# Unit-IV

# Servlet

Servlets are modules of Java code that run in a server application (hence the name "Servlets", similar to "Applets" on the client side) to answer client requests. Servlets are not tied to a specific client-server protocol but they are most commonly used with HTTP and the word "Servlet" is often used in the meaning of "HTTP Servlet".

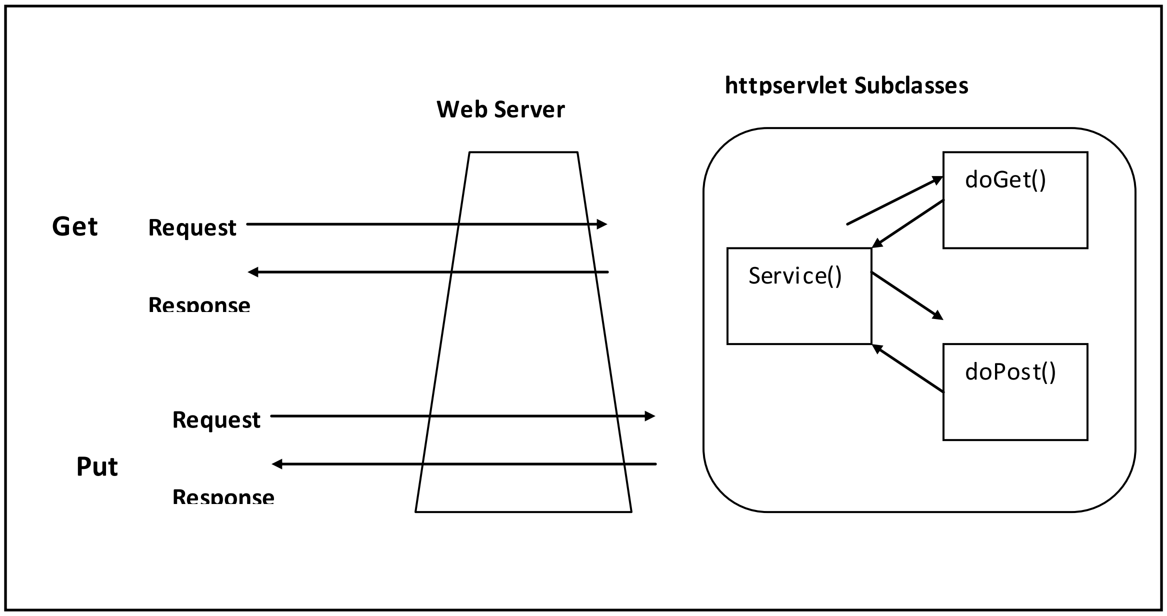
Servlets make use of the Java standard extension classes in the packages `javax.servlet` (the basic Servlet framework) and `javax.servlet.http` (extensions of the Servlet framework for Servlets that answer HTTP requests). Since Servlets are written in the highly portable Java language and follow a standard framework, they provide a means to create sophisticated server extensions in a server and operating system independent way.

Typical uses for HTTP Servlets include:

- Processing and/or storing data submitted by an HTML form.
- Providing dynamic content, e.g. returning the results of a database query to the client.
- Managing state information on top of the stateless HTTP, e.g. for an online shopping cart system which manages shopping carts for many concurrent customers and maps every request to the right customer.

## HTTP Servlet Basics

Note that although there are two types of servlets (GenericServlet and HttpServlet), we will discuss here only HttpServlet since GenericServlet can be used for advanced needs. The important thing here is that you should use GenericServlet if you want your servlet to response in another protocol rather than HTTP.

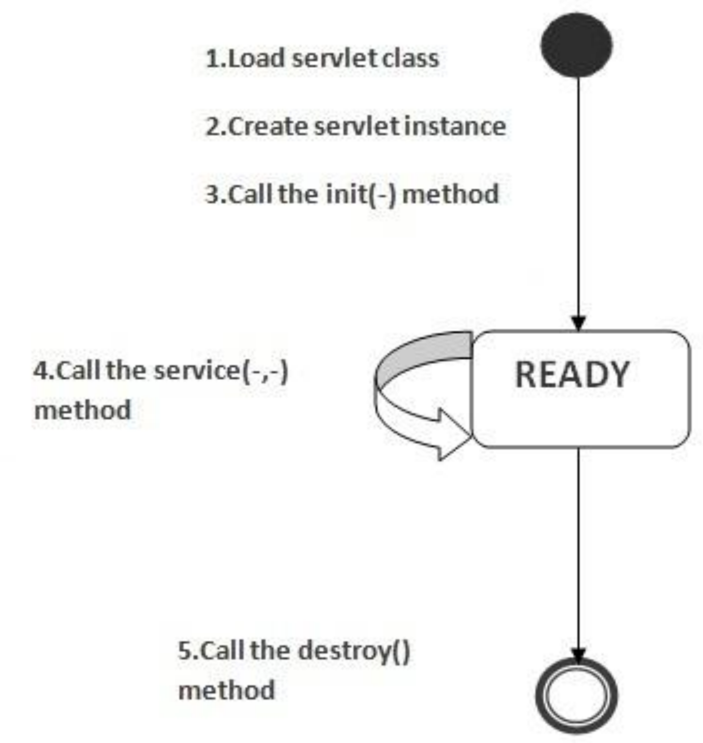


**Request and Response Model of HttpServlets**

## Servlet Life Cycle

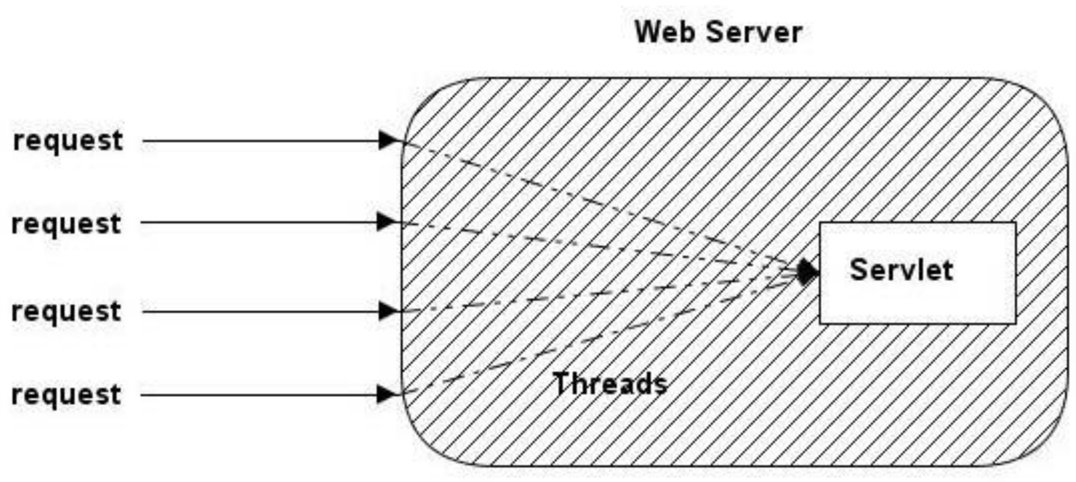
The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.


1.Load servlet class
2.Create servlet instance
3.Call the init(-) method
4.Call the service(-,-) method
READY
5.Call the destroy() method

As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

Servlet container creates only one instance of each servlet but the request of each user is handled with a separate thread. Each thread then calls doGet or doPost methods. This idea is shown in diagram



**A way of handling Servlet requests**

The init method of the servlet is called when the servlet is first created and each time the server receives a request for a servlet, the server spawns a new thread calls service method.

The service method checks the HTTP request type (GET, SET, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. method as appropriate.

Finally, the server may decide to remove a previous loaded servlets. In this case, the server calls destroy method of the servlet.

**An Sample Servlet Code**

To demonstrate the servlet life cycle, we'll begin with a simple example. Example shows a servlet that counts and displays the number of times it has been accessed. For simplicity's sake, it outputs plain text.

**Example A simple counter**

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

public class SimpleCounter extends HttpServlet {

  int count = 0;

  public void doGet(HttpServletRequest req, HttpServletResponse res)
                             throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    count++;
    out.println("Since loading, this servlet has been accessed " +
                   count + " times.");
  }
}
```

The code is simple--it just prints and increments the instance variable named `count`--but it shows the power of persistence. When the server loads this servlet, the server creates a single instance to handle every request made of the servlet. That's why this code can be so simple. The same instance variables exist between invocations and for all invocations.

### An example of GenericServlet

```
Public java.io*;

Import javax.servlet.*;



Public class simplservletdemo extends GenricServlet

{

public String getServletInfo()

{

return “Localhost”;

}

public void service(ServiceRequest  req,ServletResponse res)

{

try

{

PrintStream out=new PrintStream(res.getOutputStream());

Out.println(“Hello” + req.getRemoteHost());

Out.println9”this is the environment where the servlet is running…….”);

Out.Println(“the name of the server running is “+getServletInfo());

}

catch(Exception e)

{}
}

}
```

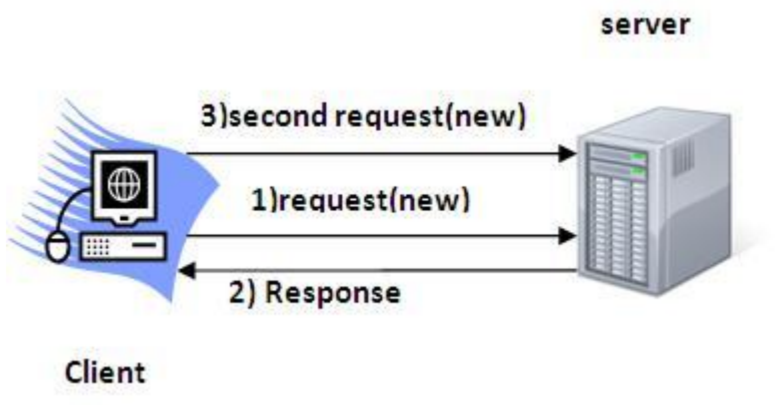
## Session Tracking in Servlets

**Session** simply means a particular interval of time.

**Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



### Why use Session Tracking?

**To recognize the user** It is used to recognize the particular user.

### Session Tracking Techniques

There are four techniques used in Session tracking:

1. **Cookies**
2. **Hidden Form Field**
3. **URL Rewriting**
4. **HttpSession**

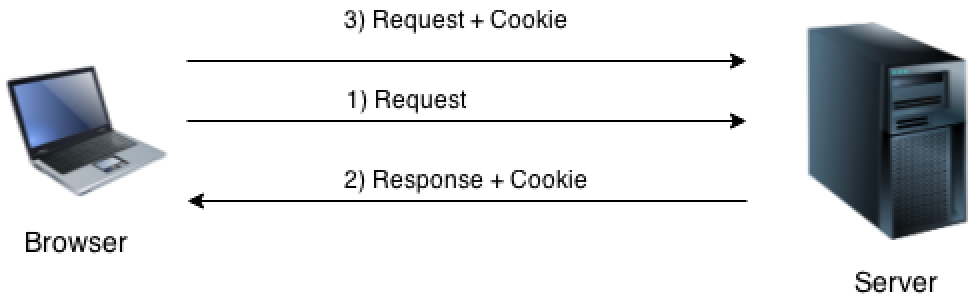
#### 1. COOKIES

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

**How Cookie works**

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



## 2. Hidden Form Field

In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.

In such case, we store the information in the hidden field and get it from another servlet. This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Let's see the code to store value in hidden field.

```
<input type="hidden" name="uname" value="Vimal Jaiswal">
```

Here, uname is the hidden field name and Vimal Jaiswal is the hidden field value.

**Real application of hidden form field**

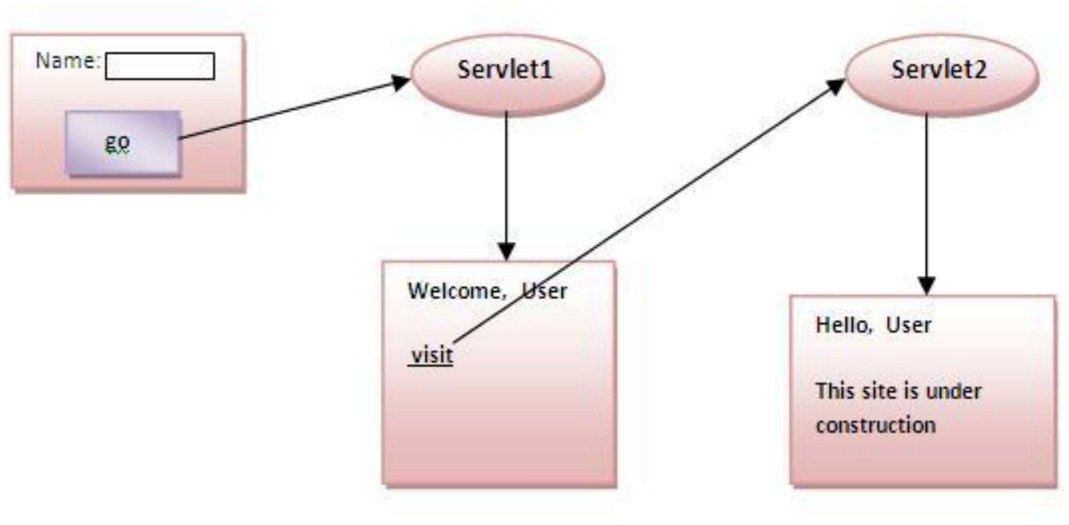
It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

## 2. URL Rewriting

In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource. We can send parameter name/value pairs using the following format:

url?name1=value1&name2=value2&??

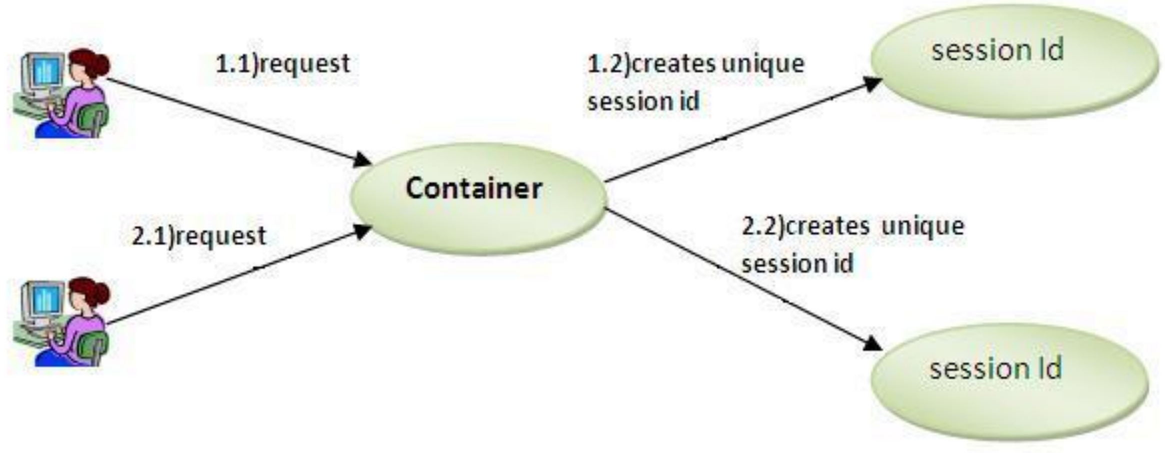
A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&). When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.



### 3. HttpSession interface

In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



The HttpServletRequest interface provides two methods to get the object of HttpSession:

1. **public HttpSession getSession():** Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **public HttpSession getSession(boolean create):** Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.