UNIT-III

Java Servlets & XML

Servlets:

1) Introduction:

- Java Servlets are the programs that run on a Web Application Server.
- It acts as a middle layer between a requests from web browser /client and database/application on HTTP server.

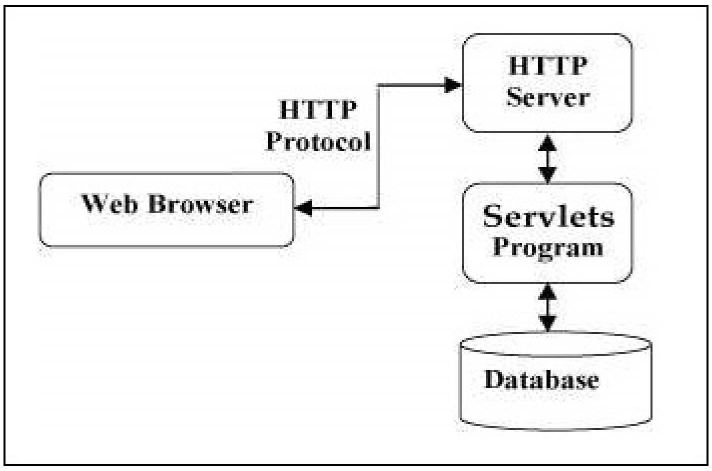


Fig 1: Architecture

2) Advantages of Servlets:

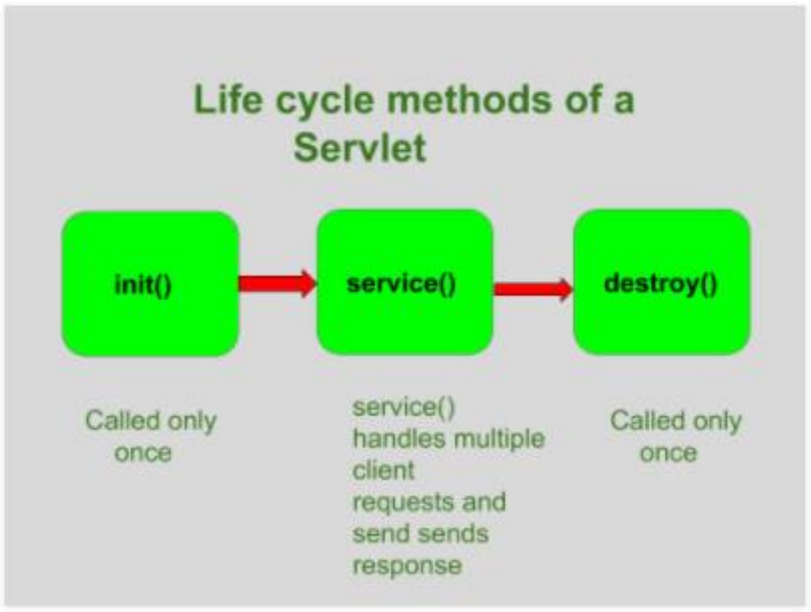
- It offers better performance than CGI (Common Gateway Interface).
- Servlets executes within address space of server.
- Servlets are platform independent just like java.
- More secure.
- Communication with other applications on other platforms easily with RMI (Remote Method Invocation).

3) Uses of Servlets:

- Read explicit data sent by browser easily. Explicit data means data i.e. sent via HTML form)
- Read implicit HTTP request data. HTTP request means cookies etc.
- Process the data at ease.
- Send explicit as well as implicit data back to the user.

4) Servlets Life Cycle:

There are three life cycle methods of a Servlet :


Life cycle methods of a Servlet
init()
service()
destroy()
Called only once
service() handles multiple client requests and send sends response
Called only once

- ***init() :*** It is only called once when the Servlet is created.

```
public void init() throws ServeltException
{

  //Initialization code

}
```

- ***service():*** It is the main method that performs the actual task. Whenever the request is made from the browser/client, the servlet container calls the service() method to handle the request coming from client. It checks the HTTP request type (Get, Post) such as doGet and doPost etc.

```
public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException
{

 // code

}

public void doGet(HTTPServletRequest request, HTTPServletResponse
response) throws ServletException, IOException
{

// code
}

public void doPost(HTTPServletRequest request, HTTPServletResponse
response) throws ServletException, IOException
{

// code

}
```

- ***destroy():*** It is called only once at the end of the Servlet life cycle. It closes the database connection, background threads, cookies etc.

```
public void destroy()
{

// cleanup code

}
```

5) First Servlet “HelloWorld” Program

HelloWorld.java

```
//Import required java libraries

/*for older versions of Tomcat*/

//import javax.servlet.*;

//import javax.servlet.http.*;


/*jakarta is required instead of javax for

Tomcat 10.0 and above versions.*/

import jakarta.servlet.*;

import jakarta.servlet.http.*;

import java.io.*;


// Extend HttpServlet class

public class HelloWorld extends HttpServlet {
  private String message;

  public void init() throws ServletException {
    // Do required initialization
    message = "Hello World";
  }

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    // Actual logic goes here.
    PrintWriter out = response.getWriter();
    out.println("<h1>" + message + "</h1>");
  }

  public void destroy() {
    // do nothing.
  }
}
```

web.xml: configuration file

```
<servlet>
   <servlet-name>HelloWorld</servlet-name>
   <servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloWorld</servlet-name>
   <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

How to Create and Execute First Servlet Program as HelloWorld Program:

1) **Download and Install Apache Tomcat:** https://tomcat.apache.org/download-10.cgi Preferably zip file of apache tomcat. Download it and extract the zip folder in your destined location (C: drive is recommended).

2) **Download and Install Java (latest version of JDK):** https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe

3) **Set the path for JDK:** Go to Home Search→ Type Variables→Edit the System Environment Variables→Advanced→Environment Variables→System Variables→Double Click on Path→New→Copy and Paste the path of our JDK eg. **C:\Program Files\Java\jdk-17.0.2\bin**

4) **Set the path for Apache Tomcat:** Go to Home Search→ Type Variables→Edit the System Environment Variables→Advanced→Environment Variables→System Variables→Double Click on Path→New→Copy and Paste the path of our JDK eg. **C:\apache-tomcat-10.0.18\bin**

5) **Create CATALINA_HOME:** Go to Home Search→ Type Variables→Edit the System Environment Variables→Advanced→Environment Variables→System Variables→New→Set variable name as CATALINA_HOME and value as **C:\apache-tomcat-10.0.18**

6) **Create CLASSPATH:** Go to Home Search→ Type Variables→Edit the System Environment Variables→Advanced→Environment Variables→System Variables→New→Set variable name as CLASSPATH and value as **C:\apache-tomcat-10.0.18\lib\jsp-api.jar; C:\apache-tomcat-10.0.18\lib\servlet-api.jar**

7) Go to **C:\apache-tomcat-10.0.18\conf** and open **tomcat-users.xml** file with notepad and comment rest of the role and user tags and add below code to it and save it.

```
<role rolename="manager-gui"/>

<user username="tomcat" password="s3cret" roles="manager-gui"/>
```

8) Go to **CMD** and type **startup**, it will start our **TomCat Server**. Minimize the server.

9) Now open Browser, type **http://localhost:8080**, click on **Manager App.** Enter username and password set it **tomcat-users.xml** file as Username: tomcar and Password: s3cret. It will open http://localhost:8080/manager/html. Consider tomcat server is started successfully.

10) Create folder structure as per given below:

I. Create a folder named **DemoApp** for example under **C:\apache-tomcat-10.0.18\webapps.**

II. Create a folder named **WEB-INF** for example under **C:\apache-tomcat-10.0.18\webapps\DemoApp.**

III. Create a folder named **classes** for example under **C:\apache-tomcat-10.0.18\webapps\DemoApp\WEB-INF**

IV. Copy the **web.xml** file from **C:\apache-tomcat-10.0.18\webapps\ROOT\WEB-INF\** to **C:\apache-tomcat-10.0.18\webapps\DemoApp\WEB-INF**

```
webapps
   |
   DemoApp
      | WEB-INF
         | classes
            |. Java file
            |.class file (it is created after compiling class file)
         | web.xml

     .html
     .css
     .js
     etc
```

11) Create a java file as our servlet and save it under **C:\apache-tomcat-10.0.18\webapps\DemoApp\WEB-INF\classes**

12) **For Compilation,** Open **CMD** and type as below:

```
javac -cp "C:\apache-tomcat-10.0.18\lib\servlet-api.jar" C:\apache-tomcat-10.0.18\webapps\DemoApp\WEB-INF\classes\HelloWorld.java
```

This will create **HelloWorld.class** file will be created under the same classes folder.

13) Now, configure your servlet in **web.xml** file stored under to **C:\apache-tomcat-10.0.18\webapps\DemoApp\WEB-INF** using a following tags.

```
<servlet>
   <servlet-name>HelloWorld</servlet-name>
   <servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloWorld</servlet-name>
   <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Copy and Paste the above code in the **web.xml** file by replacing below code.

```
<display-name>Welcome to Tomcat</display-name>
  <description>
     Welcome to Tomcat
  </description>
```

13) Run the servlet application:

1. ***Open the browser and type below URL in it:***

```
http://localhost:portnumber/urlname in web.xml
```

eg. http://localhost:8080/DemoApp/HelloWorld

6) Parameter Data

You must have come across many situations when you need to pass some information from your browser to web server and ultimately to your backend program. The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method.

6.1 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 **?** (question mark) symbol as follows −

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

- The GET method is the default method to pass information from browser to web server and it produces a long string that appears in your browser's Location:box. Never use the GET method if you have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be used in a request string.
- This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

6.2 POST Method

- A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET method, but instead of sending it as a text string after a ? (question mark) 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. Servlet handles this type of requests using **doPost()** method.

There are many differences between the Get and Post request. Let's see these differences:

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**

5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.

6.3 Reading Form Data using Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation −

- **getParameter()** − You call request.getParameter() method to get the value of a form parameter.
- **getParameterValues()** − Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- **getParameterNames()** − Call this method if you want a complete list of all parameters in the current request.

EX:1 GET Method Example Using URL

Here is a simple URL which will pass two values to HelloForm program using GET method.

http://localhost:8080/HelloForm?first_name = VISHAKHA&last_name = METRE

Given below is the **HelloForm.java** servlet program to handle input given by web browser. We are going to use **getParameter()** method which makes it very easy to access passed information –

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

   public void doGet(HttpServletRequest request, 
HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Using GET Method to Read Form Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " + 
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>First Name</b>: "
                  + request.getParameter("first_name") + "\n" +
                  "  <li><b>Last Name</b>: "
                  + request.getParameter("last_name") + "\n" +
               "</ul>\n" +
            "</body>" +
         "</html>"
      );
   }
}
```

Assuming your environment is set up properly, compile HelloForm.java as follows −

```
$ javac HelloForm.java
```

If everything goes fine, above compilation would produce **HelloForm.class** file. Next you would have to copy this class file in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
   <servlet-name>HelloForm</servlet-name>
   <servlet-class>HelloForm</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloForm</servlet-name>
   <url-pattern>/HelloForm</url-pattern>
</servlet-mapping>
```

Now type *http://localhost:8080/HelloForm?first_name=VISHAKHA&last_name=METRE* in your browser's Location:box and make sure you already started tomcat server, before firing above command in the browser. This would generate following result −

Using GET Method to Read URL

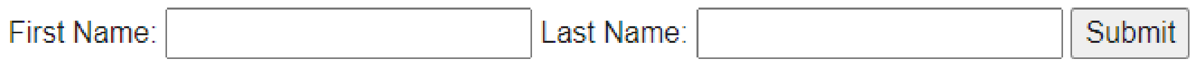
- **First Name**: VISHAKHA
- **Last Name**: METRE

EX:2 GET Method Example Using Form

Here is a simple example which passes two values using HTML FORM and submit button. We are going to use same Servlet HelloForm to handle this input.

```
<html>
   <body>
      <form action = "HelloForm" method = "GET">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

Keep this HTML in a file HelloForm.html and put it in <Tomcat-installationdirectory>/webapps/ROOT directory. When you would access *http://localhost:8080/HelloForm.html*, here is the actual output of the above form.



Try to enter First Name and Last Name and then click submit button to see the result on your local machine where tomcat is running. Based on the input provided, it will generate similar result as mentioned in the above example.

Using GET Method to Read Form Data

- **First Name**: VISHAKHA
- **Last Name**: METRE

EX:3 POST Method Example Using Form

Let us do little modification in the above servlet, so that it can handle GET as well as POST methods. Below is HelloForm.java servlet program to handle input given by web browser using GET or POST methods.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloForm extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {
```

```
      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Using GET & Post Method to Read Form Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
         "transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>First Name</b>: "
                  + request.getParameter("first_name") + "\n" +
                  "  <li><b>Last Name</b>: "
                  + request.getParameter("last_name") + "\n" +
               "</ul>\n" +
            "</body>"+
         "</html>"
      );
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now compile and deploy the above Servlet and test it using HelloForm.html with the POST method as follows −

```
<html>
   <body>
      <form action = "HelloForm" method = "POST">
         First Name: <input type = "text" name = "first_name">
         <br />
         Last Name: <input type = "text" name = "last_name" />
         <input type = "submit" value = "Submit" />
      </form>
   </body>
</html>
```

Here is the actual output of the above form, Try to enter First and Last Name and then click submit button to see the result on your local machine where tomcat is running.

Based on the input provided, it would generate similar result as mentioned in the above examples.

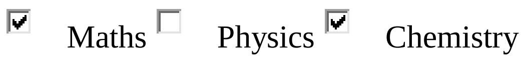
EX:4 Passing Checkbox Data to Servlet Program

Checkboxes are used when more than one option is required to be selected.

Here is example HTML code, CheckBox.htm, for a form with two checkboxes

```
<html>
   <body>
      <form action = "CheckBox" method = "POST" target = 
"_blank">
         <input type = "checkbox" name = "maths" checked = 
"checked" /> Maths
         <input type = "checkbox" name = "physics"  /> Physics
         <input type = "checkbox" name = "chemistry" checked = 
"checked" />
                                          Chemistry
         <input type = "submit" value = "Select Subject" />
      </form>
   </body>
</html>
```

The result of this code is the following form



☑ Maths ☐ Physics ☑ Chemistry

Given below is the CheckBox.java servlet program to handle input given by web browser for checkbox button.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class CheckBox extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request, 
HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Reading Checkbox Data";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
            "<head><title>" + title + "</title></head>\n" +
            "<body bgcolor = \"#f0f0f0\">\n" +
               "<h1 align = \"center\">" + title + "</h1>\n" +
               "<ul>\n" +
                  "  <li><b>Maths Flag : </b>: "
                  + request.getParameter("maths") + "\n" +
                  "  <li><b>Physics Flag: </b>: "
                  + request.getParameter("physics") + "\n" +
                  "  <li><b>Chemistry Flag: </b>: "
                  + request.getParameter("chemistry") + "\n" +
               "</ul>\n" +
            "</body>"
         "</html>"
      );
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

For the above example, it would display following result –

Reading Checkbox Data

- **Maths Flag :** : on
- **Physics Flag:** : null
- **Chemistry Flag:** : on

Reading All Form Parameters

Following is the generic example which uses **getParameterNames()** method of HttpServletRequest to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order

Once we have an Enumeration, we can loop down the Enumeration in standard way by, using *hasMoreElements()* method to determine when to stop and using *nextElement()* method to get each parameter name.

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

// Extend HttpServlet class
public class ReadParams extends HttpServlet {

   // Method to handle GET method request.
   public void doGet(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      // Set response content type
      response.setContentType("text/html");

      PrintWriter out = response.getWriter();
      String title = "Reading All Form Parameters";
      String docType =
         "<!doctype html public \"-//w3c//dtd html 4.0 " +
"transitional//en\">\n";

      out.println(docType +
         "<html>\n" +
         "<head><title>" + title + "</title></head>\n" +
         "<body bgcolor = \"#f0f0f0\">\n" +
         "<h1 align = \"center\">" + title + "</h1>\n" +
         "<table width = \"100%\" border = \"1\" align =
\"center\">\n" +
         "<tr bgcolor = \"#949494\">\n" +
            "<th>Param Name</th>"+
            "<th>Param Value(s)</th>\n"+
         "</tr>\n"
      );

      Enumeration paramNames = request.getParameterNames();

      while(paramNames.hasMoreElements()) {
         String paramName = (String)paramNames.nextElement();
         out.print("<tr><td>" + paramName + "</td>\n<td>");
         String[] paramValues =
request.getParameterValues(paramName);
```

```
         // Read single valued data
         if (paramValues.length == 1) {
            String paramValue = paramValues[0];
            if (paramValue.length() == 0)
               out.println("<i>No Value</i>");
               else
               out.println(paramValue);
         } else {
            // Read multiple valued data
            out.println("<ul>");

            for(int i = 0; i < paramValues.length; i++) {
               out.println("<li>" + paramValues[i]);
            }
            out.println("</ul>");
         }
      }
      out.println("</tr>\n</table>\n</body></html>");
   }

   // Method to handle POST method request.
   public void doPost(HttpServletRequest request,
HttpServletResponse response)
      throws ServletException, IOException {

      doGet(request, response);
   }
}
```

Now, try the above servlet with the following form –

```
<html>
   <body>
      <form action = "ReadParams" method = "POST" target =
"_blank">
         <input type = "checkbox" name = "maths" checked =
"checked" /> Maths
         <input type = "checkbox" name = "physics"  /> Physics
         <input type = "checkbox" name = "chemistry" checked =
"checked" /> Chem
         <input type = "submit" value = "Select Subject" />
      </form>
   </body>
</html>
```

Now calling servlet using the above form would generate the following result –

Reading All Form Parameters

Param Name	Param Value(s)
maths	on
chemistry	on

You can try the above servlet to read any other form's data having other objects like text box, radio button or drop down box etc.