UNIT-III
SERVLETS

Introduction to Servlets:

Servlets are server side components that provide a powerful mechanism for developing server side programs. Servlets provide component-based, platform-independent methods for building Web-based applications. Using Servlets web developers can create fast and efficient server side application which can run on any Servlet enabled web server. Servlets can access the entire family of Java APIs, including the JDBC API to access enterprise databases. Servlets can also access a library of HTTP-specific calls; receive all the benefits of the mature java language including portability, performance, reusability, and crash protection. Today Servlets are the popular choice for building interactive web applications. Servlet containers are usually the components of web and application servers, such as BEA Weblogic Application Server, IBM Web Sphere, Sun Java System Web Server, Sun Java System Application Server and others. Servlets are not designed for a specific protocol. It is different thing that they are most commonly used with the HTTP protocols Servlets uses the classes in the java packages javax.servlet and javax.servlet.http. Servlets provides a way of creating the sophisticated server side extensions in a server as they follow the standard framework and use the highly portable java language.

HTTP Servlets Typically Used To:

- Provide dynamic content like getting the results of a database query and returning to the client.
- Process and/or store the data submitted by the HTML.
- Manage information about the state of a stateless HTTP. e.g. an online shopping car manages request for multiple concurrent customers.

Methods of Servlets:

A Generic Servlet contains the following five methods:

- **init():**

  public void init(ServletConfig config) throws ServletException

  The init () method is called only once by the servlet container throughout the life of a Servlet. By this init () method the Servlet get to know that it has been placed into service. The Servlet cannot be put into the service if

  - The init () method does not return within a fix time set by the web server.
  - It throws a ServletException

  Parameters - The init () method takes a ServletConfig object that contains the initialization parameters and Servlet's configuration and throws a ServletException if an exception has occurred.

- **service():**

  public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

  Once the Servlet starts getting the requests, the service() method is called by the Servlet container to respond. The Servlet services the client's request with the help of two objects. These two objects are javax.servlet.ServletRequest and javax.servlet. Servlet Response are passed by the Servlet container. The status code of the response always

should be set for a Servlet that throws or sends an error. Parameters - The service () method takes the ServletRequest object that contains the client's request and the object ServletResponse contains the Servlet's response. The service() method throws ServletException and IOException exception.

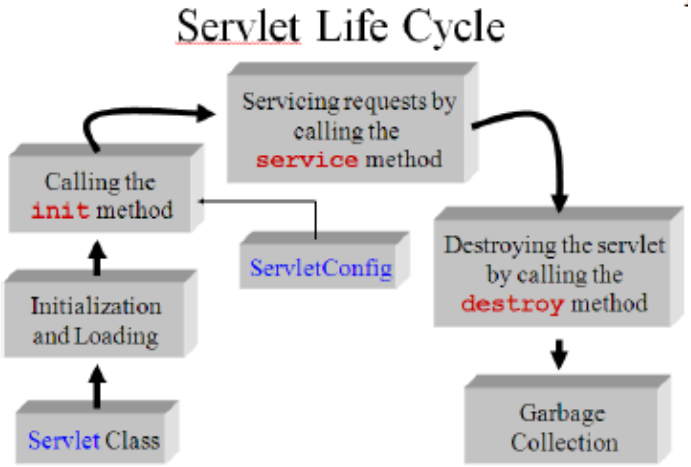
- **getServletConfig():**
  public ServletConfig getServletConfig()
  This method contains parameters for initialization and startup of the Servlet and returns a ServletConfig object. This object is then passed to the init method. When this interface is implemented then it stores the ServletConfig object in order to return it. It is done by the generic class which implements this interface.
  Returns - the ServletConfig object
- **getServletInfo():**
  public String getServletInfo ()
  The information about the Servlet is returned by this method like version, author etc. This method returns a string which should be in the form of plain text and not any kind of markup.
  Returns - a string that contains the information about the Servlet
- **destroy():**
  public void destroy()
  This method is called when we need to close the Servlet. That is before removing a Servlet instance from service, the Servlet container calls the destroy() method. Once the Servlet container calls the destroy() method, no service methods will be then called . That is after the exit of all the threads running in the Servlet, the destroy() method is called. Hence, the Servlet gets a chance to clean up all the resources like memory, threads etc which are being held.

Life cycle of Servlet:

The life cycle of a Servlet can be categorized into four parts:

1. **Loading and Instantiation:** The Servlet container loads the Servlet during startup or when the first request is made. The loading of the Servlet depends on the attribute <load-on-startup> of web.xml file. If the attribute <load-on-startup> has a positive value then the Servlet is load with loading of the container otherwise it load when the first request comes for service. After loading of the Servlet, the container creates the instances of the Servlet.
2. **Initialization:** After creating the instances, the Servlet container calls the init() method and passes the Servlet initialization parameters to the init() method. The init() must be called by the Servlet container before the Servlet can service any request. The initialization parameters persist until the Servlet is destroyed. The init() method is called only once throughout the life cycle of the Servlet. The Servlet will be available for service if it is loaded successfully otherwise the Servlet container unloads the Servlet.
3. **Servicing the Request:** After successfully completing the initialization process, the Servlet will be available for service. Servlet creates separate threads for each request. The Servlet container calls the service() method for servicing any request. The service() method determines the kind of request and calls the appropriate method (doGet () or doPost ()) for handling the request and sends response to the client using the methods of the response object.

4. **Destroying the Servlet:** If the Servlet is no longer needed for servicing any request, the Servlet container calls the destroy() method . Like the init() method this method is also called only once throughout the life cycle of the Servlet. Calling the destroy() method indicates to the Servlet container not to sent the any request for service and the Servlet releases all the resources associated with it. Java Virtual Machine claims for the memory associated with the resources for garbage collection.



The Advantages of Servlets:

1. Portability
2. Powerful
3. Efficiency
4. Safety
5. Integration
6. Extensibilty
7. Inexpensive

Each of the points is defined below:

1. **Portability:** As we know that the Servlets are written in java and follow well known standardized APIs so they are highly portable across operating systems and server implementations. We can develop a Servlet on Windows machine running the tomcat server or any other server and later we can deploy that Servlet effortlessly on any other operating system like UNIX server running on the iPlanet/Netscape Application server. So Servlets are Write Once, Run Anywhere (WORA) program.
2. **Powerful:** We can do several things with the Servlets which were difficult or even impossible to do with CGI, for example the Servlets can talk directly to the web server while the CGI programs can't do. Servlets can share data among each other, they even make the database connection pools easy to implement. They can maintain the session by using the session tracking mechanism which helps them to maintain information from request to request. It can do many other things which are difficult to implement in the CGI programs.
3. **Efficiency:** As compared to CGI the Servlets invocation is highly efficient. When the Servlet get loaded in the server, it remains in the server's memory as a single object instance. However with Servlets there are N threads but only a single copy of the Servlet

class. Multiple concurrent requests are handled by separate threads so we can say that the Servlets are highly scalable.

4. **Safety:** As Servlets are written in java, Servlets inherit the strong type safety of java language. Java's automatic garbage collection and a lack of pointers mean that Servlets are generally safe from memory management problems. In Servlets we can easily handle the errors due to Java's exception handling mechanism. If any exception occurs then it will throw an exception.
5. **Integration:** Servlets are tightly integrated with the server. Servlet can use the server to translate the file paths, perform logging, check authorization, and MIME type mapping etc.
6. **Extensibility:** The Servlet API is designed in such a way that it can be easily extensible. As it stands today, the Servlet API support Http Servlets, but in later date it can be extended for another type of Servlets.
7. **Inexpensive:** There are number of free web servers available for personal use or for commercial purpose. Web servers are relatively expensive. So by using the free available web servers you can add Servlet support to it.

DEPLOYING A SERVLET ON WEB SERVER:

- **Step 1:** First of all you need to install the Apache Tomcat Server and JDK.
  As mentioned earlier, Apache's Tomcat Server is free software available for download @ www.apache.org. You have to download the Tomcat Server 6.0. This Server supports Java Servlets 2.5 and Java Server Pages (JSPs) 2.1 specifications. Important software required for running this server is Sun's JDK (Java Development Kit) and JRE (Java Runtime Environment). The current version of JDK is 1.8. Like Tomcat, JDK is also free and is available for download at www.java.sun.com.
- **Step 2:** Next configure the Tomcat server and JDK by setting up the environment variables for JAVA_HOME variable - You have to set this variable which points to the base installation directory of JDK installation. (e.g. C:\Program Files\Java\jdk1.8.xx\bin). And CATALINA_HOME variable – you have to set this variable which points to the base installation directory of Tomcat installation. (e.g. C:\Program Files\Apache Software Foundation\Tomcatx.x\bin\.
- **Step 3:Write Your Servlet:**
  Here is simple servlet program which can be written in Notepad or EditPlus and saved using .java extension.
  PROGRAM:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class LifeCycle extends GenericServlet
{
        public void init(ServletConfig config)throws ServletException
        {
                System.out.println("init");
        }
        public void service(ServletRequest req,ServletResponse res)throws
ServletException,IOException
```

```
        {
                System.out.println("from service");
                PrintWriter out=res.getWriter();
                out.println("LifeCycle\n");
                out.println("III CSE Students");
        }
        public void destroy()
        {
                System.out.println("destroy");
        }
}
```

- **Step 4:** Then set the classpath of the servlet-api.jar file in the variable CLASSPATH inside the environment variable as "C:\Program Files\Apache Tomcat Foundation\Tomcat x.x\lib\servlet-api.jar". Then compile your servlet by using **javac LifeCycle.java.**
- **Step 5:** The next step is to create your web application folder. The name of the folder can be any valid and logical name that represents your application (e.g. bank_apps, airline_tickets_booking, shopping_cart,etc). But the most important criterion is that this folder should be created under webapps folder. The path would be similar or close to this - C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps. For demo purpose, let us create a folder called example programs under the webapps folder.
- **Step 6:** Next create the WEB-INF folder in C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\Example Programs folder.
- **Step 7:** Then create the web.xml file and the classes folder. Ensure that the web.xml and classes folder are created under the WEB-INF folder. The web.xml file contains the following information.
    - The servlet information
    - The mapping information from the server to our web application.
- **Step 8:** Then copy the Servlet class file to the classes folder in order to run the Servlet that we created. All you need to do is copy the Servlet class file (the file we obtained from Step 4) to this folder.
- **Step 9:** Edit web.xml to include Servlet's name and URL pattern. This step involves two actions viz. including the Servlet's name and then mentioning the url- pattern. Let us first see as how to include the Servlet's name in the web.xml file.

```
<web-app>
        <servlet>
                <servlet-name>MyServ</servlet-name>
                <servlet-class>LifeCycle</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>MyServ</servlet-name>
                <url-pattern>/lc</url-pattern>
        </servlet-mapping>
</web-app>
```

Note – The Servlet-name need not be the same as that of the class name. You can give a different name (or alias) to the actual Servlet. This is one of the main reasons as why this tag is used for. Next, include the url pattern using the <servlet-mapping> </servlet-mapping> tag. The url pattern defines as how a user can access the Servlet from the browser.

- **Step 10:** Run Tomcat server and then execute your Servlet by opening any one of your web browser and enter the URL as specified in the web.xml file. The complete URL that needs to be entered in the browser is: http://localhost:8080/scce/lc

Invoking Servlet using HTML:

It is a common practice to invoke servlet using HTML form. This will be achieved by the action attribute of the form tag in HTML. To understand this, we have to create a web page which will invoke above created servlet. In the above method, we have seen that the servlet is invoked from the URL, but here in this example the same servlet will be invoked by clicking the button in the web page.

HTML Program:

```
<html>
<head><title> Life Cycle of Servlet </title>
</head>
<body>
<form name="form1" action="lc">
<b> My Life Class</b>
<input type="submit" value="Go to My Life Cycle">
</form>
</body>
</html>
```

For getting the output:

1. Compile the servlet program using javac compiler.
2. Copy the generated class file into your web application's classes folder.
3. Enter the http://localhost:8080/Example%20Programs/mylifecycle.html