UNIT-III

Servlet API and Overview

Servlet Introduction:

* What is a Servlet? :-

→ Servlet is a server side technology which is used to generate dynamic web pages / web applications.

→ Servlets are server side components and are written on the server side.

→ Servlet is a Java program that extends the capabilities of a server.

→ Servlets provide a run time environment to run the web page.

→ Servlets are used to handle Http Requests and Http Responses.

→ Servlet act as middle-man between the web-server and client, takes requests, process them and returns dynamic content as response.

→ A servlet can handle many requests at once by using multiple threads.

Eg:- Examples of servlet containers / web Container are Apache Tomcat, Jetty, Glass Fish, JBoss.

* Key Features of Servlets :-

i. Platform Independent → Works on any system [OS] with Java.

ii. Reusable → One servlet can be used by many clients at the same time.

iii. Robust → They are managed by the server which reduces memory and crash issues.

iv. Web-Based → Used to build websites that can change based on the user input.

v. Extensible → Easy to extend by overriding methods like doGet() and doPost().

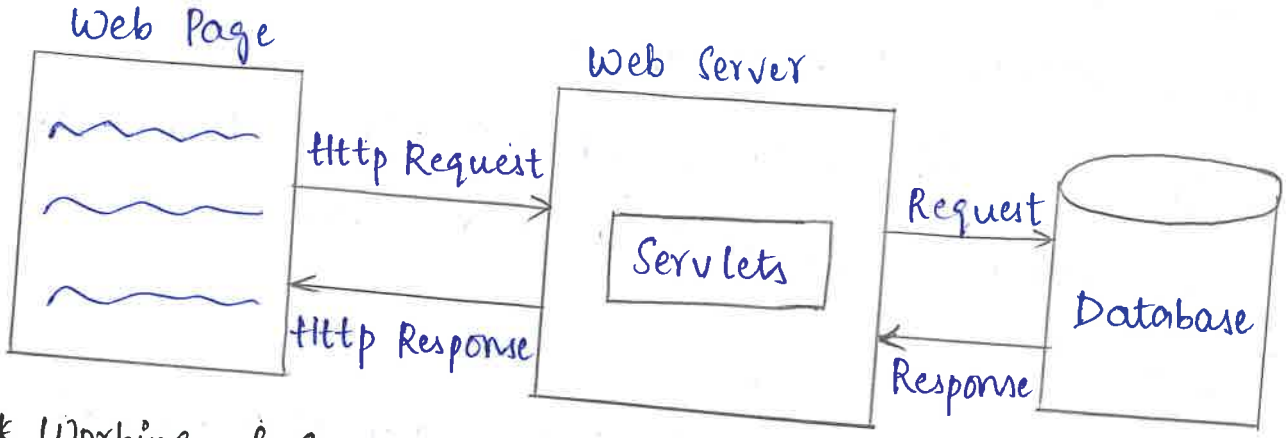
vi. Integration → Can work with databases, API's, and other Java components.

vii. Secure → Supports secure connections and user login.

viii. Efficient → Servlets handle many requests at once using threads.

ix. Lifecycle managed by server → Server controls how servlet starts, works, and ends.

* Servlet Architecture :-



* Working of Servlet Architecture :-

i. Client Request :- A web browser like Chrome sends a Http Request to the server.

ii. Web Server :- The request first reaches the web server like Glassfish or Tomcat.

iii. Servlet Container :- Inside the server, the servlet container picks up the request.

iv. Request Object :- The container creates an HttpServletRequest object with all request details like URL.

v. Servlet Execution :- The container calls the methods like doGet(), doPost() etc depending on the request.

* doGet() → Getting the data from the server.

* doPost() → Giving Sends information to the server.

* doDelete() → Used to delete the Resource at the server side.

* doPut() → Used to change the entire data at the server side.

* doPatch() → Used to edit specific data at the server side.

vi. Response Object → the servlet prepares a response and writes it to an HttpServletResponse object.

vii. Back to Client → the container sends this response back to the web-server, which returns it to the client's browser.

In simple,

Client → Web-Server → Servlet Container → Servlet (req. & resp.) → Back to client

* Advantages of Java Servlets :-

i. Fast and efficient ✓
ii. Portable ✓
iii. Secure ✓
iv. Reusable ✓
v. Easy to integrate with java apps ✓
vi. Scalable ✓
vi. Handles any types of protocols. ✓
vii. Execution is faster compared to JSP. ✓

* Disadvantages of Java Servlets :-

i. Complex for big applications.
ii. Debugging and Testing is difficult. ✓
iii. Maintanance is difficult due to large and lengthy codes. ✓
iv. Accessing is also difficult. ✓

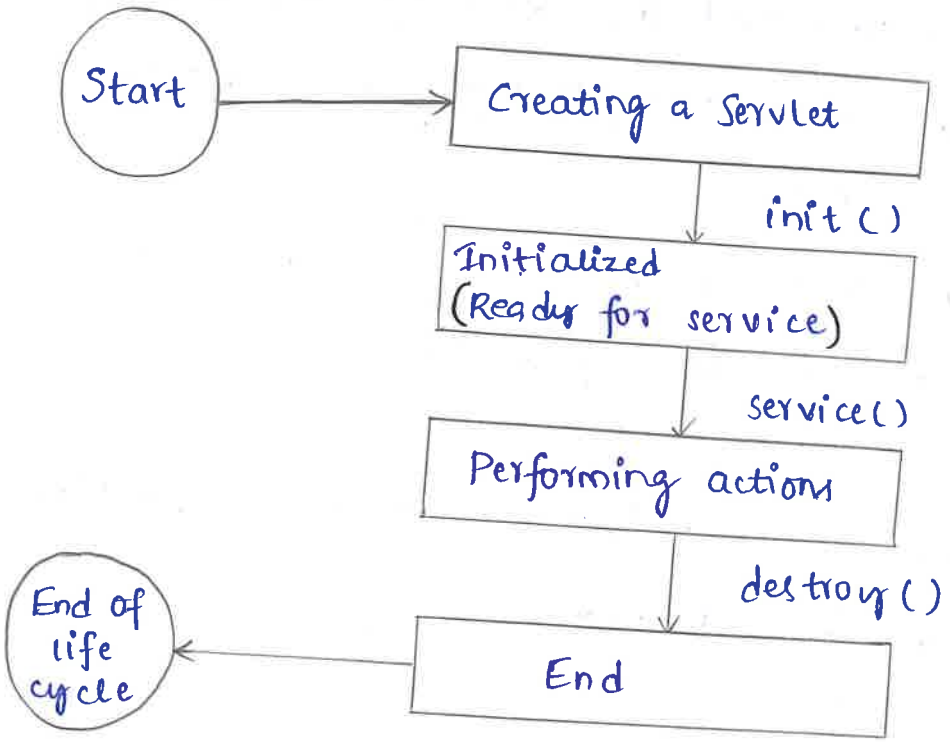
Servlet Life Cycle (SLC):

→ Servlet life cycle is the process where a servlet is created, handles request, and gets destroyed where all are managed by the web container.

→ Servlet is a server side technology which is used to create dynamic web pages/web applications.

→ Servlets are server side components and are written on the server side.

→ A servlet life cycle has mainly 5 phases and 3 methods.



Phases of a Servlet Life Cycle :

→ A servlet life cycle has mainly five phases. They are,

i. Loading

ii. Creation of servlet

iii. Initialization

iv. Performing action

v. Destruction

Load → Creation → Initialization → Action → Destruction

i. Loading a Servlet :-

→ The servlet container loads the Servlet class into memory.

→ Container makes a servlet object using a constructor.

ii. Creation of a Servlet :-

→ Container makes a servlet object using a constructor.

→ This happens only once for a given servlet.

iii. Initialization :-

→ After the servlet is created, the servlet container initializes it by calling the init() method.

iv. Performing action :-

→ After initialization, the servlet is ready to handle Http Requests and Http Responses using methods like doGet(), doPost(), doPatch(), doPut, doDelete() etc

V. Destroying a servlet :-

→ When the servlet container decides to remove the servlet, the container invokes destroy() method.

→ This method is called only once during the entire life cycle.

→ After the destroy() method completes, the servlet instance becomes ready for garbage collection by the JVM.

* The servlet life cycle consists of these stages

  * Servlet is created
  * Servlet is initialized
  * Servlet is ready to service
  * Servlet is servicing
  * Servlet is not ready to service
  * Servlet is destroyed

Life cycle of servlet methods :

init() ⟶ service() ⟶ destroy

(called only once) (handles multiple client requests & sends response) (called only once)

→ A servlet life cycle has mainly three methods. They are,

i. init() method

ii. service() method

iii. destroy() method

i. init() method :-

→ The init() method is called after the servlet creation.

→ This method can be called once during the entire life cycle.

→ The init() method is used to initialize the servlet (ready to perform actions).

Syntax :-

access specifier return type method name() { statements; }

Example :-

```
public void init()
{
  System.out.println("Servlet initialized successfully");
}
```

ii. service() method :-

→ The service() method is called after the servlet initialization.

→ This method can be called many times based on the user request.

→ The service() method is used to perform the actions.

→ The service() method contains methods like,

* doGet() → To get the data from server
* doPost() → To send information to the server
* doPut() → To change entire data at the server side
* doPatch() → To change specific data at the server side
* doDelete() → To delete the ~~entire~~ data/Resources at the server side

Syntax :-

```
access specifier  return type  method name(P/A)
{
  statements ...
  - - - - -
}
```

Example :-

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
{
   PrintWriter pw = res.getWriter();
   pw.println("IT students are talented");
}
```

iii. destroy() :-

→ The destroy() method is called after the actions are performed.

→ This method can be called only once during the entire life cycle.

→ The destroy() method is called at the end of the life cycle of the servlet to release the servlet instance or delete the servlet.

Syntax:-

```
access specifier  return type  method name
{
  statements...
  - - - - -
}
```

Example:-

```
public void destroy()
{
  System.out.println("Servlet is deleted");
}
```

```
//Example Program

package Rama;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Sitha extends HttpServlet/GenricServlet
{
  public void init() throws ServletException
  {
    System.out.println("Servlet initialized");
  }
  public void service doGet(HttpServletRequest req, HttpServletResponse
    res) throws ServletException, IO Exception
  {
    PrintWriter pw.= res.getWriter();
    pw.println("IT students are good");
  }
  public void destroy()
  {
   System.out.println("Servlet deleted");
  }
}
```

***Types of Servlets :

→ Servlets are generally two types. They are,

i. HttpServlets

ii. GenericServlets

* Servlet is Server side Technology
* Servlets creating Dynamic web pages

i. HttpServlets :-

XX → HttpServlet is the subclass of GenericServlet.

② → The servlet class is called as javax.servlet.HttpServlet.

① → HttpServlets are designed to handle only the HttpRequests and HttpResponses. [only for HTTP protocol]

③ → HttpServlets are more useful in the web-server environment.

XX → These are specialized for handling HttpRequests and HttpResponses.

④ → It contains methods like doGet(), doPost(), doPatch(), doDelete(), doPut() etc

⑤ → HttpServlets are commonly used for building the web applications due to its built-in support for Http Protocols.

//Servlet code

Note:-
doGet(HttpServletRequest req,
HttpServletResponse resp

→ HttpServlet contains methods like,

* doGet() → To get the data from server
* doPost() → To give the information to the server
* doPut() → To change the entire data at the server side
* doPatch() → To change specific data at the server side
* doDelete() → To delete the entire data at the server side.

//Servlet Code for HttpServlets

```
package Rama;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Sitha class extends HttpServlet
{
  private static final long serialVersionUID = 1L;
  protected void doGet(HttpServlet Request req,HttpServlet Response
  res) throws ServletException, IO Exception
  {
    Print Writer pw = res.getWriter();
    pw.println("IT Students are talented");
  }
}
```

// XML Code for HttpServlets

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:
SchemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID"
version="3.0">
 <display-name>ITServlets</display-name>
 <servlet>
 <servlet-name>Sitha</servlet-name>
 <servlet-class>Rama.Sitha</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>Sitha</servlet-name>
 <url-pattern>/IT</url-pattern>
 </servlet-mapping>
 </web-app>
```

Advantages of HttpServlets:

i. Performance ✓
ii. Portability ✓
iii. Robustness ✓
iv. Efficiency ✓
v. Security ✓
vi. Integration ✓
vii. Session Management ✓

Disadvantages of HttpServlets:

i. Complexity ✓
ii. Learning Curve
iii. Stateless Nature
iv. Deployment
v. Concurrency Challenges
vi. Dynamic content handling
vii. Lack of built-in templating

✓ * only support for http protocol [http request and http response]
✓ * larger code
*
*

ii. Generic Servlets :-

→ Generic servlets are protocol independent

→ They can handle requests from various protocols.

→ Generic servlet is an abstract class that implements the Servlet and ServletConfig interfaces.

→ Generic servlets are less commonly used in the modern web development, as most web applications use Http protocols.

→ Generic servlets uses only service() method, inted of doGet(), doPost(), doDelete() etc...

* For both servlet types we must implement the initializer method init() to allocate and initialize the servlet, and the destructor method destroy() to de-allocate the resources.

→ Generic Servlet offers a fundamental framework for servlet development.

→ Generic servlets can handle various protocols such as HTTP, FTP, SMTP.

```
//Servlet Code for GenericServlets

package Rama;

import java.io.*;

import javax.servlet.*;

public class Sitha extends GenericServlet
{
  private static final long serialVersionUID=1L;
  protected void service(ServletRequest req,ServletResponse res)
   throws ServletException,IOException
  {
    PrintWriter pw=res.getWriter();
    pw.println("IT Students are brave");
  }
}
```

// XML code for GenericServlets

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:Schema
Location="http://java.sun.com/xml/ns/javaeehttp://javasun.com/
xml/ns/javaee/web-app-3-0.xsd" id="WebApp_ID" version="3.0">
<display-name>ITServlets</displayname>
<servlet>
<servlet-name>Sitha</servlet-name>
<servlet-class>Rama.Sitha</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Sitha</servlet-name>
<url-pattern>/IT</url-pattern>
</servlet-mapping>
</web-app>
```

Advantages of Generic Servlets:

i. Protocol Independence. ✓
ii. Fundamental Servlet Lifecycle. ✓
iii. Simplicity for Non-Http needs. ✓

Disadvantages of Generic Servlets:

i. Lack of Http-specific features. ✓
ii. Limited practical use in web development. ✓

Servlet Configuration with Deployment Descriptor :

→ A deployment descriptor is an XML-based configuration file.

→ It defines how a web application should be deployed and managed by a web-server /server.

→ For standard java applications this file is named as web.xml.

→ The web.xml file resides in the WEB-INF directory of the application's WAR

→ In general, Java web applications uses a deployment descriptor file to

→ web.xml file contains various elements that describe the mapping of URL's to the servlets

→ web.xml file include

* servlet and jsp servlet mapping
* init parameters
* welcome files
* error pages
* security constraints
* filter and listeners

→ A deployment descriptor tells the server like Tomcat how to run your servlets.

→ It is used to tell the server that,

* Which class is a servlet.
* What URL will call that servlet
* Any extra settings like welcome pages, error pages

→ The web.xml file (deployment descriptor) usually has three main parts. They are,

i. Servlet Declaration
ii. Servlet Mapping
iii. Other configurations (init parameters, context parameters, welcome files, error pages, security settings)

i. Servlet Declaration :-

→ Servlet Declaration defines the servlet class.

→ This is used to tell the server that " which java class is a servlet. "

→ In servlet declaration, we give name and the class path.

Eg:-

```
<servlet>
<servlet-name> Rama </servlet-name> //servlet class name
< servlet-class > IT.Rama </servlet-class> // fully qualified
</servlet >
```

ii. Servlet Mapping:-

→ Servlet Mapping is used to connect servlets with the URL.

→ This tells the server that " when someone visits this URL, run that servlet."

→ It connects the name from declaration i.e. <servlet-name> with a URL pattern.

→ One servlet can have many URL mappings that means same servlet can be called using different URL's.

Eg:-

<servlet-mapping>

<servlet-name> Rama </servlet-name> //servlet class name

<servlet><url-pattern> /Sitha </url-pattern> // url link

</servlet-mapping>

iii. Other configurations:-

→ Other configurations gives the extra information or settings for the app or servlets.

* Init parameters → Values for one servlet
* Context parameters → Values for the whole app
* Welcome File → First page to show (like index.html)
* Error Pages → Custom error messages.

Eg:-

<init-param>

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

<param-value> Admin </param-value>

</init-param>

Web.xml code :

```
<?xml version="1.0" encoding="UTF-8" ?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" xsi:SchemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
<display-name> projectname [IT] </display-name>
<servlet>
<servlet-name> Rama </servlet-name>
<servlet-class> IT.Rama </servlet-class>
</servlet>
<servlet-mapping>
<servlet-name> Rama </servlet-name>
<url-pattern> /Sitha </url-pattern>
</servlet-mapping>
</web-app>
```