**National University of Computer & Emerging Sciences, Karachi**

**Computer Science Department**

**Fall 2022, Lab Manual - 08**

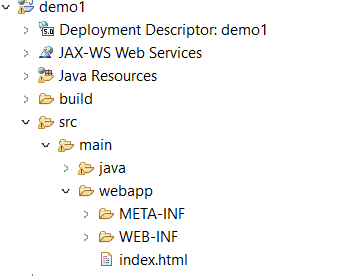
|  |  |
| --- | --- |
| **Course Code: SL3001** | **Course : Software construction and Development** |
| **Instructor :** | **Miss Nida Munawar** |

**Lab # 08**

**1st step**

Right click on project > new > html file >Index.html

It will automatically save inside src



<!DOCTYPE html>

<html>

<body>

hello <br>

<form action = *"add"* method = *"post"*>

enter 1st number <input type = *"text"* name = *"num1"*><br>

enter 2nd number <input type = *"text"* name = *"num2"*><br>

<input type = *"submit"*> <br>

</form>

</body>

</html>

**2nd step**

**Right click on project > new > class>addservelet.java**

**It will automatically save inside java resources**

Graphical user interface, text, application, chat or text message

Description automatically generated

package demo1;

import java.io.IOException;

import java.io.PrintWriter;

import jakarta.servlet.RequestDispatcher;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

public class addservelet extends HttpServlet {

public void service(HttpServletRequest req , HttpServletResponse res) throws IOException, ServletException {

int i = Integer.parseInt(req.getParameter("num1"));

int j = Integer.parseInt(req.getParameter("num2"));

int k = i+j;

System.out.println(k); // it will show output on console not on browser

}}

**3rd step**

Open web.xml and click on source tab

A picture containing graphical user interface

Description automatically generated

Add <servlet> and <servlet mapping> with the inner tags

In both <servlet> and <servlet mapping> the <servlet-name> must be same

<servlet class> is your package-name.servlet-class

The name of the action you specify inside of your index.html file is <url-pattern>, and it always begins with /.

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"https://jakarta.ee/xml/ns/jakartaee"* xmlns:web=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app\_5\_0.xsd"* id=*"WebApp\_ID"* version=*"5.0"*>

<servlet>

<servlet-name> abc </servlet-name>

<servlet-class>demo1.addservelet</servlet-class>

</servlet>

<servlet-mapping>

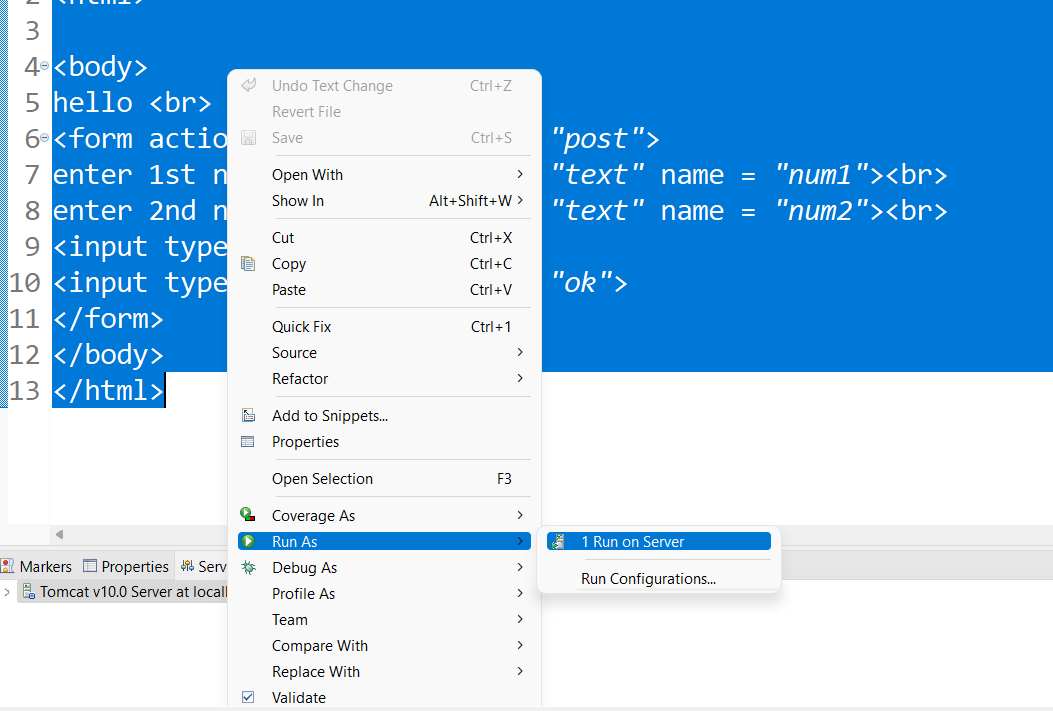
<servlet-name> abc</servlet-name>

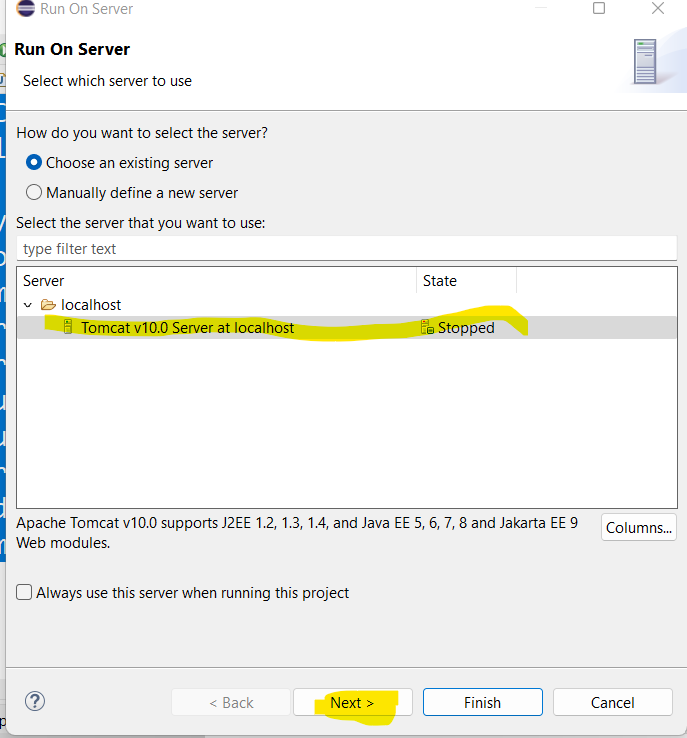
<url-pattern>/add</url-pattern>

</servlet-mapping>

</web-app>

Right click on your index.html file >





Graphical user interface, application

Description automatically generated

**Now it will redirect you to the browser**

**Printing on browser**

**Modify your servlet class**

package demo1;

import java.io.IOException;

import java.io.PrintWriter;

import jakarta.servlet.RequestDispatcher;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

public class addservelet extends HttpServlet {

public void service(HttpServletRequest req , HttpServletResponse res) throws IOException, ServletException {

int i = Integer.parseInt(req.getParameter("num1"));

int j = Integer.parseInt(req.getParameter("num2"));

int k = i+j;

PrintWriter out = res.getWriter();

out.println("add is " + k);

}

}

**Also use doGet and doPost method instead of service()**

# **RequestDispatcher in Servlet**

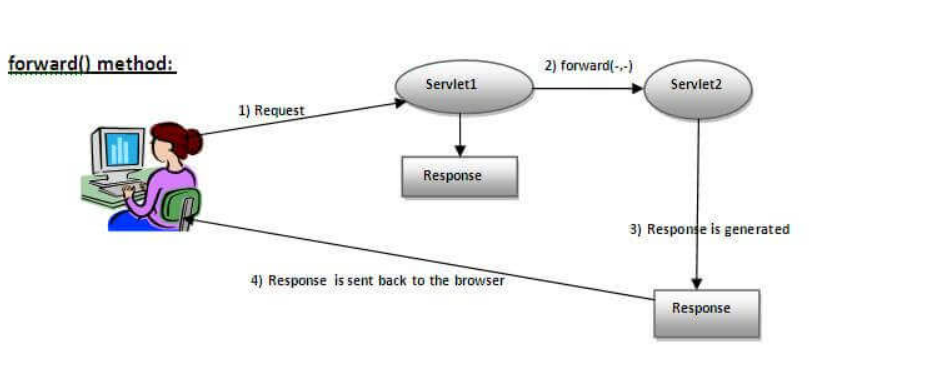
The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration.

Servlet to Servlet calling using request dispatcher

In this calling your client will not be notified that he/she redirects to another servlet RD works by using same req and res object

Two methods

1.



2.

Diagram

Description automatically generated

**Servlet to Servlet request**

**Add another java class to your project like you did it before**

package demo1;

import java.io.IOException;

import java.io.PrintWriter;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

public class sqServlet extends HttpServlet {

// TODO Auto-generated method stub

public void service(HttpServletRequest req , HttpServletResponse res) throws IOException{

PrintWriter out = res.getWriter();

out.println("hello sqrt");

}

}

**Modify your addservelet file**

**package** demo1;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.RequestDispatcher;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** addservelet **extends** HttpServlet {

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

**int** i = Integer.*parseInt*(req.getParameter("num1"));

**int** j = Integer.*parseInt*(req.getParameter("num2"));

**int** k = i+j;

RequestDispatcher rd = req.getRequestDispatcher("sq");

rd.forward(req, res);

}

}

**Modify your web.xml file**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"https://jakarta.ee/xml/ns/jakartaee"* xmlns:web=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app\_5\_0.xsd"* id=*"WebApp\_ID"* version=*"5.0"*>

<servlet>

<servlet-name> abc </servlet-name>

<servlet-class>demo1.addservelet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name> abc</servlet-name>

<url-pattern>/add</url-pattern>

</servlet-mapping>

<servlet>

<servlet-name> RD </servlet-name>

<servlet-class>demo1.sqServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name> RD</servlet-name>

<url-pattern>/sq</url-pattern>

</servlet-mapping>

</web-app>

**Passing a value from one servlet to another**

**Modify addservelet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.RequestDispatcher;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** addservelet **extends** HttpServlet

{

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

**int** i = Integer.*parseInt*(req.getParameter("num1"));

**int** j = Integer.*parseInt*(req.getParameter("num2"));

**int** k = i+j;

**req.setAttribute("k", k);**

RequestDispatcher rd = req.getRequestDispatcher("sq");

rd.forward(req, res);

}

}

**Modify sqServlet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** sqServlet **extends** HttpServlet {

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException{

int k = (int) req.getAttribute("k");// you have to cast the obj to int

PrintWriter out = res.getWriter();

out.println("value of k = " + k);

}

}

**session management**

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

# **URL Rewriting**

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.

**Send Redirect**

Servlet to Servlet calling using redirect

In this session management your client will be notified that he/she redirects to another servlet

**Modify addservelet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.RequestDispatcher;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** addservelet **extends** HttpServlet

{

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

**int** i = Integer.*parseInt*(req.getParameter("num1"));

**int** j = Integer.*parseInt*(req.getParameter("num2"));

**int** k = i+j;

**res.sendRedirect("sq");**

}

}

**Modify sqServlet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** sqServlet **extends** HttpServlet {

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException{

System.***out***.println("redirect");

}

}

**Passing a value from one servlet to another using URL rewriting**

**Modify sqServlet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** sqServlet **extends** HttpServlet {

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException{

**int** k = Integer.*parseInt*(req.getParameter("k"));

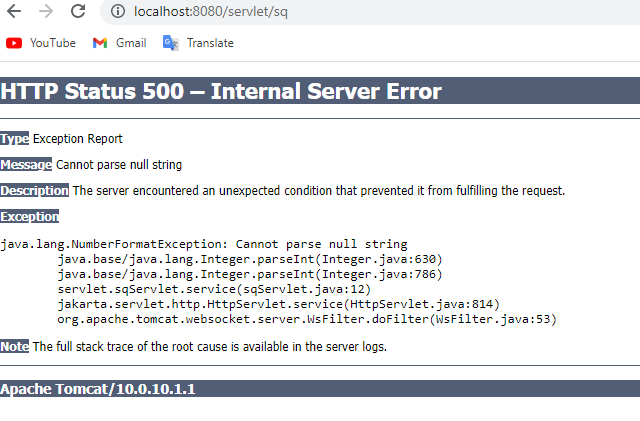
k=k+k;

PrintWriter out = res.getWriter();

out.println("the value of k " + k);

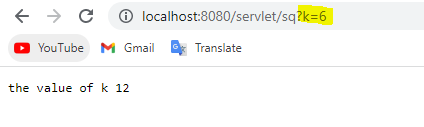
}

}



**You will get null string error because it expects parameter you can pass the parameter in url**

**?k=6**



**Second way to pass parameter**

**Modify addservelet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.RequestDispatcher;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** addservelet **extends** HttpServlet

{

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

**int** i = Integer.*parseInt*(req.getParameter("num1"));

**int** j = Integer.*parseInt*(req.getParameter("num2"));

**int** k = i+j;

**res.sendRedirect("sq?k=" + k);**

}

}

Disadvantage of URL Rewriting

1. It will work only with links.
2. It can send Only textual information.

# **2 HttpSession interface**

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.

**Modify addservelet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.RequestDispatcher;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**import** jakarta.servlet.http.HttpSession;

**public** **class** addservelet **extends** HttpServlet

{

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

**int** i = Integer.*parseInt*(req.getParameter("num1"));

**int** j = Integer.*parseInt*(req.getParameter("num2"));

**int** k = i+j;

**HttpSession session = req.getSession();**

**session.setAttribute("k", k);**

res.sendRedirect("sq");

}

}

**Modify sqServlet file**

**import java.io.IOException;**

**import java.io.PrintWriter;**

**import jakarta.servlet.http.HttpServlet;**

**import jakarta.servlet.http.HttpServletRequest;**

**import jakarta.servlet.http.HttpServletResponse;**

**import jakarta.servlet.http.HttpSession;**

**public class sqServlet extends HttpServlet {**

**public void service(HttpServletRequest req , HttpServletResponse res) throws IOException{**

**HttpSession session = req.getSession();**

**int k = (int) session.getAttribute("k");**

**k=k+k;**

**PrintWriter out = res.getWriter();**

**out.println("the value of k " + k);**

**}}**

# **Cookies in Servlet**

A **cookie** is a small piece of information that is persisted between the multiple client requests.

**Modify addservelet file**

**import** java.io.IOException;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.Cookie;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** addservelet **extends** HttpServlet

{

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

**int** i = Integer.*parseInt*(req.getParameter("num1"));

**int** j = Integer.*parseInt*(req.getParameter("num2"));

**int** k = i+j;

**Cookie coo = new Cookie("k", k + "");**

**res.addCookie(coo);**

res.sendRedirect("sq");

}

}

**Modify sqServlet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.http.Cookie;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** sqServlet **extends** HttpServlet {

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException{

**int** k = 0;

Cookie cookies [] = req.getCookies();

**for**(Cookie c : cookies ) {

**if**(c.getName().equals("k")) {

k = Integer.*parseInt*(c.getValue()) ;

}

}

k= k \*k;

PrintWriter out = res.getWriter();

out.println("the value of k " + k);

}

}

# **ServletContext Interface**

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

### Advantage of ServletContext

**Easy to maintain** if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

**Modify addservelet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.ServletContext;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** addservelet **extends** HttpServlet

{

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

PrintWriter out = res.getWriter();

out.print("hello " );

**ServletContext ctx = getServletContext();**

**String n = ctx.getInitParameter("name");**

out.println(n);

}

}

These values will be same for all servlets

**Modify web.xml file**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"https://jakarta.ee/xml/ns/jakartaee"* xmlns:web=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app\_5\_0.xsd http://xmlns.jcp.org/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"5.0"*>

<display-name>servlet</display-name>

<servlet>

<servlet-name> abc </servlet-name>

<servlet-class>servlet.addservelet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name> abc</servlet-name>

<url-pattern>/add</url-pattern>

</servlet-mapping>

<context-param>

<param-name> name </param-name>

<param-value> nida </param-value>

</context-param>

</web-app>

# **ServletConfig Interface**

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time.

**ServletConfig** and **ServletContext**, both are objects created at the time of servlet initialization and used to provide some initial parameters or configuration information to the servlet. But, the difference lies in the fact that information shared by ServletConfig is for a specific servlet, while information shared by ServletContext is available for all servlets in the web application.

**Modify addservelet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.ServletConfig;

**import** jakarta.servlet.ServletContext;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**public** **class** addservelet **extends** HttpServlet

{

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

PrintWriter out = res.getWriter();

out.print("hello " );

**ServletConfig cf = getServletConfig();**

**String n = cf.getInitParameter("name");**

out.println(n);

}

}

**Even we have sevlet context here but it will print the value that is specific to server**

**Modify web.xml file**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"https://jakarta.ee/xml/ns/jakartaee"* xmlns:web=*"http://xmlns.jcp.org/xml/ns/javaee"* xsi:schemaLocation=*"https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app\_5\_0.xsd http://xmlns.jcp.org/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"5.0"*>

<display-name>servlet</display-name>

<servlet>

<servlet-name> abc </servlet-name>

<servlet-class>servlet.addservelet</servlet-class>

**<init-param>**

**<param-name> name</param-name>**

**<param-value> Nida Munawar</param-value>**

**</init-param>**

</servlet>

<servlet-mapping>

<servlet-name> abc</servlet-name>

<url-pattern>/add</url-pattern>

</servlet-mapping>

<context-param>

<param-name> name </param-name>

<param-value> nida </param-value>

</context-param>

</web-app>

# **Servlets - Annotations**

So far, you have learnt how Servlet uses the deployment descriptor (web.xml file) for deploying your application into a web server. Servlet API 3.0 has introduced a new package called javax.servlet.annotation. It provides annotation types which can be used for annotating a servlet class. If you use annotation, then the deployment descriptor (web.xml) is not required. But you should use tomcat7 or any later version of tomcat.

Annotations can replace equivalent XML configuration in the web deployment descriptor file (web.xml) such as servlet declaration and servlet mapping. Servlet containers will process the annotated classes at deployment time.

|  |
| --- |
| **@WebServlet**  To declare a servlet. |
| **@WebInitParam**  To specify an initialization parameter. |  |

I commented out my all the servlet tags in web.xml

**Modify addservelet file**

**import** java.io.IOException;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.Cookie;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**@WebServlet("/add")**

**public** **class** addservelet **extends** HttpServlet

{

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

**int** i = Integer.*parseInt*(req.getParameter("num1"));

**int** j = Integer.*parseInt*(req.getParameter("num2"));

**int** k = i+j;

Cookie coo = **new** Cookie("k", k + "");

res.addCookie(coo);

res.sendRedirect("sq");

}

}

**Modify sqServlet file**

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.Cookie;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

**@WebServlet("/sq")**

**public** **class** sqServlet **extends** HttpServlet {

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException{

**int** k = 0;

Cookie cookies [] = req.getCookies();

**for**(Cookie c : cookies ) {

**if**(c.getName().equals("k")) {

k = Integer.*parseInt*(c.getValue()) ;

}

}

k= k \*k;

PrintWriter out = res.getWriter();

out.println("the value of k " + k);

}

}

**JSP**

***In servlet if we want to work with both designing and development This is difficult to do in servlet since we need to build separate print statements for each opening and closing tag, and the code would be messy.so instead of using html inside java, we use java inside html called JSP.***

Example without JSP if the programmer wants the add servlet output file color red

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** jakarta.servlet.ServletException;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.HttpServlet;

**import** jakarta.servlet.http.HttpServletRequest;

**import** jakarta.servlet.http.HttpServletResponse;

@WebServlet("/add")

**public** **class** addservelet **extends** HttpServlet

{

**public** **void** service(HttpServletRequest req , HttpServletResponse res) **throws** IOException, ServletException {

**int** i = Integer.*parseInt*(req.getParameter("num1"));

**int** j = Integer.*parseInt*(req.getParameter("num2"));

**int** k = i+j;

PrintWriter out = res.getWriter();

**out.print("<html><body bgcolor=red>");**

**out.println(k);**

**out.print("</html> </body>");**

}

}

# **JSP Tutorial**

**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

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.

Here we don’t need annotations and Http request and response jsp gives implicit objects

**Create a new file add.jsp**

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

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body **bgcolor=*"pink"***>

**<%**

**int i = Integer.parseInt(request.getParameter("num1"));**

**int j = Integer.parseInt(request.getParameter("num2"));**

**int k = i+j;**

**out.println("the value of k = " + k);**

**%>**

</body>

</html>

**Modify file index.html**

<!DOCTYPE html>

<html>

<body>

<form action = ***"add.jsp"*** method = *"post"*>

enter 1st number <input type = *"text"* name = *"num1"*><br>

enter 2nd number <input type = *"text"* name = *"num2"*><br>

<input type = *"submit"*> <br>

</body>

</html>

**Why servlet?**

**We can only write JSP we can’t run JSP, your tomcat is servlet container it only runs servlet not JSP so JSP is converted to Servlet automatically. your server understands servlet not JSP**

## **JSP Scripting elements**

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

* scriptlet tag
* expression tag
* declaration tag
* directive tag

### JSP scriptlet tag

In JSP, java code can be written inside the jsp page using the scriptlet tag.

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

The code written in scriptlet tag automatically pasted in the service() method inside servlet class

1. <%  java source code %>

# **JSP Declaration Tag**

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

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

So it doesn't get memory at each request.

#### **Syntax of JSP declaration tag**

The syntax of the declaration tag is as follows:

1. **<**%!  field or method declaration %**>**

### Difference between JSP Scriptlet tag and Declaration tag

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

# **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

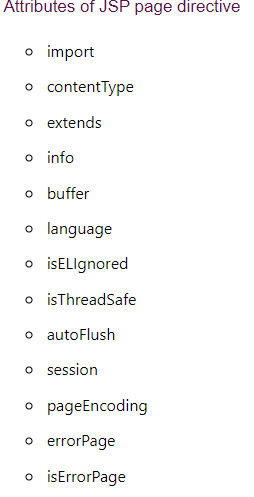
1. <%@ directive attribute="value" %>

### JSP page directive

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

### Syntax of JSP page directive

1. <%@ page attribute="value" %>



### 1)import

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

### Example of import attribute

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

### extends

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

# **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

1. <%@ include file="resourceName" %>

# **JSP Taglib directive**

We’ll learn it later in frame works

# **JSP expression tag**

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

### Syntax of JSP expression tag

1. **<**%=  statement %**>**

### Example of JSP expression tag

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

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

**Example with all tags**

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

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

**<!-- directive tag -->**

**<%@page import = *"java.util.\*"* %>**

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

**<!-- declarative tag -->**

**<%! int i = 4;%>>**

**<!-- Scriptlet tag -->**

**<% out.print(4+4);**

**Scanner s = new Scanner(System.in);**

**%>**

**<!-- Expression tag -->**

**the value of i is <%= i %>>**

</body>

</html>

# [**to find servlet(.java) files converted from jsp**](https://stackoverflow.com/questions/36358599/unable-to-find-servlet-java-files-converted-from-jsp)

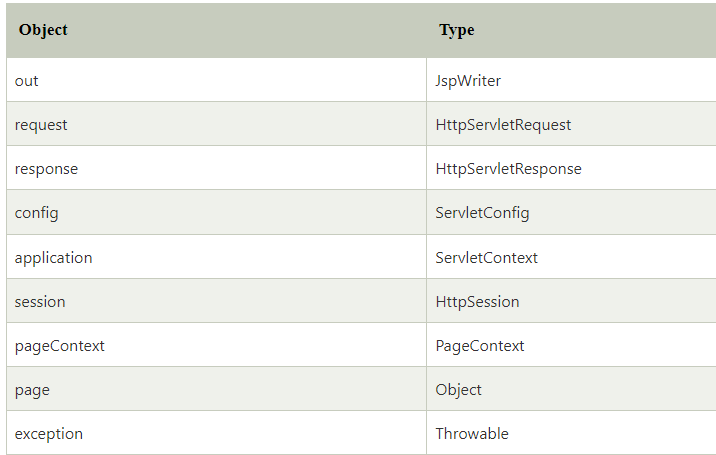
<%=getClass().getResource(getClass().getSimpleName() + ".class")%>

# **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:



# **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, one can handle exception by using try/catch block but for web applications it is convention to handle exceptions on a separate file:

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

**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.getMessage() %>

# **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>

## **The Methods in Form Processing**

GET method

The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the ? character as follows −

http://www.test.com/hello?key1=value1&key2=value2

The GET method is the default method to pass information from the browser to the web server and it produces a long string that appears in your browser's **Location:box**. It is recommended that the GET method is better not used. if you have password or other sensitive information to pass to the server.

The GET method has size limitation: **only 1024 characters can be in a request string**.

This information is passed using **QUERY\_STRING header** and will be accessible through QUERY\_STRING environment variable which can be handled using **getQueryString()** and **getParameter()** methods of request object.

POST method

A generally more reliable method of passing information to a backend program is the POST method.

This method packages the information in exactly the same way as the GET method, but instead of sending it as a text string after a ? in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which you can parse and use for your processing.

# **JSP - Page Redirecting**

public void response.sendRedirect(String location)

throws IOException

This method sends back the response to the browser along with the status code and new page location.

## **Hit Counter Application in JSP**

In this example, we will let the user know about the number of visits on a particular web page. We are creating an index.jsp page where the user will first land as a home page. Then we will implement hit counter using getAttribute() and setAttribute() methods. We are using setAttribute() to set a hit counter variable and to reset the same and getAttribute() to read the current value of the hit counter, every time a user accesses that particular web page.

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

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<%

Integer hitsCount = (Integer) application.getAttribute("hitCounter");

**if** (hitsCount == **null** || hitsCount == 0) {

/\* First visit \*/

out.println("Welcome to my website!");

hitsCount = 1;

} **else** {

/\* return visit \*/

out.println("Welcome back to my website!");

hitsCount += 1;

}

application.setAttribute("hitCounter", hitsCount);

%>

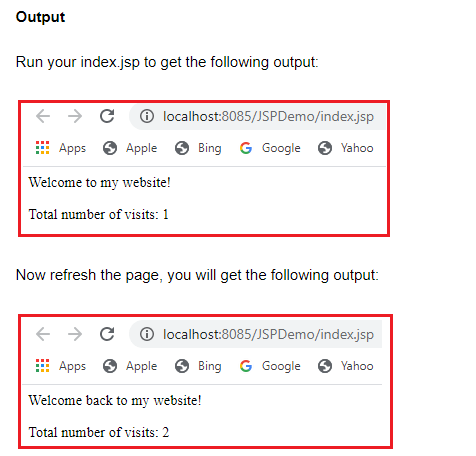
<p>

Total number of visits:

<%=hitsCount%></p>

</body>

</html>



You can try accessing the page using different browsers and you will find that the hit counter will keep increasing with every hit application object maintain the counter value throughout the browser

**What if we want after switching the browser the counter should start with 0 again for this we use session object**

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

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<%

Integer hitsCount = (Integer) session.getAttribute("hitCounter");

**if** (hitsCount == **null** || hitsCount == 0) {

/\* First visit \*/

out.println("Welcome to my website!");

hitsCount = 1;

} **else** {

/\* return visit \*/

hitsCount += 1;

}

out.println("Welcome back to my website!");

session.setAttribute("hitCounter", hitsCount);

%>

<p>

Total number of visits:

<%=hitsCount%></p>

</body>

</html>

# **JSP - Auto Refresh**

Auto Refresh in JSP. Consider a webpage which is displaying live game score or stock market status or currency exchange ration. For all such type of pages, you would need to refresh your Webpage regularly using refresh or reload button with your browser.

JSP makes this job easy by providing you a mechanism where you can make a webpage in such a way that it would refresh automatically after a given interval.

The simplest way of refreshing a Webpage is by using the **setIntHeader()** method of the response object. Following is the signature of this method −

public void setIntHeader(String header, int headerValue)

This method sends back the header "Refresh" to the browser along with an integer value which indicates time interval in seconds.

## **Auto Page Refresh Example**

In the following example, we will use the **setIntHeader()** method to set **Refresh** header. This will help simulate a digital clock −

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

pageEncoding=*"ISO-8859-1"*%>

**<%@ page import = *"java.io.\*,java.util.\*"* %>**

<!-- java.util.\* for calender

java.io.\* for -->

<!DOCTYPE html>

<html>

<head>

<meta charset=*"ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

**<%**

**// Set refresh, autoload time as 5 seconds**

**response.setIntHeader("Refresh", 5);**

**// Get current time**

**Calendar calendar = new GregorianCalendar();**

**String am\_pm;**

**int hour = calendar.get(Calendar.HOUR);**

**int minute = calendar.get(Calendar.MINUTE);**

**int second = calendar.get(Calendar.SECOND);**

**if(calendar.get(Calendar.AM\_PM) == 0)**

**am\_pm = "AM";**

**else**

**am\_pm = "PM";**

**String CT = hour+":"+ minute +":"+ second +" "+ am\_pm;**

**out.println("Crrent Time: " + CT + "\n");**

**%>**

</body>

</html>