Unit-3
SERVER SIDE PROGRAMMING

SERVLETS

Servlets are effective for developing Web-based solutions

CGI (Common Gateway Interface)

The common gateway interface (CGI) is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user. CGI does not communicate directly with the browser.

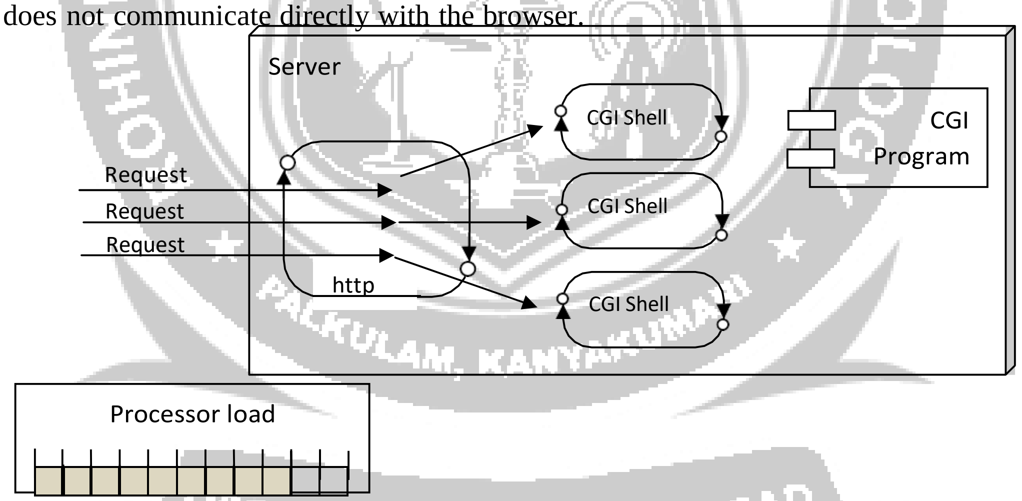


Figure 3.1 CGI

Advantages of Servlets over CGI

- Better performance: because it creates a thread for each request not process.
- Portability: because it uses java language.
- Robust: Servlets are managed by JVM so no need to worry about memory leak, garbage collection etc.
- Secure: because it uses java language.

Capabilities of Servlet

Servlet is a web component that is deployed on the server to create dynamic web page.

Differences between process and threads

Process	**Threads**
Process run in separate address space	Threads of a process share address space.
Process have their own copy of data segment of parent process.	Threads have direct access to data segment of its process
Processes must use inter-process communication to communicate with sibling processes	Threads can directly communicate with other threads of that process.
Process carry more state information	Threads carry less state information
New process require duplication of parent process, and allocation of memory and resources for it are costly	New threads are easily created.

Servlet Architecture

The server that executes a servlet is called as the **servlet container or servlet engine.**

A client sends an HTTP request to the server or servlet container.

The server or servlet container receives the request and directs it to be processed by the appropriate servlet.

The servlet does its processing, which may be interacting with a database or other server-side components.

The servlet returns its results to the client.

From the programming perspective, all servlets must implement the Servlet interface of the package javax.servlet.

The methods of interface Servlet are invoked automatically by the servlet container.

This interface defines the following five methods:

void init(ServletConfig config)

ServletConfig getServletConfig()

String getServletInfo()

void service(ServletRequest request, ServletResponse response)

void destroy()

1. **void init(ServletConfig config)**

   The servlet container calls the init method only once after creating the servlet instance.

   The init method is used to initialize the servlet.

   The ServletConfig argument is supplied by the servlet container that executes the servlet.

2. **ServletConfig getServletConfig()**

   This method returns a reference to an object that implements interface ServletConfig.

   This object provides access to the servlet's configuration information such as servlet initialization parameters and the servlet's ServletContext, which provides the servlet with access to its environment

3. **String getServletInfo()**

   This method is defined by a servlet programmer to return a String containing servlet information such as the servlet's author and version.

4. **void service(ServletRequest request, ServletResponse response)**

   The servlet container calls this method to respond to a client request to the servlet.

5. **void destroy()**

   This "cleanup" method is called by the web container before removing the servlet instance from the service.

   Resources used by the servlet, such as an open file or an open database connection, should be deallocated here.

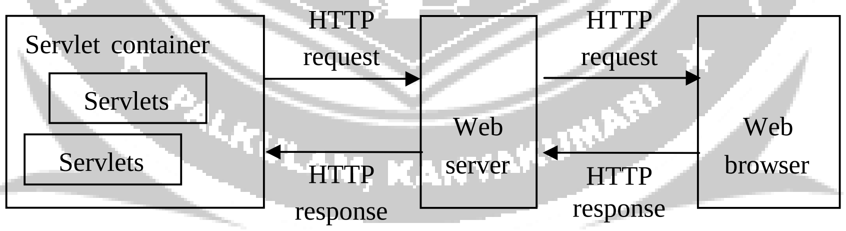


Figure: 3.3 Servlet Architecture

Servlet lifecycle

The web container maintains the life cycle of a servlet instance. The following are the phases in the life of a servlet:

– Servlet class is loaded.

– Servlet instance is created.

– init method is invoked.

– service method is invoked.

– destroy method is invoked

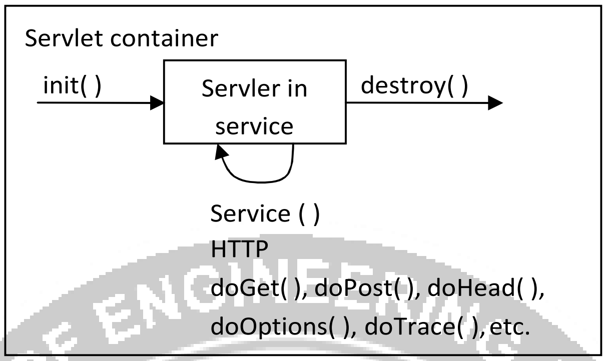


Figure 3.4 Servlet lifecycle

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.

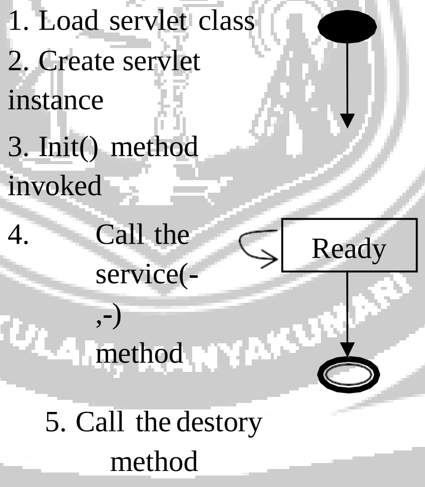

1. Load servlet class
2. Create servlet instance
3. Init() method invoked
4. Call the service(-,-) method
Ready
5. Call the destory method

Figure 3.5 State diagram of lifecycle of servlet

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) init method is invoked

The web container calls the init method only once after creating the servlet instance.The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface.

public void init(ServletConfig config) throws ServletException

4) service method is invoked

The web container calls the service method each time when request for the servlet is received.

If servlet is not initialized, it follows the first three steps as described above then calls the service method.

If servlet is initialized, it calls the service method. Remember that servlet is initialized only once.

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 spawns 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.The doGet() and doPost() are most frequently used methods with in each service request.

Syntax:public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

public void destroy()

The servlet can be created by three ways:

1. By implementing Servlet interface
2. By inheriting GenericServlet class
3. By inheriting HttpServlet class

Servlet Interface

Servlet interface provides common behaviour to all the servlets. Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods (init, service and destroy) that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods in servlet interface

Method	Description
public void init(ServletConfig config)	Initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponseresponse)	This provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	It is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	Returns the object of ServletConfig.
public String getServletInfo()	Returns information about servlet such as writer, copyright, version etc.

Creating a servlet using servlet interface

```
 import java.io.*;
 import
 javax.servlet.*;
public class First implements Servlet
      ServletConfig config=null;
       public void init(ServletConfig config)
       {              this config=config;                    }
         public void service(ServletRequest req,ServletResponse res)
                              throws IOException,ServletException
         {   res.setContentType("text/html");
         PrintWriter out=res.getWriter();
         out.print("<html><body>");
         out.print("<b>This is a simple servlet</b>");
         out.print("</body></html>");           }
        public void destroy()
        {            System.out.println("servlet is destroyed");
                     }public ServletConfig getServletConfig()
       {return config;        }
       public String getServletInfo()
```

The above program creates a servlet by implementing the servlet interface.

The servlet is initialized with its configuration in the init().

The service() does the real operation of the servlet.

In this example, the servlet creates a web page that displays "This is a simple servlet".

The service method creates two objects in its parameter field: req and res.

The req object is created for ServletRequest class. All the requests will be issues by this object.

The res object is created for ServletResponseclass. This is responsible for the output of the servlet.

res.ContentType() describes the content of the output.

PrintWriter is the output stream with object out.

The print() is called using the object of the PrintWriter class.

The print() contains the HTML tags to create the web page.

The servlet is destroyed using destroy(). In this example, getServletInfo () is optional and returns the copyrights information.

ServletConfig Interface

An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file.

public ServletConfig getServletConfig(); //This returns the object of the ServletConfig.

Methods of ServletConfig interface

Method	Description
public String getInitParameter(String name)	Returns the parameter value for the specified parameter name.
public Enumeration getInitParameterNames()	Returns an enumeration of all the initialization parameter names.
public String getServletName()	Returns the name of the servlet.
public ServletContext getServletContext()	Returns an object of ServletContext.

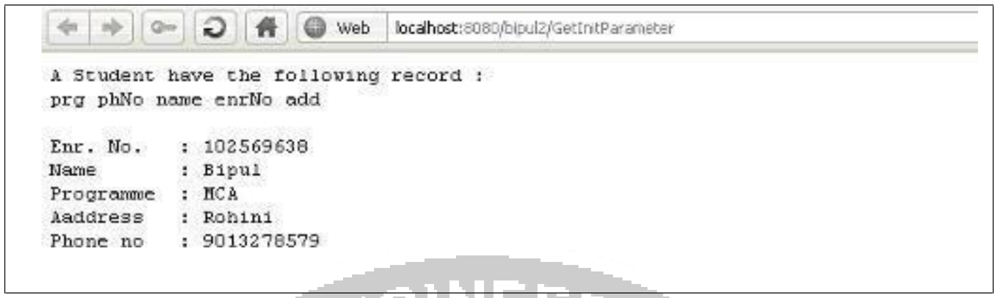
Servlet configuration

```
import java.io.*;import java.util.*;import javax.servlet.*;
import javax.servlet.http.*;
public class GetInitParameter extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
throws ServletException, IOException {
PrintWriter out = response.getWriter();
out.println("A Student have the following record : ");
Enumeration enm =
getServletConfig().getInitParameterNames();while
(enm.hasMoreElements()) {
out.print(enm.nextElement() + " ");}
out.println("\nEnr. No.:"+getServletConfig().getInitParameter("enrNo"));
out.println("Name : "+ getServletConfig().getInitParameter("name"));
out.println("Programme : " + getServletConfig().getInitParameter("prg"));
```

web.xml

```
<web-app><servlet><init-param>
<param-name>enrNo</param-name>
<param-value>102569638</param-value></init-param>
<init-param><param-name>name</param-name>
<param-value>Bipul</param-value></init-param>
<init-param><param-name>prg</param-name>
<param-value>MCA</param-value></init-param>
<init-param><param-name>add</param-name>
<param-value>Rohini</param-value></init-param>
<init-param><param-name>phNo</param-name>
<param-value>9013278579</param-value></init-param>
<servlet-name>GetInitParameter</servlet-name>
<servlet-class>GetInitParameter</servlet-class></servlet>
<servlet-mapping>
<servlet-name>GetInitParameter</servlet-name>
<url-pattern>/GetInitParameter</url-pattern>
```



```
A Student have the following record :
prg phNo name enrNo add

Enr. No.    : 102569638
Name        : Bipul
Programme   : MCA
Aaddress    : Rohini
Phone no    : 9013278579
```

GenericServlet class

GenericServlet class implements Servlet, ServletConfig and Serializable interfaces.

It provides the implementaion of all the methods of these interfaces except the service method.GenericServlet class can handle any type of request so it is protocol-independent.

Methods of GenericServlet class

Method	Description
public void init(ServletConfig config)	To initialize the servlet.
public abstract void service(ServletRequest request, ServletResponse response)	Provides service for the incoming request. It is invoked at each time when user requests for a servlet.
public void destroy()	Invoked only once throughout the life cycle and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	Returns the object of ServletConfig.
public String getServletInfo()	Returns information about servlet such as writer, copyright, version etc.
public void init()	It is a convenient method for the servlet programmers, now there is no need to call super.init(config)
public ServletContext getServletContext()	Returns the object of ServletContext.
public String getInitParameter(String name)	Returns the parameter value for the given parameter name.
public Enumeration getInitParameterNames()	Returns all the parameters defined in the web.xml file.
public String getServletName()	Returns the name of the servlet object.

Gereric servlet

```
import java.io.*;import javax.servlet.*;
public class First extends GenericServlet{
public void service(ServletRequest req,ServletResponse res) throws IOException,
        ServletException{
res.setContentType("text/html");
PrintWriter out=res.getWriter();out.print("<html><body>");
out.print("<b>hello generic servlet</b>");out.print("</body></html>");}}
```

HttpServlet Class

HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class

Method	Description
public void service(ServletRequest req,ServletResponse res)	dispatches the request to the protected service method by converting the request and response object into http type.
protected void service(HttpServletRequest req, HttpServletResponse res)	receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
protected void doGet(HttpServletRequest req, HttpServletResponse res)	handles the GET request. It is invoked by the web container.
protected void doPost(HttpServletRequest req, HttpServletResponse res)	handles the POST request. It is invoked by the web container.
protected void doHead(HttpServletRequest req, HttpServletResponse res)	handles the HEAD request. It is invoked by the web container.
protected void doOptions(HttpServletRequest req, HttpServletResponse res)	handles the OPTIONS request. It is invoked by the web container.
protected void doPut(HttpServletRequest req, HttpServletResponse res)	handles the PUT request. It is invoked by the web container.

protected void doTrace(HttpServletRequest req, HttpServletResponse res)	handles the TRACE request. It is invoked by the web container.
protected void doDelete(HttpServletRequest req, HttpServletResponse res)	handles the DELETE request. It is invoked by the web container.
protected long getLastModified(HttpServletRequest req)	returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

HTTP servlet

```
import java.io.*;import javax.servlet.*;import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
  private String message;
 public void init() throws ServletException
 {      message = "Hello World"; }
 publicvoiddoGet(HttpServletRequestrequest, HttpServletResponse response)
          throws ServletException, IOException
 {     response.setContentType("text/html");
     PrintWriter out = response.getWriter();
     out.println("<h1>" + message +"</h1>"); }
 public void destroy() { }}
```

ServletRequest Interface

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

Methods of ServletRequest interface

Method	Description
public String getParameter(String name)	Usedto obtain the value of a parameter by name.
public String[] getParameterValues(String name)	Returnsan array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.

java.util.Enumeration getParameterNames()	Returnsan enumeration of all of the request parameter names.
public int getContentLength()	Returns the size of the request entity data, or
public String getCharacterEncoding()	Returns the character set encoding for the input of this request.
public String getContentType()	Returns the Internet Media Type of the request entity data, or null if not known.
public ServletInputStream getInputStream() throws IOException	Returns an input stream for reading binary data in the request body.
public String getParameter(String name)	Usedto obtain the value of a parameter by name.
public String[] getParameterValues(String name)	Returnsan array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box.
public abstract String getServerName()	Returns the host name of the server that received the request.
public int getServerPort()	Returns the port number on which this request was received.

ServletRequest to display the name of the user

index.html

```
<form action="welcome" method="get">
Enter your name<input type="text" name="name"><br>
<input type="submit" value="login"></form>
<form action="welcome" method="get">
Enter your name<input type="text" name="name"><br>
<input type="submit" value="login"></form>
```

DemoServ.java

```
import javax.servlet.http.*; import javax.servlet.*; import java.io.*;
public class DemoServ extends HttpServlet{
```

```
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{ res.setContentType("text/html"); PrintWriter pw=res.getWriter();
String name=req.getParameter("name");//will return value
pw.println("Welcome"+name); pw.close(); }}
```

FORM GET AND POST ACTIONS

The browser uses two methods to pass this information to web server. These methods are GET () and POST ().

GET(): The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the '?' character.

Example: http://www.test.com/hello?key1=value1&key2=value2

The GET method is the default method to pass information from browser to web server.GET method should be avoided while passing password or other sensitive information to the server.The GET method can hold only 1024 characters in a request string.This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using doGet() method.

Reading Form Data using Servlet

Servlets handles form data parsing automatically using the following methods depending on the situation:

1. getParameter()-to get the value of a form parameter.
2. getParameterValues()-this method is used if the parameter appears more than once and returns multiple values(example checkbox).
3. getParameterNames()-this method is used when a complete list of all parameters is needed in the current request.

Post()

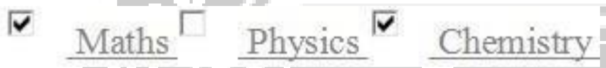
Post() is a reliable method of passing information to a backend program.This handles the information in exactly the same way as GET methods, but instead of sending it as a text string after a ?in the URL, it sends it as a separate message. This message comes to the backend program in the form of the standard input which can be parsed and processed. Servlet handles this type of requests using doPost() method.

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

```
public  class  ReadParams  extends HttpServlet {    // Method to handle GET method
request.
 publicvoiddoGet(HttpServletRequestrequest, HttpServletResponseresponse)
         throws ServletException, IOException
 {
    response.setContentType("text/html");      PrintWriter  out  = response.getWriter();
 String title = "Reading All Form Parameters";
      out.println(  "<html>\n"  +  "<head><title>"  +  title  +  "</title></head>\n"  +
     "<body bgcolor=\"#f0f0f0\">\n" + "<h1 align=\"center\">" + title + "</h1>\n"
+"<table      width=\"100%\"      border=\"1\"      align=\"center\">\n"      +      "<tr
bgcolor=\"#949494\">\n"   +"<th>Param   Name</th><th>Param   Value(s)</th>\n"+
"</tr>\n");
    Enumeration  paramNames  =  request.getParameterNames();
   while(paramNames.hasMoreElements())           {
      String paramName = (String)paramNames.nextElement();
      out.print("<tr><td>" +  paramName  +  "</td>\n<td>");
      String[]  paramValues=   request.getParameterValues(paramName);
      if (paramValues.length== 1)          {
        String  paramValue  =  paramValues[0];
        if  (paramValue.length() == 0)          out.println("<i>No Value</i>");
         else           out.println(paramValue);        }
     else {            // Read multiple valued data
          out.println("<ul>");
          for(int i=0; i < paramValues.length; i++) {
              out.println("<li>" +paramValues[i]);
          }           out.println("</ul>");        }     }
    out.println("</tr>\n</table>\n</body></html>");
 } // Method to handle POST method request.
 publicvoiddoPost(HttpServletRequestrequest, HttpServletResponse response)
    throws ServletException, IOException
{    doGet(request,  response);   }}
```

Form.html

```
<html><body>
<form action="ReadParams" method="POST" target="_blank">
<input type="checkbox" name="maths" checked="checked" /> Maths
<input type="checkbox" name="physics" /> Physics
<input type="checkbox" name="chemistry" checked="checked" /> Chem
<input type="submit" value="Select Subject" />
</form></body></html>
```



Servlet output:

Param Name	Param Value(s)
maths	on
Chemistry	on

In the above example, the subjects checked in form.html is retrieved in the servlet code using get() and post(). The getParameterValues() returns the values of more than one parameter and is a enumeration type. The hasMoreElements() checks whether there are more parameters to be passed and the nextElement() fetches the next parameter value.