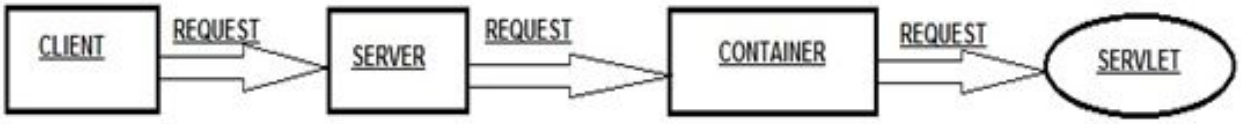
Unit - 7 Servlet and Java Server Pages (JSP)

Introduction

- A servlet program is a java program that is executed on the webserver. It is used to execute the process at the server side
- Sevlet accepts a request from a client, performs some task and returns a result to client
- Servlet runs on server and will respond to request from the client in the form of HTML pages



- To run and develop servlet program we need the following:
  - JDK
  - Web browser (Chrome)
  - Web Server (Tomcat)
- Basic structure

```
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletDemo extends HttpServlet{
        public void doGet(HttpServletRequest req,HttpServletResponse res)
        Throws ServletException{
                // code goes here
        }
}
```

The life cycle of servlet

- Lifecycle of servlet describes how and when a servlet is loaded, able to handle requests and destroyed
- A servlet life cycle can be defined as the entire process from its creation till the destruction.
- The following are the paths followed by a servlet
    - The servlet is initialized by calling the init () method.
    - The servlet calls service() method to process a client's request.
    - The servlet is terminated by calling the destroy() method.
    - Finally, servlet is garbage collected by the garbage collector of the JVM.

init() method:

- The init method is called only once.
- It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet
- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate.
- The init() method simply creates or loads some data that will be used throughout the life of the servlet.
- The init method definition looks like this:

```
public void init() throws ServletException
{
        // Initialization code...
}
```

service() method:

- The service() method is the main method to perform the actual task.

- The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server creates a new thread and calls service.
- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.
- Here is the signature of this method:

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

}
```

doGet() method:

- A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException
{
        // Servlet code
}
```

doPost() method:

- A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method

```
public void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException
{
        // Servlet code
}
```

destroy() method:

- The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists, and perform other such cleanup activities.
- After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy()
{
        // Finalization code...
}
```

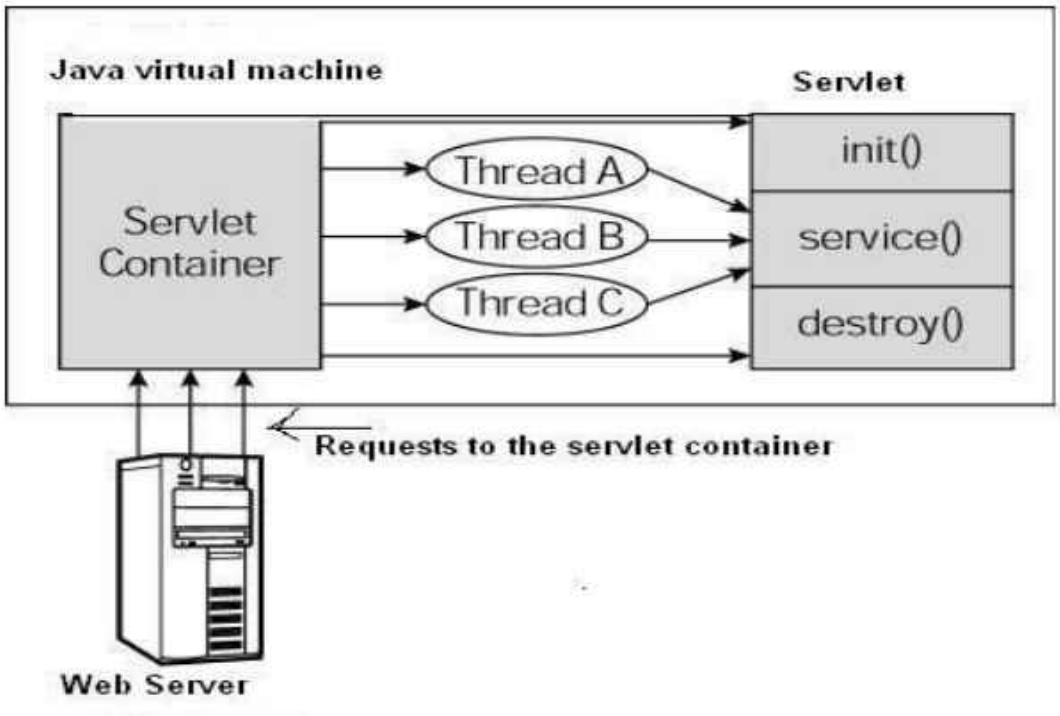


Figure: Scenario of Life cycle of servlet

b> Summary

→ init() :- initializes the servlet
(initialization params may be passed)

→ service() :- process the HTTP request

→ destroy() :- to destroy the servlet.

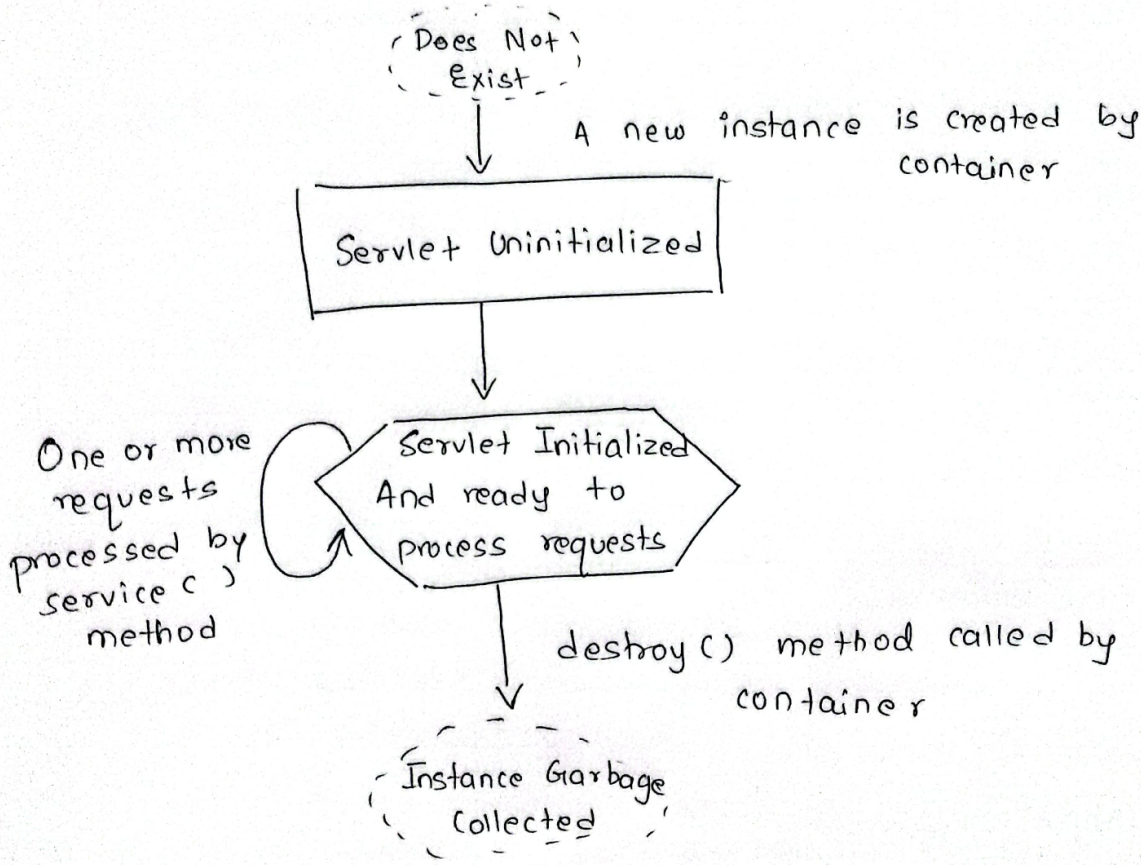


Fig: Lifecycle of a servlet.

(3) The Servlet API

→ contains packages that are required to build servlets.

→ javax.servlet and javax.servlet.http are the packages that constitute servlet API.

→ They are not part of core java rather they are extended packages. Hence they are not contained in JDK (so we need Tomcat or GlassFish server)

a) javax.servlet package

→ contains number of interfaces and classes that are foundation for servlet operation.

Interface

i) Servlet → declares life cycle methods of servlet

ii) ServletConfig → allows servlet to get initialization or config data

iii) ServletContext → enables to obtain log events and environment info.

iv) ServletRequest → used to read data from client request

v) ServletResponse → used to write data to client responce

Class

i) GenericServlet → implements servlet & servletConfig

ii) ServletInputStream → provides input stream to read data from client request.

iii) ServletOutputStream → output stream to write data to client response

iv) servletException → indicates servlet problem occured

v) UnavailableException → indicates servlet is unavailable

Servlet Interface:

All servlet must implement Servlet Interface.
It declares init(), service() and destroy() methods

void init() → when servlet is initialized
void service() → read request from client, write response to client.
void destroy() → unload servlet

ServletConfig getServletConfig() → returns servletConfig object that contains any initialization parameters

String getServletInfo() → returns servlet info in string

ServletConfig Interface:

It allows a servlet to obtain configuration data when it is loaded.

ServletContext getServletContext() → returns context for servlet

String getInitParameter(String param) → returns initialization param value

String getServletName() → returns name of servlet invoked

ServletContext Interface:

Enables servlet to obtain information about their environment.

Object getAttribute(String attr) → returns value of attribute
String getMimeType(String file) → returns MIME type of file
String getRealPath(String vpath) → returns absolute path.
String getServerInfo() → returns info about server
void log(String s) → writes string to servlet log.

ServletRequest Interface:

Enables a servlet to obtain info about client request.

Object getAttribute(String attr) → returns value of attribute
String getCharacterEncoding() → returns character encoding
int getContentLength() → returns size of request (1 for unknown)
String getContentType() → returns type of request (null)
String getParameter(String pname) → returns value of parameter
String getProtocol() → returns description of the protocol
String getRemoteAddr() → returns client IP
String getRemoteHost() → returns client host name
String getServerName() → returns name of server
String getServerPort() → returns port number
ServletInputStream getInputStream() → to read binary data from the request
BufferedReader getReader() → to read text from request.

ServletResponse Interface:

Enables servlet to handle response for a client.

String getCharacterEncoding() → returns character encoding

void setContentLength(int size) → set content length for response size

void setContentType(String type) → sets content type for response

ServletOutputStream getOutputStream() → used to write binary data to response

PrintWriter getWriter() → used to write character data to the response.

Generic Servlet Class:

It provides implementations of basic life cycle methods for a servlet. It implements Servlet and ServletConfig interfaces.

Additional methods are available i.e

```
void log(String s)
void log(String s, Throwable e)
```

They are used to append string to server log file. s is the string to append and e is exception that occured.

ServletInputStream class :

It extends InputStream. It is implemented by the servlet container and provides ~~and~~ an input stream that servlet developer can use to read the data from client request. In addition to methods provided in InputStream, the additional method is provided i.e

int readLine(byte[] buffer, int offset, int size)

Here, buffer is array in which bytes are placed starting at offset which are of size bytes.

ServletOutputStream class :

It extends OutputStream class. It is implemented by the servlet container and provides an output stream that a servlet developer can use to write data to client response. It provides print() and println() methods that are overloaded like void print(boolean b), void print(char c), void println(), void println(int i), etc.

b) javax.servlet.http package

→ contains a number of interfaces and classes that are commonly used by servlet developers.

Interface

i, HttpServletRequest → to read data from HTTP request
ii, HttpServletResponse → to write data from HTTP res.
iii, HttpSession → to read and write session data
iv, HttpSessionBindingListener → informs if an object is bound to session or not.

Class

i, Cookie → allows state to be stored on client
ii, HttpServlet → to handle HTTP req and res
iii, HttpSessionEvent → encapsulate session-change event
iv, HttpSessionBindingEvent → indicates when a listener is bound to or unbound from a session value, or that a session attribute changed.

HttpServletRequest Interface:

String getAuthType() → returns authentication scheme.
Cookie[] getCookies() → returns array of cookies
long getDateHeader(String field) → returns value of date header
String getHeader(String field) → returns header value
String getMethod() → returns HTTP method for a request
String getPathInfo() → path info after servlet path and before query string
String getQueryString() → returns query string in URL.
String getRemoteUser() → returns name of user issuing request.


OBJ

String getRequestedSessionId() → returns ID of session
String getRequestURI() → returns the URI
String getServletPath() → returns URL that is part of servle
HttpSession getSession() → returns session and created if not
HttpSession getSession(boolean new) → if new is true ~~then~~ and no session exists, creates and returns a session for this request. Otherwise, returns exisisting session for this request.
boolean isRequestedSessionIdFromCookie() → returns true if cookie contain the session ID. Else returns false.
boolean isRequestedSessionIdFromURL() → returns true if URL contains session ID. Else returns false
boolean isRequestedSessionIdValid() → returns true if the requested session ID is valid

~~###~~ HttpServletResponse Interface:

void addCookie(Cookie cookie) → adds cookie to the HTTP response
boolean containsHeader(String field) → returns true if HTTP response header contains field
void sendError(int e) throws IOException → send error to client
void sendError(int c, String s) → send error code and message
void sendRedirect(String url) → redirects the client to url
void setHeader(String field, String value) → add header with value
void setIntHeader(String field, int value) → add header with valu
void setStatus(int code) → set status code for response

Writing a Servlet Program

Reading Servlet Parameters

- The ServletRequest class includes methods that allow to read the names and values of parameters that are included in a client request.
- We will develop a servlet that illustrates their use. The example contains two files.
- A Web page is defined in index.html and a servlet is defined in GreetingServlet.java.
- The HTML source code for index.html is shown in the following listing.
- It defines a simple label with input field. There is also a submit button.
- Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP GET request.

// index.html

```
<html lang="en">
   <head>
       <title>Greeting Demo</title>
   </head>
<body>
   <form action="GreetingServlet" method="GET">
       <label for="name">Enter your name:</label> <br>
       <input type="text" name="name" id="name">
       <input type="submit">
   </form>
</body>
</html>
```

// GreetingServlet.java

```
package com.example.servetpractise;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;

public class GreetingServlet extends HttpServlet {
   protected void doGet(HttpServletRequest req,
HttpServletResponse res) throws ServletException,
IOException {
       String name = req.getParameter("name");

       res.setContentType("text/html");

       PrintWriter out = res.getWriter();

       out.println("<h1>Good morning "+name+"!</h1>");

   }
}
```

// web.xml (Deployment Descriptor)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
         version="5.0">
    <servlet>
        <servlet-name>greeting</servlet-name>

<servlet-class>com.example.servetpractise.GreetingServlet</se
rvlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>greeting</servlet-name>
        <url-pattern>/GreetingServlet</url-pattern>
    </servlet-mapping>

</web-app>
```

Handling HTTP GET Requests

- Here we will develop a servlet that handles an HTTP GET request. The servlet is invoked when a form on a Web page is submitted.

// same as above

The Cookie Class

- The Cookie class encapsulates a cookie. A cookie is stored on a client and contains state information.
- Cookies are valuable for tracking user activities. For example, assume that a user visits an online store.
- A cookie can save the user's name, address, and other information. The user does not need to enter this data each time he or she visits the store.
- A servlet can write a cookie to a user's machine via the addCookie( ) method of the HttpServletResponse interface.
- The data for that cookie is then included in the header of the HTTP response that is sent to the browser.
- The names and values of cookies are stored on the user's machine.
- Some of the information that is saved for each cookie includes the following:
    - The name of the cookie
    - The value of the cookie
    - The expiration date of the cookie
    - The domain and path of the cookie
- The expiration date determines when this cookie is deleted from the user's machine.
- If an expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends. Otherwise, the cookie is saved in a file on the user's machine.
- The domain and path of the cookie determine when it is included in the header of an HTTP request.
- If the user enters a URL whose domain and path match these values, the cookie is then supplied to the Web server. Otherwise, it is not.
- Constructor:
  Cookie(String name, String value): Constructs a new cookie with the specified name and value.
- **Basic Properties:**

setName(String name): Sets the name of the cookie.
getName(): Returns the name of the cookie.
setValue(String value): Sets the value of the cookie.
getValue(): Returns the value of the cookie.

- **Additional Properties:**
  setDomain(String domain): Sets the domain of the cookie.
  getDomain(): Returns the domain of the cookie.
  setPath(String path): Sets the path of the cookie.
  getPath(): Returns the path of the cookie.

  setMaxAge(int maxAge): Sets the maximum age of the cookie in seconds. A negative value means the cookie is not stored persistently, and a value of 0 means the cookie should be deleted.

  getMaxAge(): Returns the maximum age of the cookie in seconds.

  setSecure(boolean secure): Indicates whether the cookie should only be sent over secure connections (HTTPS).

  getSecure(): Returns true if the cookie should only be sent over secure connections.
- **Sending and Retrieving Cookies:**
  setComment(String purpose): Sets the comment describing the purpose of the cookie.
  getComment(): Returns the comment describing the purpose of the cookie.
  setHttpOnly(boolean httpOnly): Marks or unmarks the cookie as HTTP-only.
  isHttpOnly(): Checks whether the cookie is marked as HTTP-only.

Using Cookies

- Now, let's develop a servlet that illustrates how to use cookies. The servlet is invoked when a form on a Web page is submitted. The example contains three files as summarized here:
    - Index.html - Allows a user to specify a value for the cookie named MyCookie.
    - AddCookie.java - Processes the submission of AddCookie.htm.
    - GetCookie.java - Displays cookie values.

```
//index.html

<!DOCTYPE html>
<html>
<head>
   <title>Set Cookie</title>
</head>
<body>
   <form action="AddCookie" method="post">
      Enter value for MyCookie: <input type="text" name="cookieValue">
      <input type="submit" value="Set Cookie">
   </form>
</body>
</html>
```

```
//AddCookie.java

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

@WebServlet("/AddCookie")
public class AddCookie extends HttpServlet {
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
```

```
        throws ServletException, IOException {

    String cookieValue = request.getParameter("cookieValue");

    // Create a new cookie
    Cookie myCookie = new Cookie("MyCookie", cookieValue);

    // Set the cookie's maximum age to 24 hours (in seconds)
    myCookie.setMaxAge(24 * 60 * 60);

    response.addCookie(myCookie);

    // Redirect to the GetCookie servlet to display the cookie values
    response.sendRedirect("GetCookie");
  }
}
```

```
// GetCookie.java

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;

@WebServlet("/GetCookie")
public class GetCookie extends HttpServlet {
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    // Get the cookies from the request
    Cookie[] cookies = request.getCookies();

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html><head><title>Get Cookie</title></head><body>");

    if (cookies != null) {
      out.println("<h2>Cookie Values:</h2>");

      for (Cookie cookie : cookies) {
        out.println("<p><strong>" + cookie.getName() + ":</strong> " + cookie.getValue() + "</p>");
      }
```

```
        } else {
            out.println("<h2>No cookies found.</h2>");
        }

        out.println("</body></html>");
    }
}
```

// deployment descriptor

```
<servlet>
    <servlet-name>AddCookie</servlet-name>
    <servlet-class>getDemo.AddCookie</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>AddCookie</servlet-name>
    <url-pattern>/AddCookie</url-pattern>
 </servlet-mapping>
  <servlet>
    <servlet-name>GetCookie</servlet-name>
    <servlet-class>getDemo.GetCookie</servlet-class>
 </servlet>
 <servlet-mapping>
    <servlet-name>GetCookie</servlet-name>
    <url-pattern>/GetCookie</url-pattern>
 </servlet-mapping>
```

Handling HTTP Requests and Responses

- The HttpServlet class provides specialized methods that handle the various types of HTTP requests.
- A servlet developer typically overrides one of these methods. These methods are **doDelete( ), doGet( ), doHead( ), doOptions( ), doPost( ), doPut( ), and doTrace( ).**

Session Tracking

- HTTP is a stateless protocol.
- Each request is independent of the previous one.
- However, in some applications, it is necessary to save state information so that information can be collected from several interactions between a browser and a server. Sessions provide such a mechanism.
- A session can be created via the getSession( ) method of HttpServletRequest. An HttpSession object is returned.
- This object can store a set of bindings that associate names with objects. The setAttribute( ), getAttribute( ), getAttributeNames( ), and removeAttribute( ) methods of HttpSession manage these bindings.
- It is important to note that session state is shared among all the servlets that are associated with a particular client.
- Below is the demo of using session:

// Session.html

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Session Demo</title>
</head>
<body>
	<label style="color: blue"><b>Session Demo</b></label>
	<label><b>Click below to get Session Value</b></label>
	<a href="getSession">click here</a>
</body>
</html>
```

// GetSession.java

```
package getDemo;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import jakarta.servlet.ServletException;
```

```java
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
public class GetSession extends HttpServlet{
	protected void doGet(HttpServletRequest request, HttpServletResponse
response)
			throws ServletException, IOException {
			response.setContentType("text/html;charset=UTF-8");
			PrintWriter out = response.getWriter();
			try {
			// Get the HttpSession object.
			HttpSession hs = request.getSession(true);

			// Display date/time of last access.
			Date date = (Date)hs.getAttribute("date");
			// Display current date/time.
			out.println("<html>");
			out.println("<head>");
			out.println("<title>Servlet GetSession</title>");
			out.println("</head>");
			out.println("<body>");
			if(date != null) {
				out.print("Last access: " + date + "<br>");
			}
			date = new Date();
			hs.setAttribute("date", date);
			out.println("Current date: " + date);
			out.println("</body>");
			out.println("</html>");
			} finally {
				out.close();
			}
	}
}
```