.

• ***JSTL Core <c:out> Tag :-***

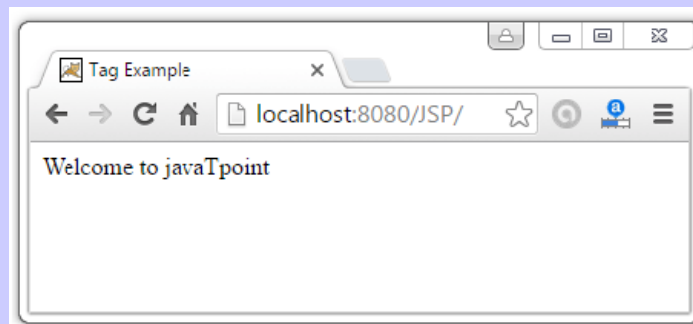
- The **<c:out>** tag is similar to JSP expression tag, but it can only be used with expression. It will display the result of an expression, similar to the way **<%=...%>** work.
- The **<c:out>** tag automatically escape the **XML tags**. Hence they aren't evaluated as actual tags.

Let's see the simple example of c:out tag:-

1. **<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>**
2. **<html>**
3. **<head>**
4. **<title>Tag Example</title>**
5. **</head>**
6. **<body>**
7. **<c:out value="\${'Welcome to javaTpoint'}"/>**
8. **</body>**
9. **</html>**

Output:

Using the core taglib :-



- DemoServlet.java:-

Using the **request Dispatcher** we **set the Attributes** of the String name by givin the value **“label”**:-

```

DemoServlet.java x display.jsp
1 package com.udaysharma;
2
3 import java.io.IOException;
11 @WebServlet("/DemoServlet")
12 public class DemoServlet extends HttpServlet {
13     protected void doGet(HttpServletRequest request,HttpServletResponse response ) throws ServletException,
14         // JSTL--> JSP Standard tag library
15         String name="uday";
16         request.setAttribute("label", name);
17         RequestDispatcher rd=request.getRequestDispatcher("display.jsp");
18         rd.forward(request, response);
19     }
20 }
  
```

- Get the attributes on the **display.jsp** file .

So,we can using the **Scriptlet tag** or we can use the **<c:out>** Given by the **core tag** from JSTL We denoting the label value we can access by using **notation** represented by **“\${}”**.

```
DemoServlet.java display.jsp ×
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
4 <!DOCTYPE html>
5 <html>
6 <head>
7 <meta charset="UTF-8">
8 <title>Insert title here</title>
9 </head>
10 <body>
11 <%
12 String name=request.getAttribute("label").toString();
13 out.println(name);
14 %>
15 ${label}<br>
16 <c:out value="hello world"/><br>
17 <c:out value="${label}" />
18 <c:import url="http://www.google.com"></c:import>
19 </body>
20 </html>
```

Insert title here × +

localhost:8080/JSTLDemo/DemoServlet

uday uday
hello world
uday

[Search](#) [Images](#) [Maps](#) [Play](#) [YouTube](#) [News](#) [Gmail](#) [Drive](#) [More »](#)

[Web History](#) | [Settings](#)

Google

Advanced search

Google Search I'm Feeling Lucky

Google offered in: [हिन्दी](#) [বাংলা](#) [తెలుగు](#) [मराठी](#) [தமிழ்](#) [ગુજરાતી](#) [ಕನ್ನಡ](#) [മലയാളം](#) [ਪੰਜਾਬੀ](#)

[Advertising](#) [Business Solutions](#) [About Google](#) [Google.co.in](#)

© 2023 - [Privacy](#) - [Terms](#)

Ex:-

- Make Another java file name Student.java using the getter and setter attribute we define roll_no and name :-


```

Student.java x DemoServlet.java x display.jsp
1 package com.udaysharma;
2
3 public class Student {
4     String name;
5     public String getName() {
6         return name; }
7     public void setName(String name) {
8         this.name = name; }
9     public int getRollno() {
10        return rollno; }
11    public void setRollno(int rollno) {
12        this.rollno = rollno; }
13    int rollno;
14    public Student(int rollno, String name) {
15        super();
16        this.rollno = rollno;
17        this.name = name;
18    }
19    @Override
20    public String toString() {
21        return "Student [name=" + name + ", rollno=" + rollno + "]";
22    }
23 }

```

```

Student.java x DemoServlet.java x display.jsp
1 package com.udaysharma;
2
3 import java.io.IOException;
4 import java.util.Arrays;
5 import java.util.List;
6
7 import javax.servlet.RequestDispatcher;
8 import javax.servlet.ServletException;
9 import javax.servlet.annotation.WebServlet;
10 import javax.servlet.http.HttpServlet;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13 @WebServlet("/DemoServlet")
14 public class DemoServlet extends HttpServlet {
15     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
16         // JSTL--> JSP Standard tag library
17         String name="uday";
18         Student s=new Student(1,"uday Sharma");
19         // we can also create a list
20         request.setAttribute("Student",s);
21     // request.setAttribute("label", name);
22     RequestDispatcher rd=request.getRequestDispatcher("display.jsp");
23     rd.forward(request, response);
24 }
25 }

```

```

Student.java x DemoServlet.java x display.jsp x
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
4 <!DOCTYPE html>
5 <html>
6 <head>
7     <meta charset="UTF-8">
8     <title>Insert title here</title>
9 </head>
10 <body>
11 <%
12     String name=request.getAttribute("Student").toString();
13     out.println(name);
14 %>
15 ${Student.name}<br>
16 <c:out value="hello world"/><br>
17 <c:out value="${Student.name}"/><br>
18 <c:out value="${Student.rollno}"/><br>
19 <c:out value="${Student.toString()}" />
20 <c:import url="http://www.google.com"></c:import>
21 </body>
22 </html>

```

- Ex:- using the array list to Access multiple values:-

```

Student.java DemoServlet.java x display.jsp
1 package com.udaysharma;
2
3 import java.io.IOException;
4 import java.util.Arrays;
5 import java.util.List;
6
7 import javax.servlet.RequestDispatcher;
8 import javax.servlet.ServletException;
9 import javax.servlet.annotation.WebServlet;
10 import javax.servlet.http.HttpServlet;
11 import javax.servlet.http.HttpServletRequest;
12 import javax.servlet.http.HttpServletResponse;
13 @WebServlet("/DemoServlet")
14 public class DemoServlet extends HttpServlet {
15     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
16         // JSTL--> JSP Standard tag library
17         String name="uday";
18         request.setAttribute("label", name);
19         Student s=new Student(1,"uday Sharma");
20         request.setAttribute("Student",s);
21         // we can also create a list
22         List<Student> studs=Arrays.asList(new Student(1,"uday Sharma"),new Student(2,"yashas"),new Student(3,"Anshul"));
23         request.setAttribute("students", studs);
24         RequestDispatcher rd=request.getRequestDispatcher("display.jsp");
25         rd.forward(request, response);
26     }
27 }

```

```

Student.java DemoServlet.java x display.jsp x
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
4 <!DOCTYPE html>
5 <html>
6 <head>
7 <meta charset="UTF-8">
8 <title>Insert title here</title>
9 </head>
10 <body>
11 ${stduents}
12 <br>
13 <c:out value="hello world"/><br>
14 <c:out value="${students}"/><br>
15 <c:forEach items="${students}" var="s">
16 ${s}<br>${s.name}<br>${s.rollno}<br>
17 </c:forEach>
18 </body>
19 </html>

```

Insert title here

localhost:8080/JSTLDemo/DemoServlet

```

hello world
[Student [name=uday Sharma, rollno=1], Student [name=yashas, rollno=2], Student [name=Anshul, rollno=3]]
Student [name=uday Sharma, rollno=1]
uday Sharma
1
Student [name=yashas, rollno=2]
yashas
2
Student [name=Anshul, rollno=3]
Anshul
3

```

• JSTL SQL Tags:-

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

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

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

- JSTL SQL Tags List:-***

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

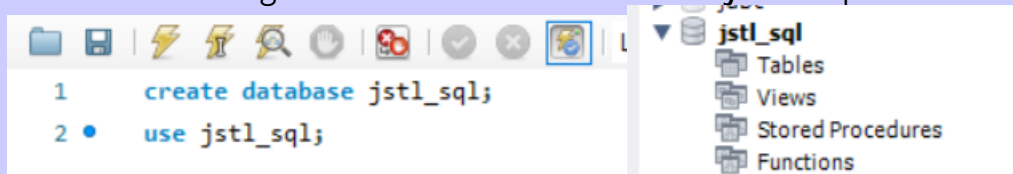
- JSTL SQL <sql:setDataSource> Tag :-***

- The **<sql:setDataSource>** tag is used for creating a simple data source suitable only for prototyping.
- It is used to create the data source variable directly from JSP and it is stored inside a scoped variable. It can be used as input for other database actions.

- Example:-***

Consider the below information about your MySQL database setup:

- We are using the **JDBC MySQL driver**
- We are using the **test** database on local machine
- We are using the **"root"** as username and **"uday123"** as password to access the test database.



- JSTL SQL <sql:query> Tag:-***

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

Example:

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

```

display.jsp x
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3 <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
4 <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
5 <!DOCTYPE html>
6 <html>
7 <head>
8   <meta charset="UTF-8">
9   <title>SQL Connection in the JSTL</title>
10 </head>
11 <body>
12 <sql:setDataSource var="db" driver="com.mysql.cj.jdbc.Driver" url="jdbc:mysql://localhost:3306/jdbc" user="root" password="ud"
13 <sql:query var="rs" dataSource="${db}">select * from stu;</sql:query>
14 <table border="2" width="50%">
15 <tr>
16 <th>ID</th>
17 <th>Name</th>
18 <th>City</th>
19 </tr>
20 <c:forEach items="${rs.rows}" var="stu">
21 <tr>
22 <td><c:out value="${stu.id}"></c:out></td>
23 <td><c:out value="${stu.Name}"></c:out></td>
24 <td><c:out value="${stu.city}"></c:out></td>
25 </tr>
26 </c:forEach>
27 </table>
28 </body>
29 </html>

```

SQL Connection in the JSTL

localhost:8080/JSTLDemo/display.jsp

ID	Name	City
1	Uday Sharma	Roorkee
2	Anshul Lakher	Kashipur
3	Yash Kapoor	Saharanpur
4	Yashas Gupta	Jawalapur

• **JSTL Function Tags:-**

- The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions. The syntax used for including JSTL function library in your JSP is:
 <%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

• **JSTL Function Tags List:-**

JSTL Functions	Description
<u>fn:contains()</u>	It is used to test if an input string containing the specified substring in a program.
<u>fn:containsIgnoreCase()</u>	It is used to test if an input string contains the specified substring as a case insensitive way.
<u>fn:endsWith()</u>	It is used to test if an input string ends with the specified suffix.
<u>fn:escapeXml()</u>	It escapes the characters that would be interpreted as XML markup.
<u>fn:indexOf()</u>	It returns an index within a string of first occurrence of a specified substring.
<u>fn:trim()</u>	It removes the blank spaces from both the ends of a string.

<u>fn:startsWith()</u>	It is used for checking whether the given string is started with a particular string value.
<u>fn:split()</u>	It splits the string into an array of substrings.
<u>fn:toLowerCase()</u>	It converts all the characters of a string to lower case.
<u>fn:toUpperCase()</u>	It converts all the characters of a string to upper case.
<u>fn:substring()</u>	It returns the subset of a string according to the given start and end position.
<u>fn:substringAfter()</u>	It returns the subset of string after a specific substring.
<u>fn:substringBefore()</u>	It returns the subset of string before a specific substring.
<u>fn:length()</u>	It returns the number of characters inside a string, or the number of items in a collection.
<u>fn:replace()</u>	It replaces all the occurrence of a string with another string sequence.

- ***Example:-***

```

jstl_function.jsp ×
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2   pageEncoding="UTF-8"%>
3   <%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions" %>
4   <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
5 <!DOCTYPE html>
6<html>
7<head>
8  <meta charset="UTF-8">
9  <title>JSTL FUNCTION</title>
10 </head>
11<body>
12 <c:set var="str" value="My name is Uday Sharma"/>
13 Length : ${fn:length(str)}<br>
14 Index : ${fn:indexOf(str,"is")}<br>
15 is There : ${fn:contains(str,"Uday")}<br>
16 is There : ${fn:contains(str,"JSP")}<br>
17 <c:if test="${fn:contains(str,'Sharma')}">Yes is there</c:if><br>
18<c:forEach items="${fn:split(str,' ')}" var="split">
19 ${split}<br>
20 </c:forEach>
21
22 </body>
23 </html>

```

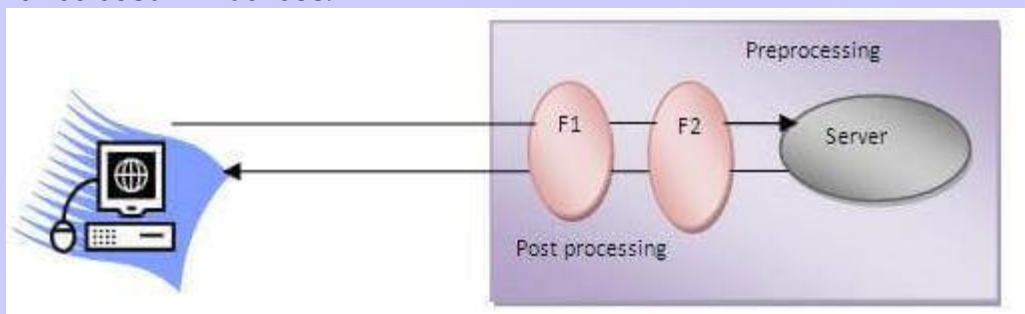
- ***Output:- After save and restarting the server :-***

```
JSTL FUNCTION
localhost:8080/JSTLDemo/jstl_function.jsp
Length : 22
Index : 8
is There : true
is There : false
Yes is there
My
name
is
Uday
Sharma
```

- **Servlet Filter:-**

1. Filter
2. Usage of Filter
3. Advantage of Filter
4. Filter API
 1. Filter interface
 2. FilterChain interface
 3. FilterConfig interface
5. Simple Example of Filter

- A **filter** is an object that is invoked at the preprocessing and postprocessing of a request.
- It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.
- The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.
- So maintenance cost will be less.



- **Note: Unlike Servlet, One filter doesn't have dependency on another filter.**
- **Usage of Filter :-**
 - recording all incoming requests
 - logs the IP addresses of the computers from which the requests originate
 - conversion
 - data compression
 - encryption and decryption
 - input validation etc.

- **Advantage of Filter :-**

1. Filter is pluggable.
2. One filter don't have dependency onto another resource.
3. Less Maintenance

- **Filter API :-**

- Like servlet filter have its own API. The **javax.servlet** package contains the three interfaces of Filter API.

1. Filter
2. FilterChain
3. FilterConfig

1) Filter interface :-

- For creating any filter, you must implement the Filter interface. Filter interface provides the life cycle methods for a filter.

Method	Description
public void init(FilterConfig config)	init() method is invoked only once. It is used to initialize the filter.
public void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain)	doFilter() method is invoked every time when user request to any resource, to which the filter is mapped. It is used to perform filtering tasks.
public void destroy()	This is invoked only once when filter is taken out of the service.

2) FilterChain interface :-

- The object of **FilterChain** is responsible to invoke the next filter or resource in the chain. This object is passed in the **doFilter** method of **Filter interface**. The **FilterChain** interface contains only one method:
 1. **public void doFilter(HttpServletRequest request, HttpServletResponse response):** it passes the control to the next filter or resource.

Code:- making a FirstfilterName.java in which Filter is auto-generate by the Eclipse IDE


```

*FirstFilterName.java x FirstFilterID.java home.jsp
1 package com.udaysharma;
2 import java.io.IOException;
14
15 @WebFilter("/MyServlet")
16 public class FirstFilterName extends HttpFilter implements Filter {
17     public FirstFilterName() {
18         super();
19     }
20     public void destroy() {
21     }
22     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
23         System.out.println("hi This is Filter 2");
24         PrintWriter out=response.getWriter();
25         HttpServletRequest req=(HttpServletRequest) request;
26         String aname=request.getParameter("aname");
27         if(!aname.isEmpty())
28             chain.doFilter(request, response);
29         else
30             out.print("Invalid input name is must");
31     }
32
33     /**
34      * @see Filter#init(FilterConfig)
35      */
36     public void init(FilterConfig fConfig) throws ServletException {
37         // TODO Auto-generated method stub
38     }
39 }
40

```

- Making another Filter id:- And setting aid should be greater than or Equal 1:-

```

*FirstFilterName.java FirstFilterID.java x home.jsp
1 package com.udaysharma;
2 import java.io.IOException;
14
15 /**
16  * Servlet Filter implementation class FirstFilterID
17  */
18 @WebFilter("/MyServlet")
19 public class FirstFilterID extends HttpFilter implements Filter {
20
21     public FirstFilterID() {
22         super();
23     }
24     public void destroy() {
25         // TODO Auto-generated method stub
26     }
27     public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
28         System.out.println("hi This is Filter");
29         PrintWriter out=response.getWriter();
30         HttpServletRequest req=(HttpServletRequest) request;
31         int aid=Integer.parseInt(request.getParameter("aid"));
32         if(aid>=1)
33             chain.doFilter(request, response);
34         else
35             out.print("Invalid input ID is requiried ");
36     }
37
38     /**
39      * @see Filter#init(FilterConfig)
40      */
41     public void init(FilterConfig fConfig) throws ServletException {
42         // TODO Auto-generated method stub
43     }
44 }
45

```

- Now calling both page by using the home.jsp file:-

```

FirstFilterName.java FirstFilterID.java home.jsp x
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4 <html>
5 <head>
6 <meta charset="UTF-8">
7 <title>Insert title here</title>
8 </head>
9 <body bgcolor="cyan">
10 <form action="MyServlet">
11 Enter Your Id <input type="text" name="aid"><br>
12 Enter your Name <input type="text" name="aname"><br>
13 <input type="Submit"><br>
14 </form>
15 </body>
16 </html>

```

- Output:-

Two browser screenshots showing the initial form. The left screenshot shows the form with empty input fields. The right screenshot shows the form with '101' entered in the ID field and 'Uday Sharma' entered in the Name field.

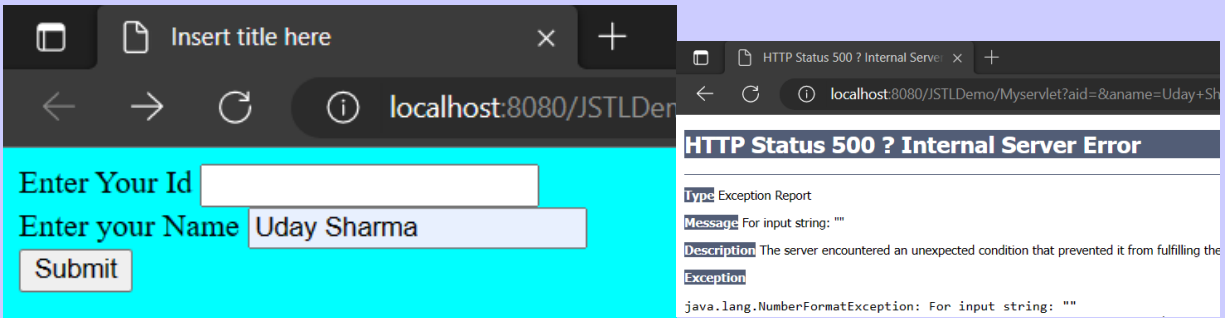
- After submitting:-

Browser screenshot showing the output "welcome Uday Sharma" after successful submission.

- If didn't enter the name we get the message in our page:-

Two browser screenshots showing the form with an error message. The left screenshot shows the form with '101' in the ID field and an empty Name field. The right screenshot shows the error message "Invalid input name is must".

- Similar with the ID:-



- At the Console we can see the Filter-1 and Filter -2 is running perfectly and we get the output on the console that our filter java file run perfectly:-

```
Console ×
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk-18.0.2.1\bin

hi THis is Filter
hi THis is Filter 2
hi THis is Filter
hi THis is Filter 2
hi THis is Filter
hi THis is Filter 2
hi THis is Filter
hi THis is Filter 2
hi THis is Filter
hi THis is Filter 2
hi THis is Filter
Apr 30, 2023 11:50:00 AM org.apache.catal
SEVERE: Servlet.service() for servlet [co
java.lang.NumberFormatException: For input
```

In upcoming Notes, covered a project based on core and Advanced Java
So, Stay Tuned.



➤ Go check out my **LinkedIn profile** for more notes and other resources content

 @Uday Sharma



mrudaysharma4600@gmail.com

<https://www.linkedin.com/in/uday-sharma-602b33267>