6.1 Introduction to Servlet

Servlets are pieces of Java source code that add functionality to a web server in order to provide proper dynamic response to the client according to the requests. The servlet clearly apply the request – response computing model i.e. when a request is arrived to servlet, it responds accordingly.

To work with Servlets, you need Java Servlet Development Kit (JSDK). The JSDK provides Servlet API to create Servlet for responding to client requests. The Servlet can do many tasks, like processing HTML forms or managing middle-tier processing to connect to existing data sources, connecting through a corporate firewall and many more. In addition to that, Servlet can maintain the database sessions, create and read cookies, managing resources, handle JSP pages and JavaBeans. A Servlet performs all these tasks better than other dynamic content generating technologies, such as CGI, ISAPI, ASP, and PHP. But the Servlet stands above all these in many the ways.

6.1.1 Advantages of Servlet over various Technologies

(i) Comparatively better performance.

(ii) No specialized and trained programmers are needed to develop and manage.

(iii) Executes within the addressed space of a web server.

(iv) No separate processes are required to handle different client requests and connecting to database.

(v) Are platform independent.

(vi) Java security manager on the server protects the resources on a server machine.

(vii) Can communicate with Applets, different databases and other remote resources via RMI and Sockets.

(viii) Feasible in terms of processing, memory requirements, open – close database connections.

(ix) Easily supports and integrates other Java based technology.

6.2 Servlet : A Better Replacement of CGI Technology

We know that there are two types of web pages, static web page and dynamic web page. Let's have a b understanding of how web browsers and servers cooperate to provide content to a user.

Consider a request for a static web page, a user enters URL into the web browser and browser generates an H request to appropriate web server. The web server directs this request to a specific file. This file is returned in an H response to the browser. The HTTP header in the response indicates the type of the content using **Multipurpose Inter Mail Extension (MIME).**

For example : MIME type of an ordinary ASCII text is text/plain and MIME type of an HTML source code is text/html.

includes C, C++

Performance related issues in terms of CGI :

This CGI suffered serious performance related problems. Such as :

(i) Expensive in terms of processor and memory.

(ii) Expensive to open and close database connections.

(iii) Not platform independent.

(iv) Experienced programmers are needed to work with CGI.

Because of these problems, many new technologies were introduced and Servlet is one of them.

As compare to CGI the Servlets provides many benefits.

Benefits of Servlets :

(i) Comparatively better performance.

(ii) Executes within the addressed space of a web server.

(iii) No separate processes are required to handle different client requests and connecting to database.

(iv) Are platform independent.

(v) Java security manager on the server protects the resources on a server machine.

(vi) Can communicate with Applets, different databases and other remote resources via RMI and Socket programming.

6.3 Servlet Life-Cycle

When a client makes a request, the server selects and loads appropriate Java class. This Java class generates contents and responds back to client. In most cases this client is web browser, Java server is **web server** and Java class is **Servlet**. To discuss this tactic lets study the life cycle of a Servlet. Three methods are central to the Servlet lifecycle. They are **init( )**, **service( )** and **destroy( )**.

They are implemented by every Servlet and invoked at specific time by the web server. To understand the working of life cycle of a Servlet, let's consider a standard situation where a client making a URL request and gets the respond from the web server.

From Fig. 6.3.1 we can see that different clients send requests to the web server; and by using multithreaded environment the each client's requests are forwarded to respective Servlets with all HTTP request data. Now let us study the working and lifecycle of Servlet in five easy steps.

Step First :

Assume that a user enters URL to a web browser. The web browser then generates an HTTP request for this URL. This request is then sent to the appropriate server.

Step Second :

This HTTP request is received by the web browser. The web server directs this request to a particular Servlet. The Servlet is then dynamically retrieved and loaded into the address space of the web server.

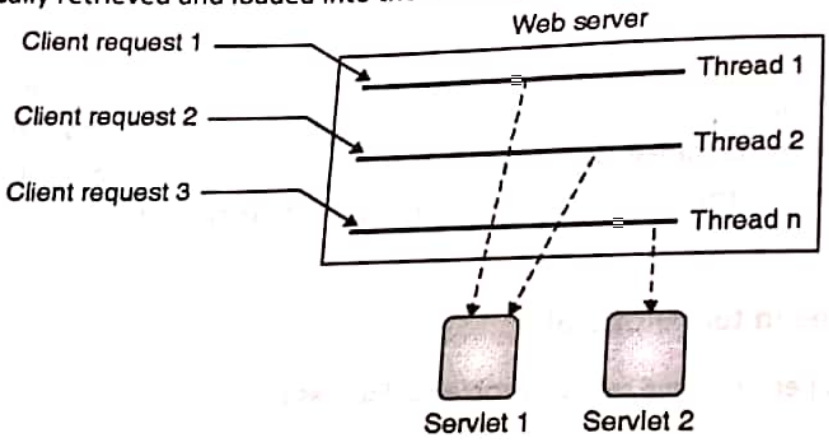

Fig. 6.3.1 : Web server forwards client's requests to appropriate servlet

Step Third :

To initialize the Servlet, the web server invokes the init( ) method of Servlet. This method is invoked when a Servlet is loaded for the first time in web server's memory. The **init( )** method is defined as:

```
protected void init(ServletConfig config)
```

```
protected void init(ServletConfig config)
```

- Here, the *ServletConfig* is the interface containing initial configuration setting for requested Servlet. The client browser can pass initialization parameters to this instance.

Step Fourth :

- After loading the Servlet and initializing the Servlet parameters the web browser invokes **service( )** method. This method actually process on HTTP request. This method is capable to read client's HTTP request data using *HttpServletRequest* instance and also to generate the response data using *HttpServletResponse* instance. This generated response is sent dynamically to client browser. The **service( )** method is defined as:

```
protected void service (HttpServletRequest request, HttpServletResponse response) throws ServletException,
IOException)
```

- After responding, the Servlet remains loaded in the Servlet's address space and remain available to process on any other HTTP request of the client.

Step Fifth :

- When web server feels that the Servlet is no longer useful, the web server decides to unload the Servlet from web server's address space. The web server executes Servlet's **destroy( )** method. The **destroy( )** method is defined as:

```
protected void destroy( )
```

- The execution is **destroy( )** method is to release the resources that are allocated by Servlet. Important data of Servlet and client may be saved to a permanent storage.

6.4 Creating Simple Servlet

Mostly the developers uses a Java IDE (such as: NetBeans IDE, Eclipse, BlueJ, TextPad, Notepad++, etc.) to develop enterprise application for Servlet. However, it is also possible to define it manually in a standard text editor (such as Notepad, Wordpad, etc.). This is done by manually defining a user-defined class and extending it with GenericServlet

6.5 Servlet API

The Servlet API contains two important java packages. They are :

(i) javax.servlet

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts bet a servlet class and the runtime environment provided for an instance of such a class by a conforming se container.

(ii) javax.servlet.http

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contra between a servlet class running under the HTTP protocol and the runtime environment provided for an instanc such a class by a conforming servlet container.

Section 6.6 and 6.7 explains both these packages in depth with list of all the interfaces, classes and except included in them.

6.7 The javax.servlet.http Package

The javax.servlet.http package contains a number of classes and interfaces that describe and define the cont between a Servlet class running under the HTTP protocol and the runtime environment provided for an instance of su class by a conforming Servlet container. Following Tables 6.7.1 and Table 6.7.2 enlists the interfaces and classes includ javax.servlet.http package.

Table 6.7.1 : Interface of javax.servlet.http Package

Interface	Description
HttpServletRequest	Extends the ServletRequest interface to provide request information for HTTP servlets.
HttpServletResponse	Extends the ServletResponse interface to provide HTTP-specific functionality in sending a response.
HttpSession	Provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.
HttpSessionActivationListener	Objects that are bound to a session may listen to container events notifying them that sessions will be passivated and that session will be activated.
HttpSessionAttributeListener	This listener interface can be implemented in order to get notifications of changes to the attribute lists of sessions within this web application.
HttpSessionBindingListener	Causes an object to be notified when it is bound to or unbound from a session.
HttpSessionListener	Implementations of this interface are notified of changes to the list of active sessions in a web application.

application.

Table 6.7.2 : Classes included in javax.servlet.http

Classes	Description
Cookie	Creates a cookie, a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server.
HttpServlet	Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.
HttpServletRequestWrapper	Provides a convenient implementation of the HttpServletRequest interface that can be subclassed by developers wishing to adapt the request to a Servlet.
HttpServletResponseWrapper	Provides a convenient implementation of the HttpServletResponse interface that can be subclassed by developers wishing to adapt the response from a Servlet.
HttpSessionBindingEvent	Events of this type are either sent to an object that implements HttpSessionBindingListener when it is bound or unbound from a session, or to aHttpSessionAttributeListener that has been configured in the deployment descriptor when any attribute is bound, unbound or replaced in a session.
HttpSessionEvent	This is the class representing event notifications for changes to sessions within a web application.

Explaining each of these interfaces and classes will go beyond the scope of this chapter. Hence following are brief

6.7.1 Interface HttpServletRequest

The interface HttpServletRequest enables an HTTP Servlet to obtain information about a client request. This interface is defined as :

```
public interface HttpServletRequest extends ServletRequest.
```

This interface extends the ServletRequest interface to provide request information for Servlets. The Table 6.7.3 shows commonly used methods of this interface.

Table 6.7.3 : Methods of HttpServletRequest Interface

Interface	Description
String getAuthType( )	Returns the name of the authentication scheme used to protect the servlet.
String getContextPath( )	Returns the portion of the request URI that indicates the context of the request.

	Description
boolean isRequestedSessionIdFromCookie( )	Checks whether the requested session ID came in as a cookie.
boolean isRequestedSessionIdFromURL( )	Checks whether the requested session ID came in as part of the request URL.
boolean isRequestedSessionIdValid( )	Checks whether the requested session ID is still valid.
void login(String username, String password)	Validate the provided username and password in the password validation realm used by the web container login mechanism configured for the ServletContext.
void logout( )	Establish null as the value returned when getUserPrincipal, getRemoteUser, and getAuthType is called on the request.

6.7.2 Interface HttpServletResponse

The interface *HttpServletResponse* enables a Servlet to formulate an HTTP response to a client. This interface is defined as:

```
public interface HttpServletResponse extends ServletResponse
```

This interface extends the ServletResponse interface to provide HTTP-specific functionality in sending a response. For example, it has methods to access HTTP headers and cookies. The Table 6.7.4 contains the commonly used methods of this interface.

sending sensitive data such as password and banking details. Servlet handles this type of requests using doPost( ) way of method.

6.8.1 Difference between doGet() and doPost() Methods

To know more about doGet( ) and doPost( ) methods, lets study the difference between them.

Sr. No.	doGet( ) method	doPost( ) method
1.	In doGet( ) method, the parameters are appended to the URL and sent with request information.	In doPost( ) method, the parameters are sent separately to the web server through a socket.
2.	The maximum amount of information that can be sent using doGet( ) method is 1024 characters.	We can send as much information as we want. There is no such restriction. Additionally we can send files and even binary data through doPost( ) method
3.	Parameters are not encrypted.	Parameters are encrypted.
4.	Comparatively faster than doPost( ) method.	Comparatively slower than doPost( ) method.
5.	doGet( ) is allowed to be repeated safely any number of times.	doPost( ) contains sensitive data, hence not allowed to repeat multiple times.

Program 6.8.1 : Write a program to demonstrating the use of doGet( ) method.

File-1 : Index1.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
```

Program 6.8.1 : **Write a program to demonstrating the use of doGet( ) method.**

File-1 : index1.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
```

6.10 Session and Session Tracking

Usually, a web browser generates a request. This request is forwarded to the web server through the internet. Th web server immediately responds to this request and then forgets about the browser and its request due to lake constant connection. This causes a problem for website attempting to do business on the internet.

This problem arises when a user wish to make multiple purchases from a website. The web server forgets the brows once and continuity in client's navigations will not be maintained. Hence, web server will find it impossible to create single bill for multiple purchases made through different requests by a client. The solution for this is to use session.

A session is a kind of virtual connection between client's computer and web server that stores some information. Th data is not accessible outside the web server i.e. a client cannot access the session information.

6.10.1 Life Cycle of HTTP Session

The following steps explains life cycle of HTTP Session.

1. A visitor makes a request for a resource, using a web browser, to web server.
2. The web server delivers the authentication module that result in the visitor's web browser. (such as, dis page)
3. The visitor enters valid username and password

Explanation :

Important point is, when POST method is used, the request parameters are not getting appended in URL. Because, the POST method will send the request parameters with-in the request header in encrypted form; whereas, the GET method appends the request parameters in URL.

6.9 Cookies

Cookies are text files that are created and sent by web servers on the client machine. These cookies can store any type of information for the session tracking purpose.

A cookie is a file containing two important parameters: name and value. But in fact, a cookie contains total 6 parameters. They are:

(i) The name of cookie

(ii) The value of cookie

(iii) The expiration date of cookie

iv) The path the cookie is valid for

v) The domain the cookie is valid for

i) The need for a secure connection to exists to use the cookie

Two of these parameters are mandatory [i.e. cookie name and its value]. All these information are stored as a text file, a special sub-directory on the hard disk of the client computer.

Now, the question arise is, **how cookie works?** Whenever a web browser makes an attempt to connect to a web ver, an executable built-in constructor will scan for the cookie related to that web server. If any cookie is found, then browser automatically sends that cookie to web server with every request and web server can read the information of cookie and then responds.

er2 : New Servlet.java

```
package p1;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class NewServlet extends HttpServlet
{
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response
    ServletException, IOException
    {
        PrintWriter out = response.getWriter( );
        String nm = request.getParameter("txtsname");
        String rl = request.getParameter("txtsroll");
        out.println("Student name is : " + nm);
        out.println("<br><br>");
        out.println("student Roll number is : " + rl);
    }
```

File-2 : New Servlet.java

```
package p1;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class NewServlet extends HttpServlet
{
    @Override
    protected void doPost(HttpServletRequest request, HttpServ
    ServletException, IOException
    {
        PrintWriter out = response.getWriter( );
        String nm = request.getParameter("txtsname");
```

```
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Demonstrating GET method</title>
    </head>
<body>
    <form action="NewServlet" method="GET">
    Enter Student name <input type="text" name="txtsname" /> <br> <br>
    Enter Roll number <input type="text" name="txtsroll" /> <br> <br>
    <input type="submit" value="Send Data" />
    </form>
</body>
</html>
```

File-2 : New Servlet.java

```
package p1;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```

```
    <input type="submit" value="Send Data" />
    </form>
  </body>
</html>
```

File-2 : New Servlet.java

```
package p1;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;
```