

## Chapter – 7

### Java Servlets

#### Introduction to Web

Web is basically a system of *Internet* servers that supports formatted documents. The documents are formatted using a markup language called HTML (*HyperText Markup Language*) that supports links to other documents, like graphics, audio, and video files.

Web consists of billions of clients and servers connected through wires and wireless networks. First, web clients make requests to a web server. Then, the web server receives the request, finds the resources and returns the response to the client. When a server answers a request, it usually sends some type of content to the client. Then, the client uses a web browser to send a request to the server. The server often sends a response back to the browser with a set of instructions written in HTML. All browsers know how to display HTML pages to the client.

Basically, this is all about the backend working of the WWW (World Wide Web). Now, let's understand the connectivity between Web & HTTP.

#### Web & HTTP

A website is a collection of static files i.e. web pages such as HTML pages, images, graphics etc. A *Web application* is a website with dynamic functionality on the server. **Google**, **Facebook**, **Twitter** are examples of web applications.

So, what is the link between the Web and HTTP? Let's now find out.

#### HTTP (Hypertext Transfer Protocol)

- HTTP is a protocol that clients and servers on the web to communicate.
- It is similar to other internet protocols such as **SMTP** (Simple Mail Transfer Protocol) and **FTP**(File Transfer Protocol).
- HTTP is a *stateless protocol* i.e it supports only one request per connection. This means that with HTTP the clients connect to the server to send one request and then disconnect. This mechanism allows more users to connect to a given server over a period of time.
- The client sends an HTTP request and the server answers with an HTML page to the client, using HTTP.

The HTTP request can be made using a variety of methods, but the ones which we use widely are **Get** and **Post**. The method name itself tells the server the kind of request that is being made, and how the rest of the message will be formatted.

Now, with the help of the below table, let's understand the difference between Get and Post methods of HTTP.

Get	Post
1. Data is sent in the header body	1. Data is sent in the request body
2. Restricted to limited data transfer	2. Supports a large amount of data transfer
3. It is not secured	3. It is completely secured
4. It can be bookmarked	4. It cannot be bookmarked

Now, that you have learned a few basics of web, let's jump to the core topic and understand the concept of a servlet.

### What are Servlets?

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

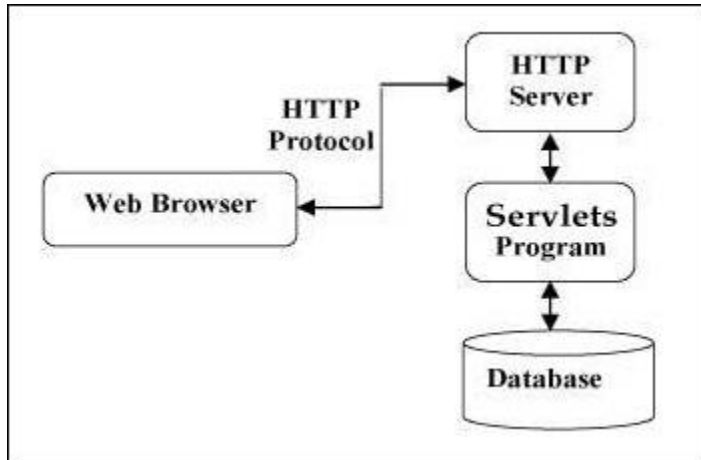
Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.

- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

### Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.



### Servlets Tasks

Servlets perform the following major tasks –

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

## Servlets Packages

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the **javax.servlet** and **javax.servlet.http** packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

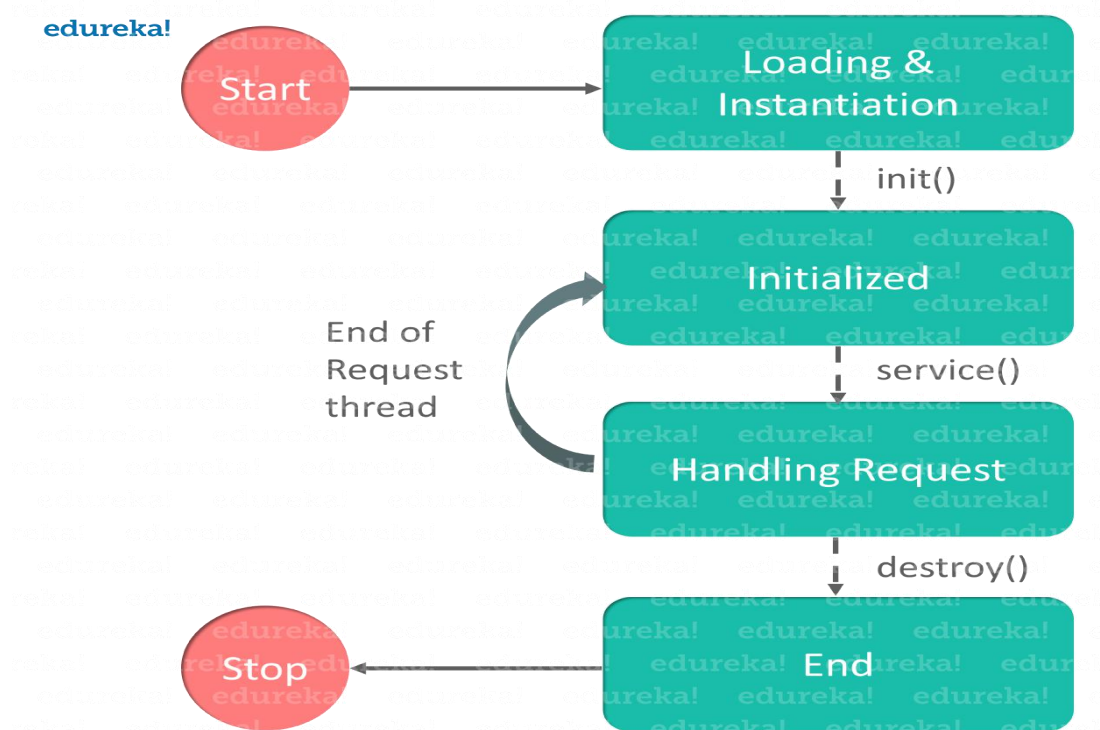
These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

## Servlet Life Cycle

The Servlet life cycle mainly includes the following four stages,

- Loading a Servlet
- Initializing the Servlet
- Request handling
- Destroying the Servlet



1. When the web server (e.g. Apache Tomcat) starts up, the servlet container deploy and loads all the servlets.
2. The servlet is initialized by calling the `init()` method. The `Servlet.init()` method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into service.
3. The servlet then calls `service()` method to process a client's request. This method is invoked to inform the Servlet about the client requests.
4. The servlet is terminated by calling the `destroy()`.
5. The `destroy()` method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance.

`init()` and `destroy()` methods are called only once. Finally, a servlet is garbage collected by the garbage collector of the JVM. So this concludes the life cycle of a servlet. Now, let me guide you through the steps of creating java servlets.

### **Java Servlets: Steps to Create Servlet**

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Add mappings to the web.xml file
5. Start the server and deploy the project
6. Access the servlet

Now, based on the above steps, let's write a program and understand how a servlet works.

To run a servlet program, we should have Apache Tomcat Server installed and configured. *Eclipse for Java EE provides in-built Apache Tomcat*. Once the server is configured, you can start with your program. One important point to note – for any servlet program, you need 3 files – *index.html file, Java class file, and web.xml file*. The very first step is to create a Dynamic Web Project and then proceed further.

Now, let's see how to add 2 numbers using servlets and display the output in the browser.

**First, I will write index.html file**

```
    <html>
    <!DOCTYPE HTML>

    <html>
    <body>

    <form action = "add" method="GET">
    Enter 1st number: <input type="text" name ="num1">
    Enter 2nd number: <input type="text" name="num2">
    </form>

    </body>
    </html>
```

Above program creates a form to enter the numbers for the addition operation. Without the Java class file, you can't perform addition on 2 numbers. So let's now create a class file.

```
package edureka;
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 Add extends HttpServlet{
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws
    IOException
    {
        int i = Integer.parseInt(req.getParameter("num1"));
        int j = Integer.parseInt(req.getParameter("num2"));
        int k= i+j;
        PrintWriter out = res.getWriter();
        out.println("Result is"+k);
    }
}
```

After writing the Java class file, the last step is to add mappings to the web.xml file. Let's see how to do that.

The *web.xml* file will be present in the WEB-INF folder of your web content. If it is not present, then you can click on Deployment Descriptor and click on *Generate Deployment Descriptor Stub*. Once you get your web.xml file ready, you need to add the mappings to it. Let's see how mapping is done using the below example:

```
<?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" version="3.0">
<display-name>Basic</display-name>
<servlet>
<servlet-name>Addition</servlet-name>
<servlet-class>edureka.Add</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Addition</servlet-name>
<url-pattern>/add</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

Once done, you can execute the program by starting the server and get the desired output on the browser.

## Java Servlets: Servlet Classes & Interfaces

Servlet API consists of two important packages that encapsulate all the important classes and interfaces, namely:

- `javax.servlet`
- `javax.servlet.http`

With the help of below table let's see some important *classes and Interfaces* of a servlet.

<i><b>Servlet</b></i>	Declares LifeCycle methods of the servlet
<i><b>ServletConfig</b></i>	Allows the servlet to get initialization methods
<i><b>ServletContext</b></i>	Enables the servlet to log access and access information
<i><b>ServletRequest</b></i>	Used to read data from the client request
<i><b>ServletResponse</b></i>	Used to write data to clients response
<i><b>GenericServlet</b></i>	Implements Servlet and Servlet.Config Interface
<i><b>ServletInputStream</b></i>	Provides input stream to read requests from the client
<i><b>ServletOutputStream</b></i>	Provides output stream to write responses to the client
<i><b>ServletException</b></i>	Indicates that the servlet is not available
<i><b>HttpServlet</b></i>	Provides methods to handle HTTP Request & Response