Servlets

1. Servlets are used to develop dynamic web applications.
2. Servlets are nothing but the Java programs which reside on the server side and their main purpose is to serve the client request.
3. Servlets are fully compatible with Java. You can use any of the available Java APIs like JDBC inside the servlets.
4. As servlets are written in Java, they are platform independent, robust, and secured.
5. In Servlets, a thread is created for each request unlike in CGI where a process is created for each request. Hence, servlets give better performance than CGI.
6. Servlets are protocol independent. i.e. they support FTP, SMTP, HTTP etc. protocols.

Java Servlets Architecture :

Step 1 : Client i.e. web browser sends the request to the web server.

Step 2 : Web server receives the request and sends it to the servlet container. Servlet container is also called web container or servlet engine. It is responsible for handling the life of a servlet.

Step 3 : Servlet container understands the request's URL and calls the particular servlet. Actually, it creates a thread for execution of that servlet. If there are multiple requests for the same servlet, then for each request, one thread will be created.

Step 4 : Servlet processes the request object and prepares response object after interacting with the database or performing any other operations and sends the response object back to the web server.

Step 5 : Then web server sends the response back to the client.

Servlet Container

Servlet container, also known as Servlet engine, is an integrated set of objects that provide a run time environment for Java Servlet components. In simple words, it is a system that manages Java Servlet components on top of the Web server to handle the Web client requests.

Services provided by the Servlet container:

- **Network Services:** Loads a Servlet class. The loading may be from a local file system, a remote file system or other network services. The Servlet container provides the network services over which the request and response are sent.
- **Decode and Encode MIME-based messages**: Provides the service of decoding and encoding MIME-based messages.
- **Manage Servlet container:** Manages the lifecycle of a Servlet.
- **Resource management**: Manages the static and dynamic resources, such as HTML files, Servlets, and JSP pages.
- **Security Service:** Handles authorization and authentication of resource access.
- **Session Management:** Maintains a session by appending a session ID to the URL path

Advantages Of Servlets :

1. As servlets support all protocols like FTP, SMTP, HTTP etc. they can be used to develop any kind of web applications like E-commerce, Content management systems, chat based or file based web applications etc.
2. As servlets are fully compatible with Java, you can make use of wide range of available Java APIs inside the servlets.
3. As they run on Java enabled servers, You need not to worry about garbage collection and memory leaks. JVM handles them for you.
4. Since servlets are written in Java, they are portable and platform independent. You can run them on any operating systems and on any web servers available today.
5. Servlets inherit security features from JVM and web server.
6. As servlets are written in Java, you can extend them according to your requirements.
7. As servlets are compiled into bytecodes, they are faster than any other server-side scripting languages.

Servlet life cycle

The servlet life cycle describes the stages a servlet goes through from its creation to its destruction within a web container. It involves loading the servlet class, creating an instance, initialization, handling requests, and finally, destruction.

Servlet life cycle contains five steps:

1. Loading of Servlet
2. Creating instance of Servlet
3. Invoke init() once
4. Invoke request service() repeatedly for each client
5. Invoke destroy()

1. Loading and Instantiation:

The web container loads the servlet class and creates an instance of it. This happens when the web application starts or when the servlet is first accessed, depending on how it's configured.

2. Initialization:

The init() method is called once per servlet instance, usually when the servlet or web application starts. This method is used for setting up resources like database connections or configuration parameters.

3. Servicing Requests:

For every client request, the servlet container calls the service() method of the servlet.

The service() method processes the request and decides whether to call **doGet(), doPost(),** or other specific methods based on the HTTP request type.

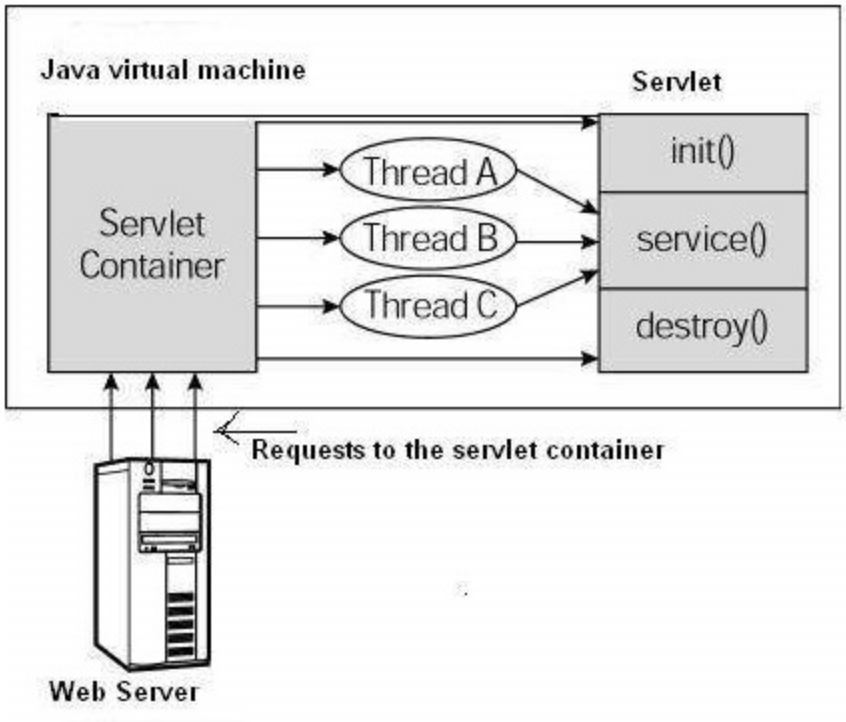
4. Destruction:

When the servlet is no longer needed, the web container calls the destroy() method.

The purpose is to clean up resources like closing database connections or releasing memory.

Summary of Methods:

- **init()**: Initialization phase (runs once).
- **service()**: Request handling phase (runs for every request).
- **destroy()**: Termination phase (runs once when the servlet is no longer needed).



Servlets APIs and Packages

Servlets are built from two packages:

- **jakarta.servlet(Basic):** Provides basic Servlet classes and interfaces.
- **jakarta.servlet.http(Advance):** Advanced classes for handling HTTP-specific requests.

What is CGI (Common Gateway Interface)?

CGI is a way for a web server to run a separate program, written in languages like C or C++, to handle requests from users and create dynamic web content.

How CGI works when you request a dynamic web page:

1. The web server finds the CGI program using the web address (URL) you provided.
2. It starts a new process to handle your request.
3. The server runs the CGI program and gives it the request details.
4. The CGI program processes the request and creates a response.
5. The server collects the response, closes the process, and sends the result back to you as a web page.

Difference Between Java Servlets and CGI

The table below demonstrates the difference between servlet and CGI

Servlet	CGI (Common Gateway Interface)
Servlets are portable and efficient.	CGI is not portable.
In Servlets, sharing data is possible.	In CGI, sharing data is not possible.
Servlets can directly communicate with the webserver.	CGI cannot directly communicate with the webserver.
Servlets are less expensive than CGI.	CGI is more expensive than Servlets.
Servlets can handle the cookies.	CGI cannot handle the cookies.

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( )**.

Handling HTTP GET Requests

we will develop a servlet that handles an HTTP GET request. The servlet is invoked when a form on a web page is submitted. The example contains two files. A web page is defined in **ColorGet.html**, and a servlet is defined in **ColorGetServlet.java**.

```
<html>
<body>
<center>
<form name="Form1"
action="http://localhost:8080/examples/servlets/servlet/ColorGetServlet">
<B>Color:</B>
<select name="color" size="1"> <option value="Red">Red</option> <option
value="Green">Green</option>
<option value="Blue">Blue</option> </select>
<br><br>
<input type=submit value="Submit"> </form>
</body>
</html>
```

ColorGetServlet.java

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


public class ColorGetServlet extends HttpServlet {


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

String color = request.getParameter("color"); response.setContentType("text/html"); PrintWriter
pw = response.getWriter(); pw.println("<B>The selected color is: "); pw.println(color);

pw.close();

}

}
```

index.html:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
 <form action="add" method="post">
  Enter num1: <input type ="text" name = "num1"><br>
  Enter num2: <input type ="text" name = "num2"><br>
  <input type="submit">
 </form>
</body>
</html>
```

AddServlet.Java

```
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 AddServlet extends HttpServlet {


   public void doPost(HttpServletRequest req,
                   HttpServletResponse res)
      throws IOException
   {
      int num1= Integer.parseInt(req.getParameter("num1"));
      int num2 = Integer.parseInt(req.getParameter("num2"));
      int result = num1 + num2;
      PrintWriter out = res.getWriter();
      out.println("The result is = " + result);
   }
}
```

Redirecting Requests to Other Resources or Servlet - sendRedirect() Method :

While processing the request, let's say if need to call another servlet from a different server, we need to redirect the response to that resource. To achieve this, Java servlets provide sendRedirect() method in HttpServletResponse interface in ***javax.servlet.http*** package. To understand better, let's look at some real-time examples.

Example 1: Nowadays, there are so many online shopping sites, where we can purchase goods. Once we select the product, are ready to purchase, and click on pay, the browser will redirect to the respective online payment page. Here, the response from the shopping website redirects it to the payment page and a new URL can be seen in the browser.

sendRedirect() Method

sendRedirect() method redirects the response to another resource, inside or outside the server. It makes the client/browser to create a new request to get to the resource. It sends a temporary redirect response to the client using the specified redirect location URL.

Syntax:

void sendRedirect(java.lang.String location) throws java.io.IOException

Index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Home</title>
</head>
<body>
   <form action="redirect" method="get">
   <h2>sendRedirect() method in Java Servlets</h2>
<p>
        sendRedirect() method, redirects the client request from the one
        servlet to another servlet. <br> It creates a new request from
        the client browser for the resource.
      </p>

For More information <input type="submit" value="Click Here">
</form>
</body>
</html>
```

We have a form with action ="redirect" method = "get", so Index.html maps the servlet having '/redirect' URL and executes the 'doGet' method in it.

ServletRedirect.java

```
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/redirect")
public class ServletRedirect extends HttpServlet {

   @Override
   protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {

      resp.sendRedirect("https://www.aktu.ac.in");

   }

}
```

Session tracking

HTTP is a "stateless" protocol, which means that each time a client requests a Web page, the client establishes a new connection with the Web server, and the server does not retain track of prior requests.

- The conversation of a user over a period of time is referred to as a **session**. In general, it refers to a certain period of time.
- The recording of the object in session is known as **tracking**.
- **Session tracking** is the process of remembering and documenting customer conversations over time. Session management is another name for it.
- The term "**stateful web application**" refers to a web application that is capable of remembering and recording client conversations over time.

Session Tracking Techniques:

These methods allow servlets to associate requests with a specific user's session:

- **Cookies:** Small pieces of data stored on the user's browser. The server can include a session ID in a cookie, and the browser sends it back with subsequent requests.
- **Hidden Form Fields:** HTML form elements that store data that is not visible to the user but is sent with the form submission. This can include a session ID.

```
<input type = hidden'  name = 'session' value = '12345' >
```

- **URL Rewriting:** Modifying URLs to include session IDs. This is often used when cookies are not available or disabled.

- **HttpSession:**
  The HttpSession object is provided by the servlet container to manage session data. Servlets can access and modify data associated with a user's session through the HttpSession object.

```
HttpSession session = request.getSession( );
Session.setAttribute("username", "password");
```

Storing Data in a Session using HttpSession:

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

public class SessionExample extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException {
      // Get the current session or create a new one
      HttpSession session = request.getSession();
      session.setAttribute("userName", "Ashish123"); // Store data in the session

      // Send response to the client
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<h1>Session Created Successfully</h1>");
   }
}
```

Comparison: Cookies vs. HttpSession

Feature	Cookies	HttpSession
Data Storage	Stored in the client's browser.	Stored on the server.
Size	Limited in size (~4KB).	No strict size limit.
Security	Less secure (stored on the client).	More secure (stored on the server).
Usage	Used for lightweight tracking.	Used for complex, server-side tracking.

Introduction to JSP

JavaServer Pages (JSP) is a server-side technology that creates dynamic web applications. It allows developers to embed Java code directly into HTML pages, and it makes web development more efficient.
JSP is an advanced version of Servlets. It provides enhanced capabilities for building scalable and platform-independent web pages

Advantages of JSP over Servlets
JSP simplifies web development by combining the strengths of Java with the flexibility of HTML.
Some advantages of JSP over Servlets are listed below:

- JSP code is easier to manage than Servlets as it separates UI and business logic.
- JSP minimizes the amount of code required for web applications.
- Generate content dynamically in response to user interactions.
- It provides access to the complete range of Java APIs for robust application development.
- JSP is suitable for applications with growing user bases.

Features of JSP

- It is platform-independent; we can write once, run anywhere.
- It simplifies database interactions for dynamic content.
- It contains predefined objects like request, response, session, and application, reducing development time.
- It has built-in mechanisms for exception and error management.
- It supports custom tags and tag libraries.

Steps to Create a JSP

1. Take any HTML file you have previously created.
2. Change the file extension from .html to .jsp.
3. Load the new .jsp file in a browser.

When we load a JSP file for the first time:

- JSP is converted into a Java file.
- Java file is compiled into a servlet.
- Compiled servlet is loaded and executed.

hello.jsp:

```
<!DOCTYPE html>
<html>
<body>
   Hello! The time is now <%= new java.util.Date() %>
</body>
</html>
```

Explanation:

- The <%= %> tags enclose a Java expression.
- new java.util.Date() expression retrieves the current date and time.
- When the JSP page is loaded in the browser, the Java expression is evaluated at runtime, and output is embedded into the HTML.
- Each time you reload the page, it displays the current time, demonstrating how JSP dynamically generates HTML content based on Java logic.

JSP Elements

In JSP elements can be divided into 4 different types.

- Expression
- Scriplets
- Directives
- Declarations

1. Expression

This tag is used to output any data on the generated page. These data are automatically converted to a string and printed on the output stream.

Syntax:

<%= "Anything" %>

Note: JSP Expressions start with Syntax of JSP Scriptles are with <%=and ends with %>. Between these, you can put anything that will convert to the String and that will be displayed.

Example:

<%="HelloWorld!" %>

2. Scriplets

This allows inserting any amount of valid Java code. These codes are placed in the _jspService() method by the JSP engine.

Syntax:

<%
// Java codes
%>

Note: JSP Scriptlets begins with <% and ends %> . We can embed any amount of Java code in the JSP Scriptlets. JSP Engine places these codes in the _jspService() method.

Example:

<%

String name = "Aakash";
out.println("Hello, " + name);
%>
Variables available to the JSP Scriptlets are:

- Request
- Response
- Session
- Out

3. Directives

A JSP directive starts with <%@ characters. In the directives, we can import packages, define error-handling pages, or configure session information for the JSP page.

Syntax:

<%@ directive attribute="value" %>

Types of Directives:

- page: It defines page settings.
- include: It includes other files.
- taglib: It declares a custom tag library.

4. Declarations

This is used for defining functions and variables to be used in the JSP.

Syntax:

<%!
//java codes
%>

Note: JSP Declaratives begins with <%! and ends %> with We can embed any amount of java code in the JSP Declaratives. Variables and functions defined in the declaratives are class-level and can be used anywhere on the JSP page.

Example:

```
<%@ page import="java.util.*" %>
<html>
<body>

<%! int max_marks = 100; %>

   Hello! The time is now <%= max_marks %>
</body>
</html>
```

Differences Between JSP and Servlets

Features	JSP	Servlet
Code Length	Jsp required less code	Servlets required more code
Ease of Use	Jsp is simple to use	Servlet is more complex to use
Dynamic Content	Easily embedded in HTML	Requires HTML generation in code
Page Maintenance	Easier to maintain	More challenging

Life Cycle of JSP

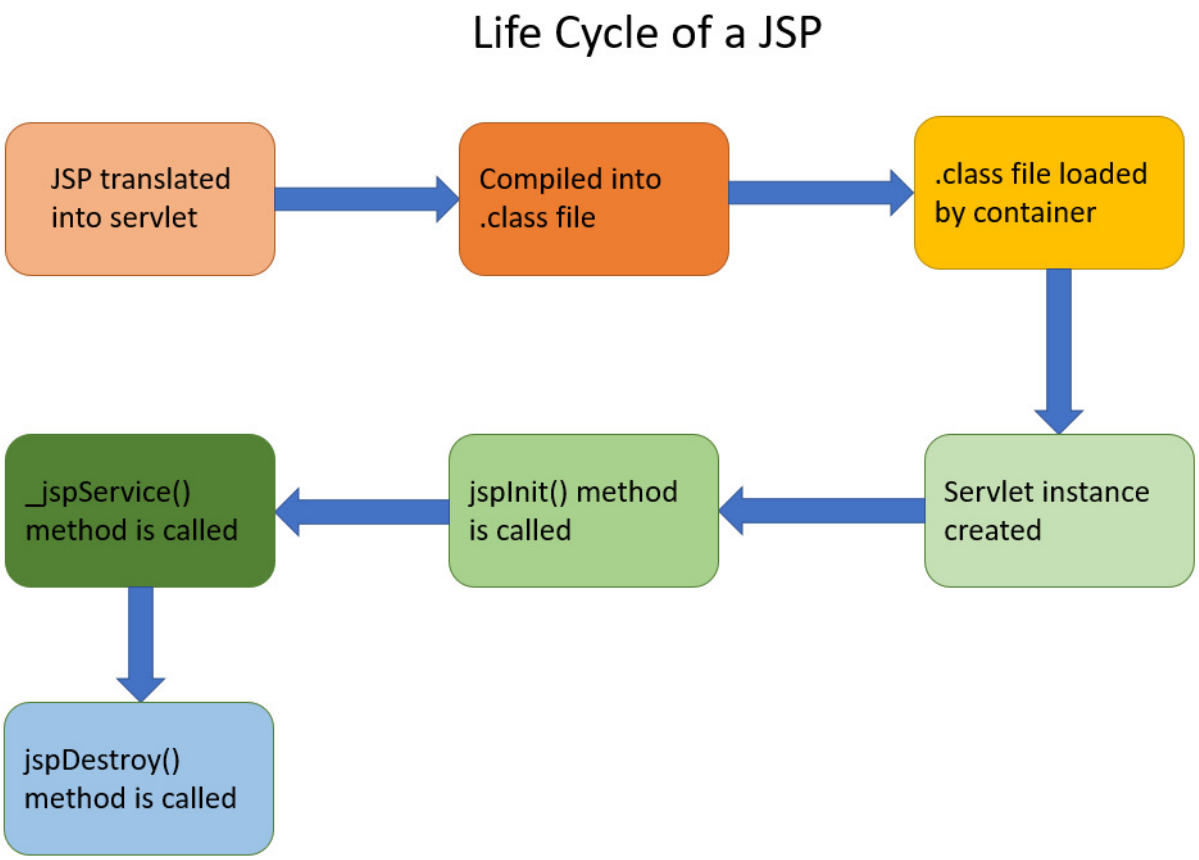
The life cycle of a JavaServer Page (JSP) consists of various phases that start from its creation, followed by its translation into a servlet, and finally managed by the servlet lifecycle. The JSP engine handles this process automatically.

The JSP (JavaServer Pages) lifecycle describes the stages a JSP page goes through from initial request to completion.

The life cycle of a jsp page comprises of following phases

1. Translation phase
2. Compilation phase
3. Initialization phase
4. Execution phase
5. Destruction phase

1. **Translation:** The JSP engine (web container) converts the JSP file into a servlet (a Java file).
2. **Compilation:** The generated servlet (.java file) is compiled into a .class file.
3. **Class Loading:** The servlet class is loaded into the container.
4. **Instantiation:** An instance of the servlet class is created.
5. **Initialization:** The jspInit() method is called, allowing for initialization tasks like setting up database connections or opening files. This is typically done only once.
6. **Request Processing:** The _jspService() method handles incoming requests, generating HTML output that is sent back to the client.
7. **Destruction:** The jspDestroy() method is called when the JSP is no longer needed, allowing for cleanup operations.



In JSP, implicit objects are predefined, readily available objects that you can use directly within your code without needing to declare or initialize them explicitly. They are essentially variables that the JSP engine provides to simplify web development tasks.

There are a total of nine implicit objects, as follws-

- **request**

  - Type: javax.servlet.http.HttpServletRequest
  - Description: Represents the HTTP request. It provides methods to get request parameters, attributes, headers, and other details about the client request.
  - Example:

    <%= request.getParameter("username") %>

response

  - Type: javax.servlet.http.HttpServletResponse
  - Description: Represents the HTTP response. It allows sending output to the client, setting headers, cookies, and managing redirects.

- Example:

```
response.setContentType("text/html");
```

out

- Type: javax.servlet.jsp.JspWriter
- Description: Used to send output to the client. It is an instance of JspWriter.
- Example:

```
out.println("Hello, World!");
```

session

- Type: javax.servlet.http.HttpSession
- Description: Represents the user's session. It allows you to store and retrieve data that persists across multiple page requests.
- Example:

```
session.setAttribute("user", "John");
```

application

- Type: javax.servlet.ServletContext
- Description: Represents the application-wide context and allows sharing data across the application.
- Example:

```
application.setAttribute("appName", "MyApp");
```

config

- Type: javax.servlet.ServletConfig
- Description: Represents the servlet configuration object. It allows you to access initialization parameters and other servlet configuration information
- Example:

```
String servletName = config.getServletName();
```

pageContext

- Type: javax.servlet.jsp.PageContext
- Description: Provides access to the page's environment, including other implicit objects, request, response, session, application, and attributes
- Example:

```
pageContext.setAttribute("key", "value");
```

page

- Type: java.lang.Object
- Description: Represents the JSP page itself. It provides access to the current page's information, such as its attributes.
- Example:

  ```
  <%= page.getClass().getName() %>
  ```

exception

- Type: java.lang.Throwable
- Description: Represents an exception that occurred on the page, if any.
- Example:

  ```
  <%= exception.getMessage() %>
  ```

Servlet Overview and Architecture, Interface Servlet and the Servlet Life Cycle, Handling HTTP get Requests, Handling HTTP post Requests, Redirecting Requests to Other Resources, Session Tracking, Cookies, Session Tracking with Http Session. Java Server Pages (JSP): Introduction, Java Server Pages Overview, A First Java Server Page Example, Implicit Objects, Scripting, Standard Actions, Directives, Custom Tag Libraries.