Session- 6

SERVLETS

What is servlet?

- ✓ It is a server side component which will be used to provide service to the multiple clients. (or)
- ✓ It is an implementation class of Servlet interface which is available in javax.servlet package. (or)
- ✓ It is a subclass of GenericServlet which is available in javax.servlet package. (or)
- ✓ It is a subclass of HttpServlet which is available in javax.servlet.http package.

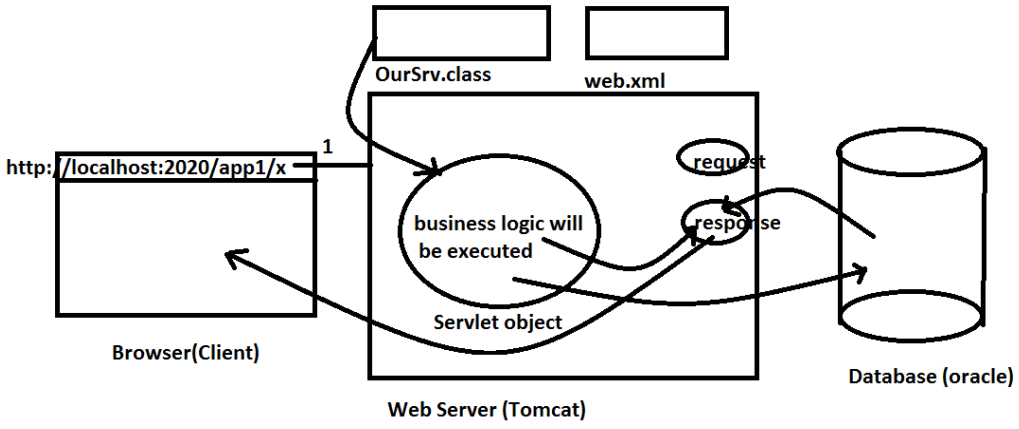
-> Servlet-api was implemented by set of server vendors and given in the form of jar file called 'servlet-api.jar'.

-> Servlet API contains 2 packages. They are

i) javax.servlet

ii) javax.servlet.http

Architecture of Servlets:



-> Before servlet technology, we have CGI (Common Gateway Interface) technology can be used to develop web applications.

-> Under CGI, we can use native languages(c/c++), perl script.

-> But all the above languages follow process-based tasking. Due to this, processor load will be increased, and response time will be increased.

-> To overcome the above problems, servlet technology was implemented by set of webservers like

a) tomcat given by apache

b) web sphere given by IBM

c) WebLogic given by oracle

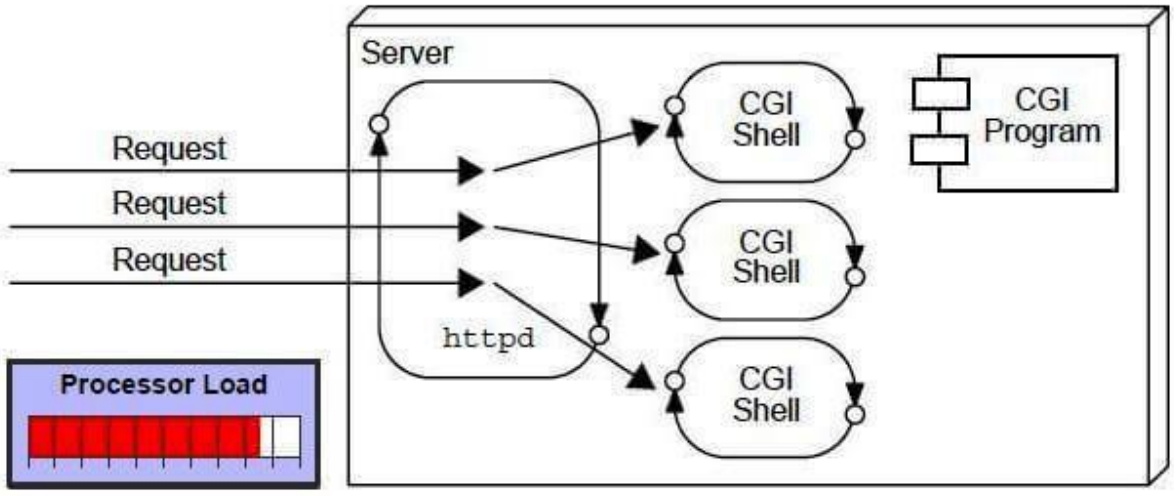
d) glassfish given by sun microsystems

e) jboss given by redhat etc...

-> Servlet Technology was implemented based on threads concept. So it is light weight.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

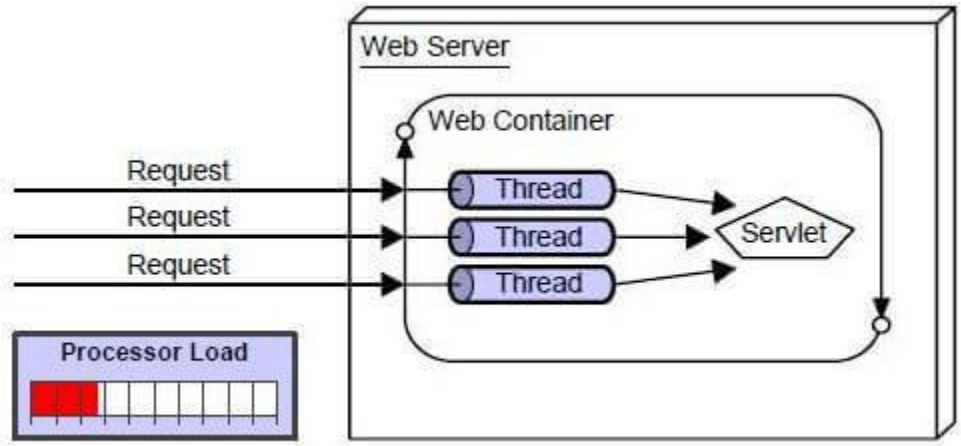


Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl

Advantages of Servlet



There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. **Better performance:** because it creates a thread for each request, not process.
2. **Portability:** because it uses Java language.
3. **Robust:** JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. **Secure:** because it uses java language.

Java Servlets: Steps to Create Servlet

-> To create any web-application under tomcat server, we have to follow the following directory structure:

Note:- This directory structure we have to create under 'webapps' folder which is available in

C:\program files\Apache software foundation\tomcat8.0

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Add mappings to the web.xml file
5. Start the server and deploy the project
6. Access the servlet

Now, based on the above steps, let's write a program and understand how a servlet works.

To run a servlet program, we should have Apache Tomcat Server installed and configured. *Eclipse for Java EE provides in-built Apache Tomcat.* Once the server is configured, you can start with your program. One important point to note – for any servlet program, you need 3 files – *index.html file, Java class file, and web.xml file*. The very first step is to create a Dynamic Web Project and then proceed further.

Now, let's see how to add 2 numbers using servlets and display the output in the browser.

First, I will write index.html

```
<!DOCTYPE HTML>

<html>

<body>

<form action = "add">

Enter 1st number: <input type="text" name ="num1">

 Enter 2nd number: <input type="text" name="num2">

</form>

</body>

</html>
```

Above program creates a form to enter the numbers for the addition operation. Without the Java class file, you can't perform addition on 2 numbers. So let's now create a class file.

```
import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class Add extends HttpServlet{

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

{

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

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

int k= i+j;

PrintWriter out = res.getWriter();

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

}

}
```

After writing the Java class file, the last step is to add mappings to the web.xml file. Let's see how to do that.

The *web.xml* file will be present in the WEB-INF folder of your web content. If it is not present, then you can click on Deployment Descriptor and click on *Generate Deployment Descriptor Stub.* Once you get your web.xml file ready, you need to add the mappings to it. Let's see how mapping is done using the below example:

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

<web-app>

<display-name>Basic</display-name>

<servlet>

<servlet-name>Addition</servlet-name>

<servlet-class>Add</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Addition</servlet-name>

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

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>
```

To create any Servlet you must implement the Servlet interface directly or indirectly (indirectly implementation means extending those classes that implements Servlet interface, These classes are GenericServlet and HttpServlet).

If you are creating protocol dependent servlet such as http servlet then you should extend HttpServlet class else for protocol independent Servlet you extend GenericServlet class.

In short you have 3 ways to create a servlet:
1) By extending HttpServlet class
2) By extending GenericServlet class
3) By implementing Servlet interface
Note: However you should always prefer the first way of creating servlet i.e. by extending HttpServlet class.

Servlet Interface methods

Here is the list of methods available in Servlet interface.
1) **void destroy()**: This method is called by Servlet container at the end of servlet life cycle. Unlike service() method that gets called multiple times during life cycle, this method is called only once by Servlet container during the complete life cycle. Once destroy() method is called the servlet container does not call the service() method for that servlet.

2) **void init(ServletConfig config)**: When Servlet container starts up (that happens when the web server starts up) it loads all the servlets and instantiates them. After this init() method gets called for each instantiated servlet, this method initializes the servlet.

3) **void service(ServletRequest req, ServletResponse res)**: This is the only method that is called multiple times during servlet life cycle. This methods serves the client request, it is called every time the server receives a request.

4) **ServletConfig getServletConfig()**: Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.

5) **java.lang.String getServletInfo()**: Returns information about the servlet, such as author, version, and copyright.

Example:

In this example we have created a servlet class by extending Servlet interface.

index.html

```
<ahref="welcome">Click here to call the servlet</a>
```

DemoServlet.java

```
import java.io.*;
import javax.servlet.*;
publicclassDemoServletimplementsServlet{
ServletConfig config=null;
publicvoid init(ServletConfig config){
this.config=config;
System.out.println("Initialization complete");
}
publicvoid service(ServletRequest req,ServletResponse res)
```

```
throwsIOException,ServletException{
       res.setContentType("text/html");
PrintWriter pwriter=res.getWriter();
       pwriter.print("<html>");
       pwriter.print("<body>");
       pwriter.print("<h1>Servlet Example Program</h1>");
       pwriter.print("</body>");
       pwriter.print("</html>");
}
publicvoid destroy(){
System.out.println("servlet life cycle finished");
}
publicServletConfig getServletConfig(){
return config;
}
publicString getServletInfo(){
return"A Demo program written by us";
}
}
```

web.xml

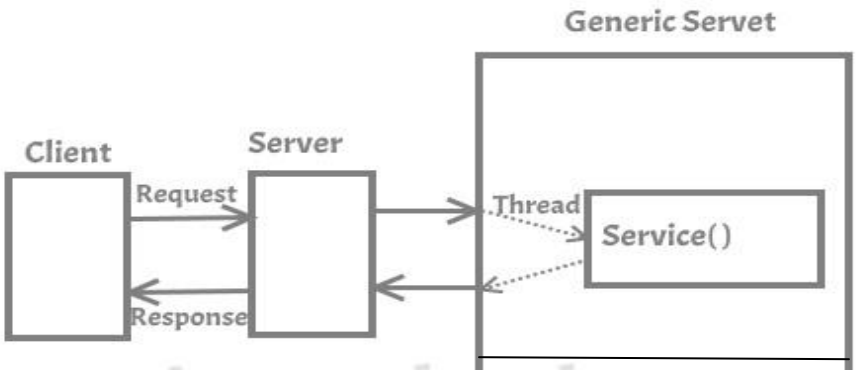
```
<web-app>
<servlet>
<servlet-name>ServletDemo</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>ServletDemo</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Generic Servlet:

A generic servlet is a protocol independent Servlet that should always override the service() method to handle the client request. The service() method accepts two arguments ServletRequest object and ServletResponse object. The request object tells the servlet about the request made by client while the response object is used to return a response back to the client.

How Generic Servlet works?



Hierarchy of Generic Servlet

```
java.lang.Object
        |_extended byjavax.servlet.GenericServlet
```

GenericServlet is an abstract class and it has only one abstract method, which is service(). That's why when we create Generic Servlet by extending GenericServlet class, we must override service() method.

Pros of using Generic Servlet:
1. Generic Servlet is easier to write
2. Has simple lifecycle methods
3. To write Generic Servlet you just need to extend javax.servlet.GenericServlet and override the service() method (check the example below).

Cons of using Generic Servlet:
Working with Generic Servlet is not that easy because we don't have convenience methods such as doGet(), doPost(), doHead() etc in Generic Servlet that we can use in Http Servlet.
In Http Servlet we need to override particular convenience method for particular request, for example if you need to get information then override doGet(), if you want to send information to server override doPost(). However in Generic Servlet we only override service() method for each type of request which is cumbersome.

we would always recommend you to use HttpServlet instead of the GenericServlet. HttpServlet is easier to work with, and has more methods to work with than GenericServlet.

index.html

We are creating an html file that would call the servlet once we click on the link on web page. This path of the file should look like this: WebContent/index.html.

```
<!DOCTYPE html>
<html>
<head>
<metacharset="UTF-8">
<title>Generic Servlet Demo</title>
</head>
<body>
<ahref="welcome">Click to call Servlet</a>
</body>
</html>
```

ExampleGeneric.java

Now, we are creating a Generic Servlet by extending GenericServlet class. When creating a GenericServlet you should always override service() method

```
import java.io.*;
import javax.servlet.*;

publicclassExampleGenericextendsGenericServlet{
publicvoid service(ServletRequest req,ServletResponse res)
throwsIOException,ServletException{
   res.setContentType("text/html");
PrintWriter pwriter=res.getWriter();
   pwriter.print("<html>");
   pwriter.print("<body>");
   pwriter.print("<h2>Generic Servlet Example</h2>");
   pwriter.print("<p>Hello Readers!</p>");
   pwriter.print("</body>");
   pwriter.print("</html>");
 }
}
```

web.xml

This file can be found at this path WebContent/WEB-INF/web.xml. In this file we will map the Servlet with the specific URL. Since we are calling welcome page upon clicking the link on index.html page so we are mapping the welcome page to the Servlet class we created above.

```
<web-app>
<display-name>Servlet</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>default.html</welcome-file>
<welcome-file>default.htm</welcome-file>
<welcome-file>default.jsp</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>MyGenericServlet</servlet-name>
<servlet-class>ExampleGeneric</servlet-class>
</servlet>

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

</web-app>
```

Methods of GenericServlet class:

Here is the list of all the methods of GenericServlet class.
1. **public void init()**: it is a convenient method. This method can be overridden so that there's no need to call super.init(config).

2. **public void init(ServletConfig config)**: Called by the servlet container to indicate that the servlet is being placed into service, this method is used for initializing the servlet.

3. **public String getInitParameter(String name)**: Returns a String containing the value of the given initialization parameter, or null if the parameter does not exist.

4. **public Enumeration getInitParameterNames()**: Returns the names of all the parameters defined in the web.xml file or null if web.xml does't have any parameter.

5. **public abstract void service(ServletRequest request, ServletResponse response)**: Called by the Servlet container to allow servlet to respond to the requests made by client.

6. **public void destroy()**: It is called by servlet container once at the end of servlet life cycle to indicate that servlet is being destroyed.

7. **public ServletConfig getServletConfig()**: Return the ServletConfig object that initialized this servlet

8. **public String getServletInfo()**: Returns information about servlet.

9. **public ServletContext getServletContext()**: Return ServletContext object, passed to this servlet by the init method

10. **public String getServletName()**: Returns the name of the servlet instance.

11. **public void log(String msg)**: Writes the given message in the servlet log file.

12. **public void log(String msg,Throwable t)**: Writes the explanatory message in the servlet log file including a String that describes the error or exception.

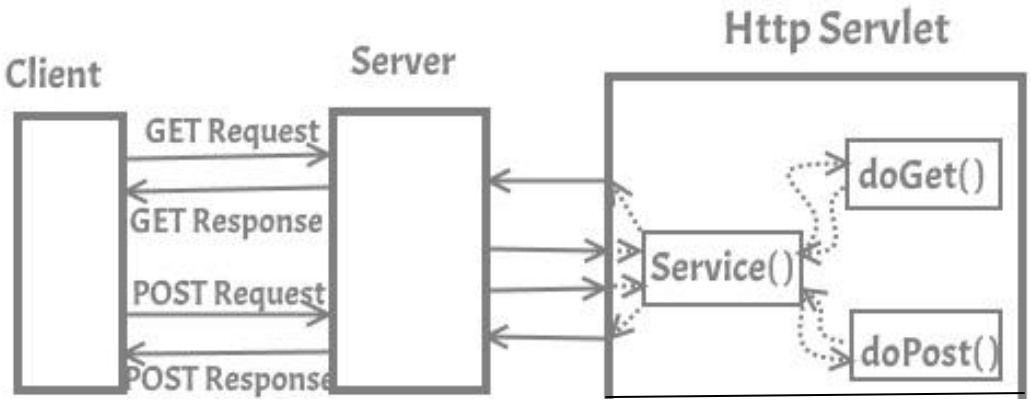
HttpServlets:

Unlike Generic Servlet, the HTTP Servlet doesn’t override the service() method. Instead it overrides the doGet() method or doPost() method or both. The doGet() method is used for getting the information from server while the doPost() method is used for sending information to the server.

In Http Servlet there is no need to override the service() method because this method dispatches the Http Requests to the correct method handler, for example if it receives HTTP GET Request it dispatches the request to the doGet() method.

How Http Servlet works?

As you can see in the diagram below that client (user's browser) make requests. These requests can be of any type, for example – Get Request, Post Request, Head Request etc. Server dispatches these requests to the servlet's service() method, this method dispatches these requests to the correct handler for example if it receives Get requests it dispatches it to the doGet() method.



Hierarchy of Http Servlet

```
java.lang.Object
        |_extended byjavax.servlet.GenericServlet
        |_extended byjavax.servlet.http.HttpServlet
```

I have already discussed in the Generic Servlet article that you should always use HttpServlet instead of the GenericServlet. HttpServlet is easier to work with, and has more methods to work with than GenericServlet.

Http Servlet example

index.html

We are creating an html file that would call the servlet once we click on the link on web page.

```
index<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>HttpServletDemo</title>
</head>
<body>
<a href="welcome">Click to call Servlet</a>
</body>
</html>
```

ExampleHttpServlet.java

Now, we are creating a Http Servlet by extending HttpServlet class

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
// Creating Http Servlet by Extending HttpServlet class
publicclassExampleHttpServletextendsHttpServlet
{
privateString mymsg;
publicvoid init() throwsServletException
    {
       mymsg = "Http Servlet Demo";
    }
publicvoid doGet(HttpServletRequest request,
HttpServletResponse response) throwsServletException,
IOException
    {
// Setting up the content type of web page
        response.setContentType("text/html");
// Writing the message on the web page
PrintWriterout = response.getWriter();
out.println("<h1>" + mymsg + "</h1>");
out.println("<p>" + "Hello Friends!" + "</p>");
    }
publicvoid destroy()
    {
// Leaving empty. Use this if you want to perform
//something at the end of Servlet life cycle.
    }
}
```

web.xml
In this file we will map the Servlet with the specific URL. Since we are calling welcome page upon clicking the link on index.html page so we are mapping the welcome page to the Servlet class we created above.

```
<web-app>
<display-name>BeginnersBookServlet</display-name>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
<welcome-file>index.htm</welcome-file>
<welcome-file>index.jsp</welcome-file>
<welcome-file>default.html</welcome-file>
<welcome-file>default.htm</welcome-file>
<welcome-file>default.jsp</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>MyHttpServlet</servlet-name>
<servlet-class>ExampleHttpServlet</servlet-class>
</servlet>

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

</web-app>
```

Methods of HttpServlet class

1. **protected void doGet(HttpServletRequest req, HttpServletResponse resp)**: This method is called by servlet service method to handle the HTTP GET request from client. When overriding this method, read the request data, write the response headers, get the response's writer or output stream object, and finally, write the response data.

2. **protected long getLastModified(HttpServletRequest req)**: Returns a long integer specifying the time the HttpServletRequest object was last modified, in milliseconds since midnight, January 1, 1970 GMT, or -1 if the time is not known

3. **protected void doHead(HttpServletRequest req, HttpServletResponse resp)**: This method is called by servlet service method to handle the HTTP HEAD request from client. The client sends a HEAD request when it wants to see only the headers of a response, such as Content-Type or Content-Length

4. **protected void doPost(HttpServletRequest req, HttpServletResponse resp)**: This method is called by servlet service method to handle the POST request from client. The HTTP POST method allows the client to send data of unlimited length to the Web server a single time and is useful when posting information to the server. Unlike, doGet where we get information from the sever this method is used when we are transferring information from client to the server.

5. **protected void doPut(HttpServletRequest req, HttpServletResponse resp)**: This method is called by servlet service method to handle the PUT request from client. This method is similar to doPost method but unlike doPost method where we send information to the server, this method sends file to the server, this is similar to the FTP operation from client to server.

6. **protected void doDelete(HttpServletRequest req, HttpServletResponse resp)**: Called by servlet service() method to handle the DELETE request from client that allows a client to delete a document, webpage or information from the server.

7. **protected void doOptions(HttpServletRequest req, HttpServletResponse resp)**: Called by the service method to allow a servlet to handle a OPTIONS request. The OPTIONS request determines which HTTP methods the server supports and returns an appropriate header.

8. **protected void doTrace(HttpServletRequest req, HttpServletResponse resp)**: This method is called by service() method for handling TRACE request. Used for debugging purposes.

9. **protected void service(HttpServletRequest req, HttpServletResponse resp)**: There is no need to override this method, this method receives the HTTP request from client and forwards them to the corresponding doXXX methods such as doGet(), doPost(), doHEAD() etc.

10. **public void service(ServletRequest req, ServletResponse res)**: Forwards client request to the protected service method. There’s no need to override this method as well.