SERVLETS

Any company want to develop the website that can be developed in two ways, they are static website and dynamic website.

1. A static website is one where there is no interaction from the end user. To develop static website, we can use the markup languages like HTML, DHMTL, XML, JavaScript etc.
2. A dynamic website is one in which there exist end user interaction. To develop dynamic websites, in industry we have lot of technologies such as CGI (Common Gateway Interface), java, dot net, etc.

In the recent years SUN micro systems has developed a technology called Servlets to develop the dynamic websites and also for developing distributed applications.

A distributed application is one which always runs in the context of browser or www. The result of distributed application is always sharable across the globe. To develop distributed application one must follow the following:

Client-Server architecture:

1. 2-tier architecture (Client program, Database program).
2. 3-tier or MVC (Model [Database] View [JSP] Controller [Servlets]) architecture (Client program, Server program, Database program).
3. n-tier architecture (Client program, Firewall program, Server program, Database program). To exchange the data between client and server we use a protocol caller http which is a part of TCP/IP.
4. A client is the program which always makes a request to get the service from server.
5. A server is the program which always receives the request, process the request and gives
6. response to 'n' number of clients concurrently

A server is the third-party software developed by third party vendors according to SUN micro systems specification. All servers in the industry are developed in java language only. The basic purpose of using server is that to get concurrent access to a server-side program.

According to industry scenario, we have two types of servers; they are web server and application server.

WEB SERVER	APPLICATION SERVER
1. A web server is one which always supports http protocol only. 2. Web server does not contain enough security to prevent unauthorized users. 3. Web server is not able to provide enough services to develop effective server-side program. 4. For examples Tomcat server, web logic server, etc.	1. Any protocol can be supported. 2. An application server always provides 100% security to the server side program. 3. An application server provides effective services to develop server-side program. 4. For examples web logic server, web sphere server, pramathi server, etc.

NOTE: Web logic server acts as both web server and as well as application server

In the initial days of server side programming there is a concept called CGI and this was implemented in the languages called C and PERL. Because of this approach CGI has the following disadvantages

1. Platform dependency.
2. Not enough security is provided.
3. Having lack of performance. Since, for each and every request a new and separate process is creating (for example, if we make hundreds of requests, in the server side hundreds of new and separate processes will be created)

To avoid the above problems SUN micro system has released a technology called **Servlets.**

A servlet is a simple platform independent, architectural neutral server independent java program which extends the functionality of either web server or application server by running in the context of www.

Advantages of SERVLETS over CGI:

1. Servlets are always platform independent.

2. Servlets provides 100% security.

3. Irrespective of number of requests, a single process will be created at server side. Hence, Servlets are known as single instance multiple thread technology.

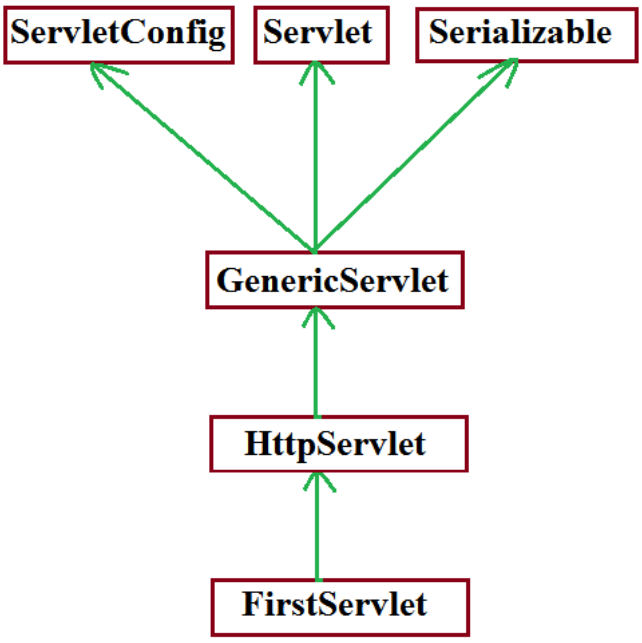
Servlets is the standard specification released by SUN micro systems and it is implemented by various server vendors such as BEA corporation (Web logic server), Apache Jakarta (Tomcat server).

In order to run any servlet one must have either application server or web server. In order to deal with servlet programming we must import the following packages:

javax.servlet.*;

javax.servlet.http.*;

Servlet Hierarchy:



LIFE CYCLE METHODS of servlets:

In servlets we have three life cycle methods, they are

- public void init ();
- public void service (ServletRequest req, ServletResponse res);
- public void destroy ();

public void init ():

1. Whenever client makes a request to a servlet, the server will receive the request and it automatically calls init () method i.e., init () method will be called only one time by the server whenever we make first request.
2. In this method, we write the block of statements which will perform one time operations, such as, opening the file, database connection, initialization of parameters, etc.

public void service (ServletRequest, ServletResponse):

After calling init () method, service () method will be called when we make first request from second request to further subsequent requests, server will call only service method. Therefore, service () method will be called each and every time as and when we make a request.

In service () method we write the block of statements which will perform repeated operations such as retrieving data from database, retrieving data from file, modifications of parameters data, etc. Hence, in service () method we always write business logic.

Whenever control comes to service () method the server will create two objects of **ServletRequest** and **ServletResponse** interfaces. Object of ServletRequest contains the data which is passed by client. After processing client data, the resultant data must be kept in an object of ServletResponse.

An object of ServletRequest and ServletResponse must be used always within the scope of service () method only i.e., we cannot use in init () method and destroy () method.

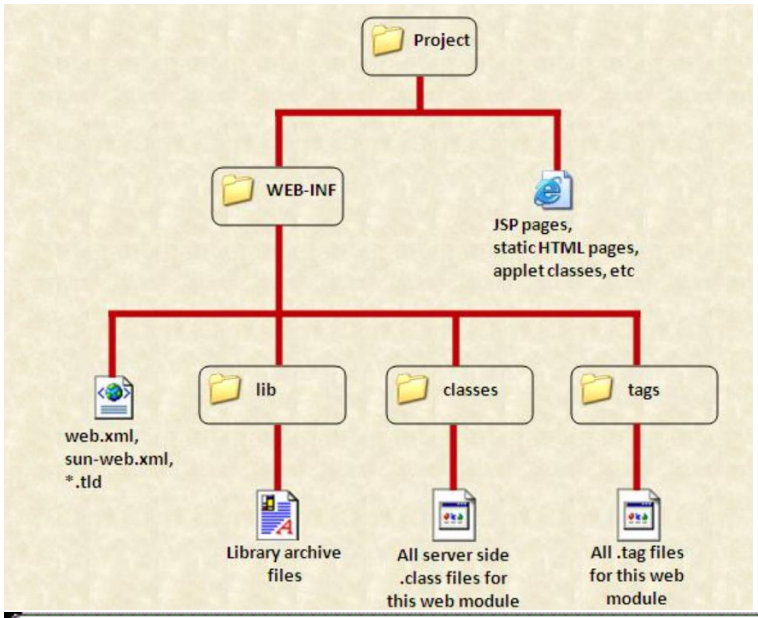
Once the service () method is completed an object of ServletRequest and an object of ServletResponse will be destroyed.

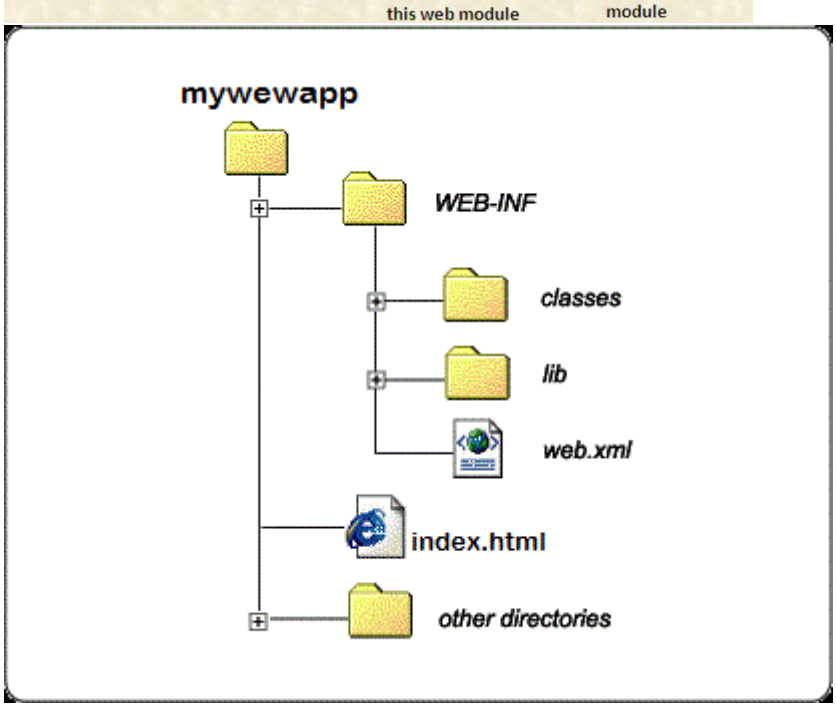
public void destroy ():

The destroy () method will be called by the server in two situations; they are when theserver is closed and when the servlet is removed from server context. In this method we write the block of statements which are obtained in init () method.

NOTE: Life cycle methods are those which will be called by the server at various times to perform various operations

Tomcat Directory Structure:





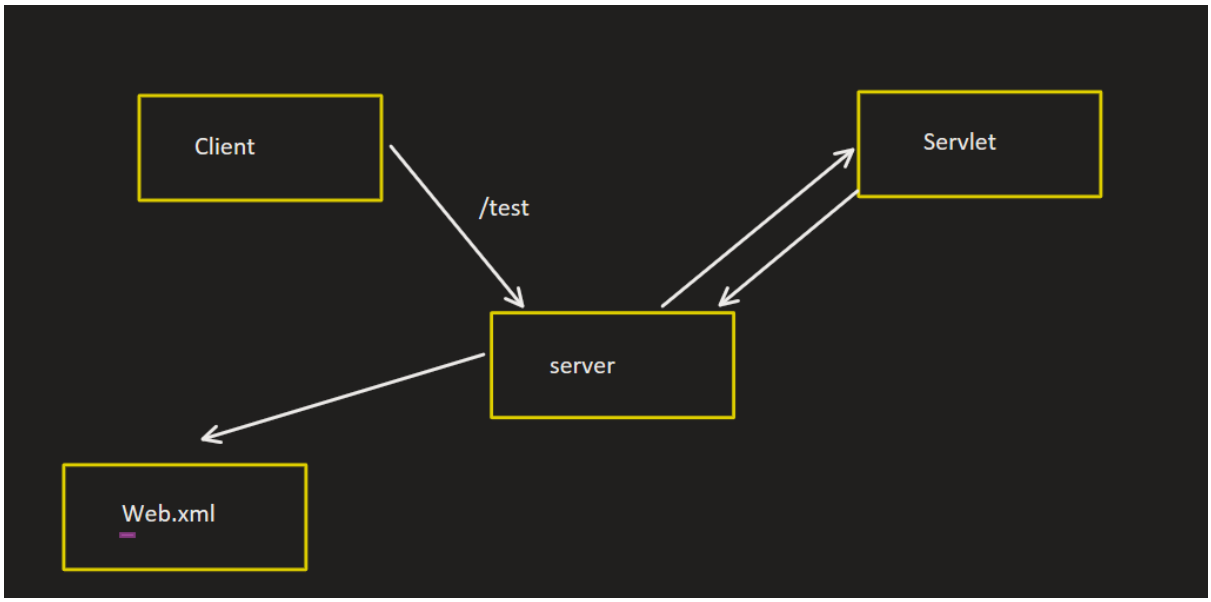
web.xml:

1. Whenever client makes a request to a servlet that request is received by server and server goes to a predefined file called web.xml for the details about a servlet.

2. web.xml file always gives the details of the servlets which are available in the server.

3. If the server is not able to find the requested servlet by the client then server generates an error (resource not found) [A resource is a program which resides in server].

4. If the requested servlet is available in web.xml then server will go to the servlet, executes the servlet and gives response back to client.




```
<servlet>
  <description></description>
  <display-name>FirstServlet</display-name>
  <servlet-name>FirstServlet</servlet-name>
  <servlet-class>first.FirstServlet</servlet-class>
 </servlet>
 <servlet-mapping>
  <servlet-name>FirstServlet</servlet-name>
  <url-pattern>/test</url-pattern>
 </servlet-mapping>
```

How to create servlet manually and deploy

1. D:\Spark2.0\ServletDemo>md WEB-INF
2. D:\Spark2.0\ServletDemo>cd WEB-INF
3. D:\Spark2.0\ServletDemo\WEB-INF>md classes
4. D:\Spark2.0\ServletDemo\WEB-INF>cd classes
5. D:\Spark2.0\ServletDemo\WEB-INF\classes>start notepad FirstServlet.java
6. D:\Spark2.0\ServletDemo\WEB-INF\classes>javac FirstServlet.java
7. D:\Spark2.0\ServletDemo\WEB-INF\classes>cd ..

8. D:\Spark2.0\ServletDemo\WEB-INF>start notepad web.xml
9. D:\Spark2.0\ServletDemo\WEB-INF>cd ..
10. D:\Spark2.0\ServletDemo>jar cvf first.war .

added manifest

adding: WEB-INF/(in = 0) (out= 0)(stored 0%)

adding: WEB-INF/classes/(in = 0) (out= 0)(stored 0%)

adding: WEB-INF/classes/FirstServlet.class(in = 642) (out= 408)(deflated 36%)

adding: WEB-INF/classes/FirstServlet.java(in = 320) (out= 208)(deflated 35%)

adding: WEB-INF/web.xml(in = 232) (out= 104)(deflated 55%)

How to deploy war in tomcat

---

1. start tomcat from instal directory
2. open browser
3. open http://localhost:8086/
4. click on manage apps
5. enter username and password
6. if not opened then set new username and password from install dir
7. select war and click deploy

Setp to prepare Web Application

---

1. click on fine menu
2. click on new
3. click on Dynamic Web Project
4. type project Name : FirstProject
5. click next button
6. click again next button
7. select .Web xml
8. click finish

Step to servlet program

---

1. right click on project
2. click on new
3. click on Servlet
4. type package name : first
5. type Servlet name : firstServlert
6. rename super class to jakarta.servlet.GenericServlet

Step to run servlet on server

---

1. right click on project
2. click on run as
3. click run on server
4. expand localhost
5. click finis

GenericServlet

GenericServlet is an abstract class provided by the Servlet API. It serves as a basic implementation of the Servlet interface, providing default implementations of all its methods except service(). It can be used as a foundation for building various types of servlets (not limited to HTTP).

1. Implements Servlet, ServletConfig, and Serializable.
2. Provides default implementations for init(), getServletConfig(), and destroy().
3. The service() method must be overridden by the developer, which processes the request and generates a response.

- **Usage**:
  It's useful when you're building a servlet for protocols other than HTTP, or when you want more control over the behavior that isn't locked into HTTP-specific operations.

HttpServlet

HttpServlet is a subclass of GenericServlet designed specifically for handling HTTP requests and responses. It is part of the Java Servlet API and is widely used in web applications.

1. Provides pre-defined methods like doGet(), doPost(), doPut(), doDelete(), doHead(), etc., for handling HTTP-specific requests.
2. Each method corresponds to an HTTP verb (GET, POST, etc.), allowing a clean separation of logic based on request type.
3. Developers only need to override the relevant method based on the HTTP method being used (e.g., override doGet() for GET requests).
4. The service() method is overridden internally, which then delegates the request to one of the appropriate doXXX() methods based on the HTTP request method.

- **Usage**:
  HttpServlet is the go-to class for building servlets that interact with web clients over HTTP. It simplifies handling web requests by allowing developers to focus on processing only the necessary HTTP methods.

ServletConfig (one per SERVLET):

1. ServletConfig is an interface which is present in javax.servlet.* package.
2. The purpose of ServletConfig is to pass some initial parameter values, technical information (driver name, database name, data source name, etc.) to a servlet.
3. An object of ServletConfig will be created one per servlet.
4. An object of ServletConfig will be created by the server at the time of executing public void init (ServletConfig) method.
5. An object of ServletConfig cannot be accessed in the default constructor of a Servlet class. Since, at the time of executing default constructor ServletConfig object does not exist.
6. By default ServletConfig object can be accessed with in init () method only but not in doGet and doPost. In order to use, in the entire servlet preserve the reference of ServletConfig into another variable and declare this variable into a Servlet class as a data member of ServletConfig

web.xml entries for ServletConfig

```
<servlet>
…….
<init-param>
      <param-name>Name of the parameter</param-name>
      <param-value>Value of the parameter</param-value>
</init-param>
…….
</servlet>
```

For example:

```
<servlet>
	<servlet-name>abc</servlet-name>
	<servlet-class>serv1</servlet-class>
	<init-param>
		<param-name>v1</param-name>
		<param-value>10</param-value>
	</init-param>
	<init-param>
		<param-name>v2</param-name>
		<param-value>20</param-value>
	</init-param>
</servlet>
```

RETRIEVING DATA from ServletConfig interface object:

In order to get the data from ServletConfig interface object we must use the following methods:

public String getInitParameter (String);

public Enumeration getInitParameterNames ();

Method-1 is used for obtaining the parameter value by passing parameter name.

String val1=config.getInitParameter ("v1");

String val2=config.getInitParameter ("v2");

String val3=config.getInitParameter ("v3");

Method-2 is used for obtaining all parameter names and their corresponding parameter values.

For example:

```
Enumeration en=config.getInitParameterNames ();
while (en.hasMoreElements ())
{
	Object obj=en.nextElement ();
	String pname= (String) obj;
```

String pvalue=config.getInitParameter (pname);

out.println (pvalue+" is the value of "+pname);

}

```
public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        ServletConfig config=getServletConfig ();
        String val1=config.getInitParameter ("v1");
        String val2=config.getInitParameter ("v2");
        String val3=config.getInitParameter ("v3");
        String val4=config.getInitParameter ("v4");
        pw.println ("<h3> Value of v1 is "+val1+"</h3>");
        pw.println ("<h3> Value of v2 is "+val2+"</h3>");
        pw.println ("<h3> Value of v3 is "+val3+"</h3>");
        pw.println ("<h3> Value of v4 is "+val4+"</h3>");
}
```

```
public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
        res.setContentType ("text/html");
        PrintWriter pw=res.getWriter ();
        ServletConfig config=getServletConfig ();
        Enumeration en=config.getInitParameterNames ();
        while (en.hasMoreElements ())
        {
                Object obj=en.nextElement ();
                String pname= (String) obj;
                String pvalue=config.getInitParameter (pname);
                pw.println ("</h2>"+pvalue+" is the value of "+pname+"</h2>");
        }
}
```

ServletContext (one per WEB APPLICATION):

1. ServletContext is an interface which is present in javax.servlet.* package.
2. Whenever we want to give a common data or global data to the group of servlets which belongs to same web application then we must create an object of ServletContext interface.

3. An object of ServletContext will be created by servlet container (server) whenever we deploy into the server.

4. In order to provide a common data to a group of servlets, we must write that data into web.xml file with the tag called <context-param>…</context-param>. This tag must be written with in <web-app>…</web-app> before <servlet>.

<web-app>

<context-param>

<param-name>Name of the param</param-name>

<param-value>Value of the param</param-value>

</context-param>

<servlet>

……..

……..

</servlet>

<servlet-mapping>

……..

……..

</servlet-mapping>

</web-app>

1. Whatever the data we write with in <context-param>…</context-param> that data will be paste automatically in the object of ServletContext interface and this object contains the in the form of (key, value) pair. Here, key represents context parameter name and value represents context parameter value.
2. The value of key must be always unique; if duplicate values are placed we get recent duplicate value for the key by overlapping previous values.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="htt
  <display-name>FirstServlet</display-name>

  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.jsp</welcome-file>
    <welcome-file>default.htm</welcome-file>
  </welcome-file-list>
  <context-param>
    <param-name>driver</param-name>
    <param-value>com.mysql.cj.jdbc.Driver</param-value>
</context-param>
  <servlet>
    <description></description>
    <display-name>FirstServlet</display-name>
    <servlet-name>FirstServlet</servlet-name>
    <servlet-class>first.FirstServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FirstServlet</servlet-name>
    <url-pattern>/test</url-pattern>
  </servlet-mapping>
</web-app>
```

```
public doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
{
        ……
        ……
        ServletContext ctx=this.getServletContext ();
        ……
        ……
}
```