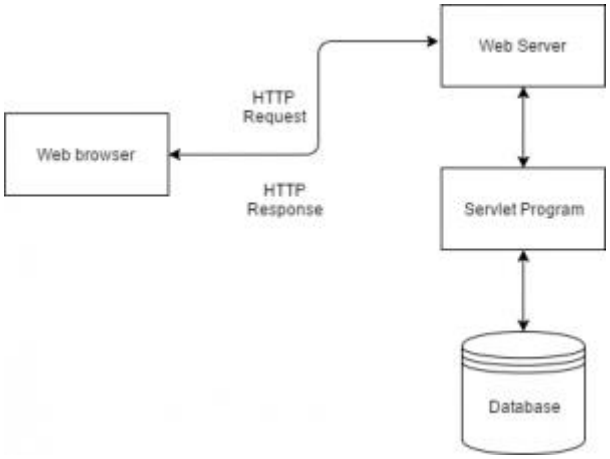
UNIT-I Servlets

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.

Properties of Servlets are as follows:

- Servlets work on the server-side.
- Servlets are capable of handling complex requests obtained from the webserver.

Servlet Architecture is can be depicted from the image itself as provided below as follows:



Execution of Servlets basically involves six basic steps:

1. The clients send the request to the webserver.
2. The web server receives the request.
3. The web server passes the request to the corresponding servlet.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the webserver.
6. The web server sends the response back to the client and the client browser displays it on the screen.

Now let us do discuss eccentric point that why do we need For Server-Side extensions?

The **server-side extensions** are nothing but the technologies that are used to create dynamic Web pages. Actually, to provide the facility of dynamic Web pages, Web pages need a container or Web server. To meet this requirement, independent Web server providers offer some proprietary solutions in the form of **APIs**(Application Programming Interface).

These **APIs** allow us to build programs that can run with a Web server. In this case, **Java Servlet** is also one of the component APIs of **Java Platform Enterprise Edition** which sets standards for creating dynamic Web applications in Java.

Before learning about something, it's important to know the need for that something, it's not like that this is the only technology available for creating dynamic Web

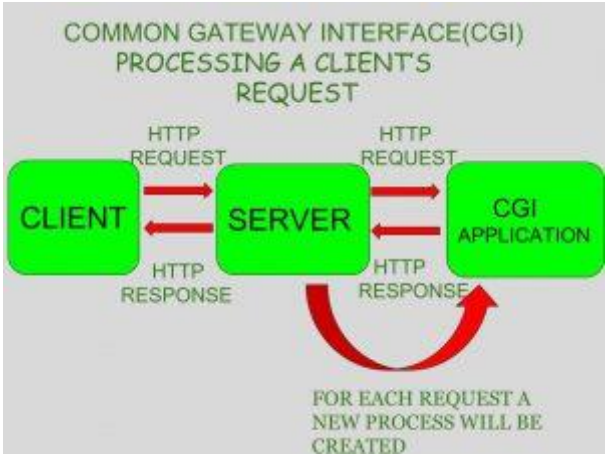
pages. The Servlet technology is similar to other Web server extensions such as **Common Gateway Interface**(CGI) scripts and **Hypertext Preprocessor** (PHP). However, Java Servlets are more acceptable since they solve the limitations of **CGI** such as low performance and low degree scalability.

What is CGI?

CGI is actually an external application that is written by using any of the programming languages like **C** or **C++** and this is responsible for processing client requests and generating dynamic content.

In CGI application, when a client makes a request to access dynamic Web pages, the Web server performs the following operations :

- It first locates the requested web page *i.e* the required CGI application using URL.
- It then creates a new process to service the client's request.
- Invokes the CGI application within the process and passes the request information to the application.
- Collects the response from the CGI application.
- Destroys the process, prepares the HTTP response, and sends it to the client.



So, in **CGI** server has to create and destroy the process for every request. It's easy to understand that this approach is applicable for handling few clients but as the number of clients increases, the workload on the server increases and so the time is taken to process requests increases.

Difference between Servlet and CGI

Servlet	CGI(Common Gateway Interface)
Servlets are portable and efficient.	CGI is not portable
In Servlets, sharing data is possible.	In CGI, sharing data is not possible.
Servlets can directly communicate with the webserver.	CGI cannot directly communicate with the webserver.

Servlet	CGI(Common Gateway Interface)
Servlets are less expensive than CGI.	CGI is more expensive than Servlets.
Servlets can handle the cookies.	CGI cannot handle the cookies.

Servlets API's:
Servlets are build from two packages:

- javax.servlet(Basic)
- javax.servlet.http(Advance)

Advantages of a Java Servlet over CGI

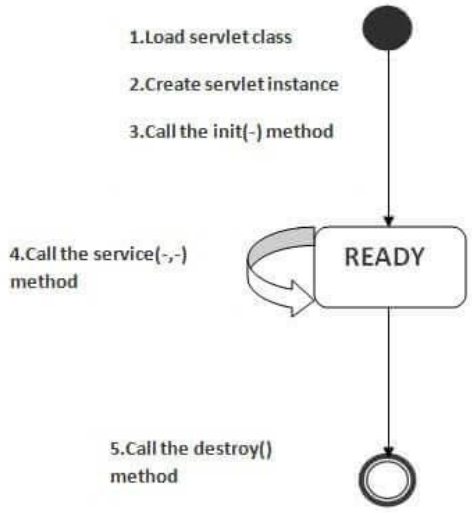
- Servlet is **faster** than CGI as it doesn't involve the creation of a new process for every new request received.
- Servlets, as written in Java, are platform-independent.
- Removes the overhead of creating a **new process** for each request as Servlet doesn't run in a separate process. There is only a single instance that handles all requests concurrently. This also saves the memory and allows a Servlet to easily manage the client state.
- It is a server-side component, so Servlet inherits the **security** provided by the Web server.
- The **API** designed for Java Servlet automatically acquires the advantages of the Java platforms such as platform-independent and portability. In addition, it obviously can use the wide range of APIs created on Java platforms such as **JDBC** to access the database.
- Many Web servers that are suitable for personal use or low-traffic websites are offered for free or at extremely **cheap costs** eg. Java servlet. However, the majority of commercial-grade Web servers are rather expensive, with the notable exception of Apache, which is free.

Life Cycle of a Servlet (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.

As displayed in this 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.



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 initial servlet. It is the life cycle method of the javax.servlet.Servlet interface. Syntax of the init method is given below:

```
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. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

```
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()
```

HttpServlet Class In Java

HttpServelt is an abstract class, it comes under package '**javax.servlet.http.HttpServlet**' . To create a servlet the class must extend the HttpServlet class and override at least one of its methods (doGet, doPost, doDelete, doPut). The HttpServlet class extends the GenericServlet class and implements a Serializable interface.

Constructor of HttpServlet Class

HttpServlet()

This is an abstract class so, the constructor does nothing.

Methods of HttpServlet Class

1. doGet() Method

- This method is used to handle the GET request on the server-side.
- This method also automatically supports HTTP HEAD (HEAD request is a GET request which returns nobody in response) request.
- The GET type request is usually used to *preprocess* a request.

Modifier and Type: protected void

Syntax:

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

Parameters:

- **request –** an HttpServletRequest object that contains the request the client has made of the servlet.
- **response –** an HttpServletResponse object that contains the response the servlet sends to the client.

Exceptions:

- **IOException –** if an input or output error is detected when the servlet handles the GET request.
- **ServletException –** Use to handle the GET request.

2. doPost() Method

- This method is used to handle the POST request on the server-side.
- This method allows the client to send data of unlimited length to the webserver at a time.
- The POST type request is usually used to post-process a request.

Modifier and Type: protected void

Syntax:

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

Parameter:

- **request –** an HttpServletRequest object that contains the request the client has made of the servlet.
- **response –** an HttpServletResponse object that contains the response the servlet sends to the client.

Throws:

- **IOException –** if an input or output error is detected when the servlet handles the POST request.
- **ServletException –** Use to handle the POST request.

3. doHead() Method

- This method is overridden to handle the HEAD request.
- In this method, the response contains the only header but does not contain the message body.
- This method is used to improve performance (avoid computing response body).

Modifier and Type: protected void
Syntax:
protected void doHead(HttpServletRequest request, HttpServletResponse response) throws ServletException,IOException

Parameters:

- **request –** an HttpServletRequest object that contains the request the client has made of the servlet.
- **response –** an HttpServletResponse object that contains the response the servlet sends to the client.

Exceptions:

- **IOException –** if an input or output error is detected when the servlet handles the HEAD request.
- **ServletException –** Use to handle the HEAD request.

4. doPut() Method

- This method is overridden to handle the PUT request.
- This method allows the client to store the information on the server(to save the image file on the server).
- This method is called by the server (via the service method) to handle a PUT request.

Modifier and Type: protected void
Syntax:
protected void doPut(HttpServletRequest request, HttpServletResponse response) throws ServletException,IOException

Parameters:

- **request –** an HttpServletRequest object that contains the request the client has made of the servlet.

- **response –** an HttpServletResponse object that contains the response the servlet sends to the client.

Throws:

- **IOException –** if an input or output error is detected when the servlet handles the PUT request.
- **ServletException –** Use to handle the PUT request.

5. doDelete() Method

- This method is overridden to handle the DELETE request.
- This method allows a client to remove a document or Web page from the server.
- While using this method, it may be useful to save a copy of the affected URL in temporary storage to avoid data loss.

Modifier and Type: protected void

Syntax:

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

Parameters:

- **request –** an HttpServletRequest object that contains the request the client has made of the servlet.
- **response –** an HttpServletResponse object that contains the response the servlet sends to the client.

Exceptions:

- **IOException –** if an input or output error is detected when the servlet handles the DELETErequest.
- **ServletException –** Use to handle the DELETE request.

6. doOptions() Method

- This method is overridden to handle the OPTIONS request.
- This method is used to determine which HTTP methods the server supports and returns an appropriate header.

Modifier and Type: protected void

Syntax:

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

Parameters:

- **request –** an HttpServletRequest object that contains the request the client has made of the servlet.
- **response –** an HttpServletResponse object that contains the response the servlet sends to the client.

Exceptions:

- **IOException –** if an input or output error is detected when the servlet handles the OPTIONS request.
- **ServletException –** Use to handle the OPTIONS request.

7. doTrace() Method

- This method is overridden to handle the TRACE request.
- This method returns the headers sent with the TRACE request to the client so that they can be used in debugging.

Modifier and Type: protected void
Syntax:
protected void doTrace(HttpServletRequest request, HttpServletResponse response) throws ServletException,IOException

Parameters:

- **request –** an HttpServletRequest object that contains the request the client has made of the servlet.
- **response –** an HttpServletResponse object that contains the response the servlet sends to the client.

Exceptions:

- **IOException –** if an input or output error is detected when the servlet handles the TRACE request.
- **ServletException –** Use to handle the TRACE request.

8. getLastModified() Method

- This method returns the time the HttpServletRequest object was last modified.
- If time is unknown the method will return a negative number.
- This method makes browser and proxy caches work more effectively.
- also reducing the load on server and network resources.

Modifier and Type: protected long
Syntax:
protected long getLastModified(HttpServletRequest request)

Parameter: *request* – an HttpServletRequest object that contains the request the client has made of the servlet.

9. service() Method

This method receives standard HTTP requests from the public *service* method and dispatches them to the *doXXX* methods defined in this class.
Modifier and Type: protected void
Syntax:
protected void service(HttpServletRequest request, HttpServletResponse response) throws ServletException,IOException

Parameters:

- **request –** an HttpServletRequest object that contains the request the client has made of the servlet.
- **response –** an HttpServletResponse object that contains the response the servlet sends to the client.

Exceptions:

- **IOException –** if an input or output error is detected when the servlet handles the HTTP request.
- **ServletException –** Use to handle the HTTP request.

10. service() Method

This method is used to dispatch client requests to the protected service method.

Modifier and Type: public void
Syntax:
public void service(ServletRequest reqequest,ServletResponse response) throws ServletException,IOException

Parameters:

- **request –** an HttpServletRequest object that contains the request the client has made of the servlet.
- **response –** an HttpServletResponse object that contains the response the servlet sends to the client.

Exceptions:

- **IOException –** if an input or output error is detected when the servlet handles the HTTP request.
- **ServletException –** Use to handle the HTTP request.

Example:

- Java

```
package com.gfg;

import javax.servlet.*;

import javax.servlet.http.*;

  // here GFGServlet class inherit HttpServlet

public class GFGServlet extends HttpServlet {

      // we are defining the doGet method of HttpServlet

   // abstract class

   public void doGet(HttpServletRequest rq,

                     HttpServletResponse rs)

   {
```

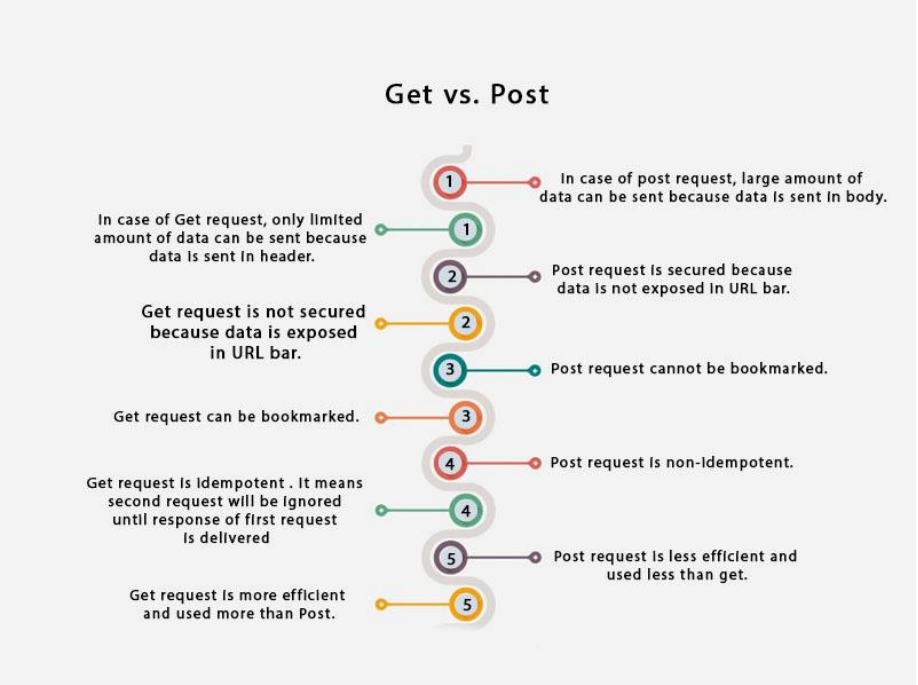
```
    // here user write code to handle doGet request
  }
  // we are defining the doPost method of HttpServlet
  // abstract class
  public void doPost(HttpServletRequest rq,
                HttpServletResponse rs)
  {
    // here user write code to handle doPost request
  }
} // class ends
```

Note: This is server-side code, it is only for clarification on how to use HttpServlet class.

Get vs. Post

There are many differences between the Get and Post request. Let's see these differences:

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.



GET and POST

Two common methods for the request-response between a server and client are:

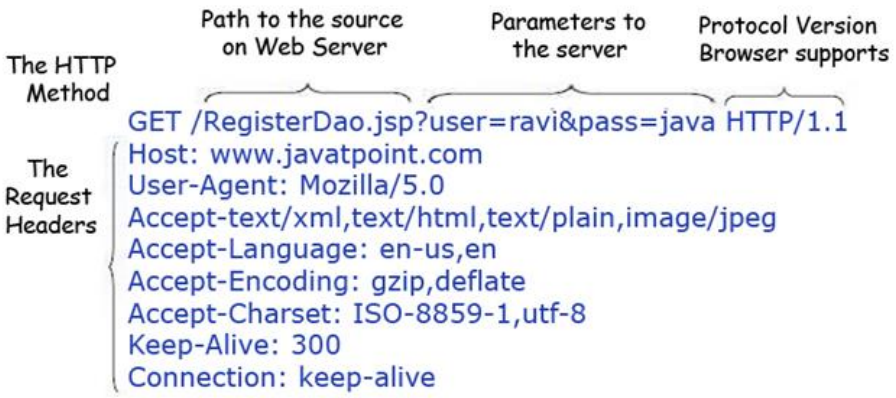
- **GET**- It requests the data from a specified resource
- **POST**- It submits the processed data to a specified resource

Anatomy of Get Request

The query string (name/value pairs) is sent inside the URL of a GET request:

1. GET/RegisterDao.jsp?name1=value1&name2=value2

As we know that data is sent in request header in case of get request. It is the default request type. Let's see what information is sent to the server.



Some other features of GET requests are:

- It remains in the browser history
- It can be bookmarked
- It can be cached
- It have length restrictions
- It should never be used when dealing with sensitive data
- It should only be used for retrieving the data

Anatomy of Post Request

The query string (name/value pairs) is sent in HTTP message body for a POST request:

1. POST/RegisterDao.jsp HTTP/1.1
2. Host: www. javatpoint.com
3. name1=value1&name2=value2

As we know, in case of post request original data is sent in message body. Let's see how information is passed to the server in case of post request.

Some other features of POST requests are:

- This requests cannot be bookmarked
- This requests have no restrictions on length of data
- This requests are never cached
- This requests do not retain in the browser history

Difference Between Web server and Application server

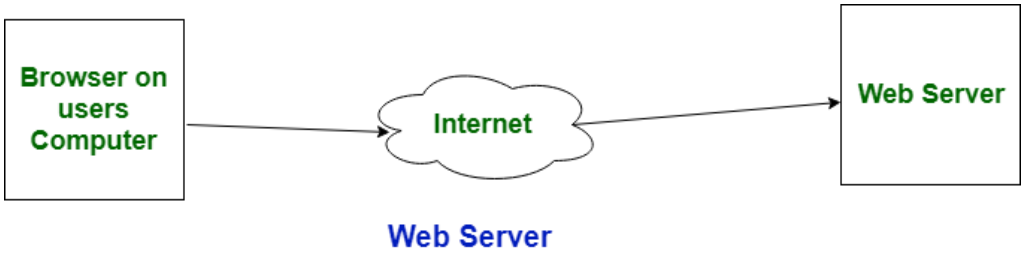
A server is a central repository where information and computer programs are held and accessed by the programmer within the network. **Web server** and **Application server** are kinds of the server which employed to deliver sites and therefore the latter deals with application operations performed between users and back-end business applications of the organization.

Web Server:

It is a computer program that accepts the request for data and sends the specified documents. Web server may be a computer where the online content is kept. Essentially internet server is employed to host sites however there exist different web servers conjointly like recreation, storage, FTP, email, etc.

Example of Web Servers:

- Apache Tomcat
- Resin



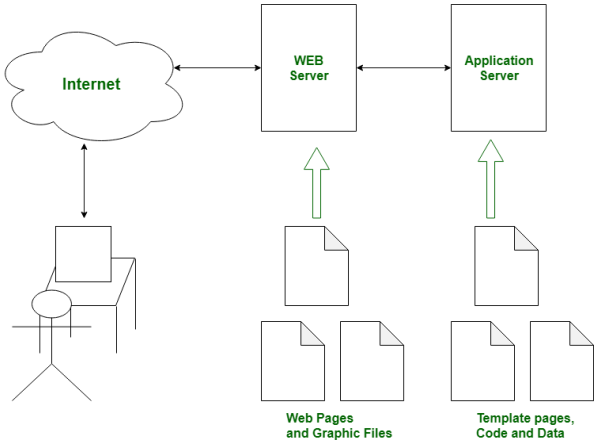
Web Server

Application server:

It encompasses Web container as well as EJB container. Application servers organize the run atmosphere for enterprises applications. Application server may be a reasonably server that mean how to put operating system, hosting the applications and services for users, IT services and organizations. In this, user interface similarly as protocol and RPC/RMI protocols are used.

Examples of Application Server:

- Weblogic
- JBoss
- Websphere



Difference between web server and application server:

S.No	Web Server	Application Server
1.	Web server encompasses web container only.	While application server encompasses Web container as well as EJB container.
2.	Web server is useful or fitted for static content.	Whereas application server is fitted for dynamic content.
3.	Web server consumes or utilizes less resources.	While application server utilizes more resources.
4.	Web servers arrange the run environment for web applications.	While application servers arrange the run environment for enterprises applications.
5.	In web servers, multithreading is supported.	While in application server, multithreading is not supported.
6.	Web server's capacity is lower than application server.	While application server's capacity is higher than web server.
7.	In web server, HTML and HTTP protocols are used.	While in this, GUI as well as HTTP and RPC/RMI protocols are used.
8.	Processes that are not resource-intensive are supported.	Processes that are resource-intensive are supported.
9.	Transactions and connection pooling is not supported.	Transactions and connection pooling is supported.
10.	The capacity of fault tolerance is low as compared to application servers.	It has high fault tolerance.
11.	Web Server examples are Apache HTTP Server, Nginx.	Application Servers example are JBoss, Glassfish.

Steps to create a servlet example

There are given 6 steps to create a **servlet example**. These steps are required for all the servers. The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

Here, we are going to use **apache tomcat server** in this example. The steps are as follows:

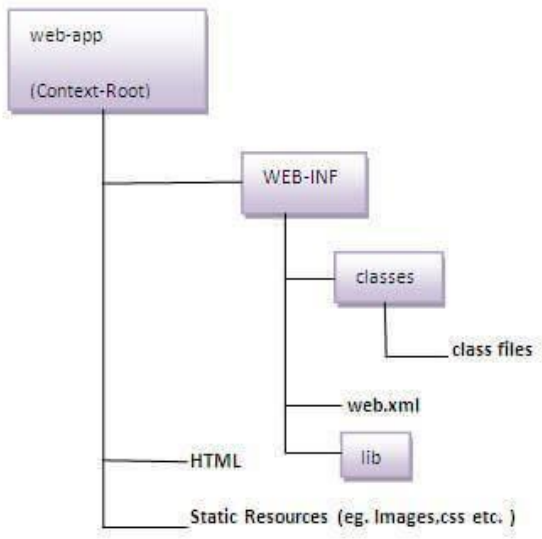
1. Create a directory structure
2. Create a Servlet

3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

1)Create a directory structures

The **directory structure** defines that where to put the different types of files so that web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

2)Create a Servlet

There are three ways to create the servlet.

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

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.

In this example we are going to create a servlet that extends the HttpServlet class. In this example,

we are inheriting the HttpServlet class and providing the implementation of the doGet() method.

Notice that get request is the default request.

DemoServlet.java

```
1. import javax.servlet.http.*;
2. import javax.servlet.*;
3. import java.io.*;
4. public class DemoServlet extends HttpServlet{
5. public void doGet(HttpServletRequest req,HttpServletResponse res)
6. throws ServletException,IOException
7. {
8. res.setContentType("text/html");//setting the content type
9. PrintWriter pw=res.getWriter();//get the stream to write the data
10.
11. //writing html in the stream
12. pw.println("<html><body>");
13. pw.println("Welcome to servlet");
14. pw.println("</body></html>");
15.
16. pw.close();//closing the stream
17. }}
```

3)Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

---

4)Create the deployment descriptor (web.xml file)

The **deployment descriptor** is an xml file, from which Web Container gets the information about the servlet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
1. <web-app>
2.
3. <servlet>
4. <servlet-name>sonoojaiswal</servlet-name>
5. <servlet-class>DemoServlet</servlet-class>
6. </servlet>
7.
8. <servlet-mapping>
9. <servlet-name>sonoojaiswal</servlet-name>
10. <url-pattern>/welcome</url-pattern>
11. </servlet-mapping>
12.
13. </web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file. The elements are as follows:

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

---

5)Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

Servlet – War File

A war (web archive) A web project's files are contained in this file. It may contain files such as servlet, xml, jsp, image, html, css, and js. Here, we'll go through what a war file is, how to make one, how to deploy one, and how to remove one.

What is WAR File?

The WAR file (Web Application Resource or Web Application ARchive) is a container for JAR files, JavaServer Pages, Java Servlets, Java classes, XML files, tag libraries, static sites (HTML and associated files), and other resources that make up an online application. A file entitled web.xml is located in the /WEB-INF directory of the WAR file, and it describes the structure of the online application. The web.xml file isn't technically essential if the online application is only providing JSP files. If the online apps utilize servlets, the servlet container looks at web.xml to figure out which servlet a URL request should go to.

Advantages

1. Web applications may be easily tested and deployed.
2. The version of the deployed application may be easily identified.
3. WAR files are supported by all Java EE containers.
4. WAR files are supported by the MVC framework.

How to create a WAR File?

On the command prompt, type the following command to generate a war file:

jar -cvf project_name.war*
Here,

- -c: It is used to create a file
- -v: It is used to generate the verbose output
- -f: It is used to specify the archive filename
- *: It signifies all the files of this directory

Example – D:\apps\gfgapp>jar -cvf testapp1.war*

ServletRequest Interface

When a Servlet accepts a call from a client, then it receives two objects, one is a ServletRequest and the other is the ServletResponse. ServletRequest encapsulates the Communications from the client to the server, while ServletResponse encapsulates the Communication from the Servlet back to the client. The Object of the ServletRequest is used to provide the client request information data to the Servlet as a content type, a content length, parameter names, and the values, header information and the attributes, etc.

ServletRequest allows the Servlet to access information such as:

- Names of the parameters are passed by the client
- The protocol [scheme] such as the HTIP POST and PUT methods being used by the client
- The names of the remote host that are made the request
- The server that received it
- An input stream is for reading the binary data from the request body

The Subclasses of the ServletRequest allows the Servlet to retrieve more protocol-based-specific data. For example, HttpServletRequest contains the methods for accessing HTTP-specific based header Information.

ServletResponse allows Servlet

- To settle up the content length and mime type of the reply
- Provides an output stream and a Writer

Through ServletResponse, the Servlet can send the reply data. The Subclasses of ServletResponse provide the Servlet with more protocol-based-specific capabilities. For example, up ServletResponse can contain methods that allow the Servlet to Control the HTTP-specific header information.

Example of ServletRequest to display the name of the user

In this example, we are displaying the name of the user in this servlet, And for this, we have used the getParameter method that returns the value for the given request parameter name. We have the use of Servlet – Request Interface in it And When we use it within this example and running will successfully establish the method.

newindex.html

- HTML

```
<form action="welcome User" method="get">
  Enter your name<input type="text" name="Name"><br>
<input type="Submit" value="Login">
</form>
```

MockServer.java

- Java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class MockServ extends HttpServlet {
   public void doGet(HttpServletRequest req,
                     HttpServletResponse resp)
      // Uses throws exception
      throws ServletException, IOException
```

```
    {
        resp.setContentType("text/html");

        // sets Content Type
        PrintWriter ab = resp.getWriter();

        String Name = req.getParameter("Name");

        // thiswill return value
        ab.println(" Welcome User " + Name);

        // this will print the Name Welcome User
        ab.close();
    }
}
```

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.

If the configuration information is modified from the web.xml file, we don't need to change the servlet. So, it is easier to manage the web application if any specific content is modified from time to time.

Advantage of ServletConfig

The core advantage of ServletConfig is that you don't need to edit the servlet file if information is modified from the web.xml file.

Methods of ServletConfig interface

1. **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
2. **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
3. **public String getServletName():**Returns the name of the servlet.
4. **public ServletContext getServletContext():**Returns an object of ServletContext.

ServletContext Interface

An object of ServletContext is created by the web container at time of deploying the project. This object can be used to get configuration information from web.xml file. There is only one ServletContext object per web application.

If any information is shared to many servlet, it is better to provide it from the web.xml file using the **<context-param>** element.

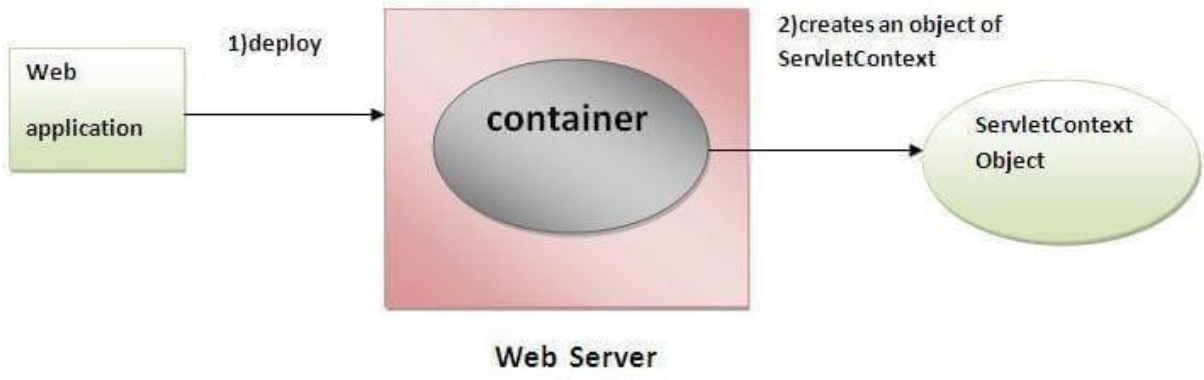
Advantage of ServletContext

Easy to maintain if any information is shared to all the servlet, it is better to make it available for all the servlet. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet. Thus it removes maintenance problem.

Usage of ServletContext Interface

There can be a lot of usage of ServletContext object. Some of them are as follows:

1. The object of ServletContext provides an interface between the container and servlet.
2. The ServletContext object can be used to get configuration information from the web.xml file.
3. The ServletContext object can be used to set, get or remove attribute from the web.xml file.
4. The ServletContext object can be used to provide inter-application communication.



Generic Servlet Class

GenericServlet implements the Servlet interface and provides an implementation for all its method except the **service() method** hence it is **abstract**. GenericServlet class defines a protocol-independent(HTTP-less) servlet. However, while building a website or an online application, we may want to have HTTP protocol, in that case, we must extend HttpServlet

instead of GenericServlet. Developing Servlet by extending GenericServlet is very easy because we have to provide implementation only for the **service()** method. GenericServlet class is in javax.servlet package (javax.servlet.GenericServlet).

The prototype of **service( ) method** is:

public void service (ServletRequest req, ServletResponse resp) throws ServletException,IOException

This method is automatically called by the server whenever a request for a GenericServlet arrives. The **service()** method accepts two parameters:

1. A ServletRequestObject
2. A SerlvetResponseObject

The ServletRequest object allows to read data provided by the client request and the ServletResponse object is used to send the response to the client

Methods of GenericServlet class

1. **public void init(ServletConfig config)** is used to initialize the servlet.
2. **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.
3. **public void destroy()** is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. **public ServletConfig getServletConfig()** returns the object of ServletConfig.
5. **public String getServletInfo()** returns information about servlet such as writer, copyright, version etc.
6. **public void init()** it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. **public ServletContext getServletContext()** returns the object of ServletContext.
8. **public String getInitParameter(String name)** returns the parameter value for the given parameter name.
9. **public Enumeration getInitParameterNames()** returns all the parameters defined in the web.xml file.
10. **public String getServletName()** returns the name of the servlet object.
11. **public void log(String msg)** writes the given message in the servlet log file.
12. **public void log(String msg,Throwable t)** writes the explanatory message in the servlet log file and a stack trace.

RequestDispatcher in Servlet

The RequestDispatcher is an Interface that comes under package ***javax.servlet***. Using this interface we get an object in servlet after receiving the request. Using the RequestDispatcher object we send a request to other resources which include (*servlet, HTML file, or JSP file*). A RequestDispatcher object can be used to forward a request to the resource or to include the resource in a response. The resource can be dynamic or static.

How to Create an Object of RequestDispatcher?

There are **three ways** to get an object:

1. RequestDispatcher requestDispatcher=ServletContext.getRequestDispatcher(String path);

Description:

- *public interface ServletContext. Defines a set of methods that a servlet uses to communicate with its servlet container.*
- *path is a string specifying the pathname to the resource(servlet, HTML file, or JSP file).*

2. RequestDispatcher requestDispatcher=ServletContext.getNamedDispatcher(String name);

Description:

- *public interface ServletContext. Defines a set of methods that a servlet uses to communicate with its servlet container.*
- *name is a string specifying the name of a servlet to wrap.*

3. RequestDispatcher requestDispatcher=request.getRequestDispatcher("String path");

Description:

- *request is the HttpServletRequest type object.*
- *path is a string specifying the pathname to the resource. If it is relative, it must be relative to the current servlet.*

Servlet – sendRedirect() Method

Servlets are the Java programs that run on the server-side and generate dynamic responses to the client request. Servlet accepts the request from the browser, processes it, and generates the response to the browser. While processing the request, let's say if need to call another servlet from a different server, we need to redirect the response to that resource.

To achieve this, Java servlets provide sendRedirect() method in HttpServletResponse interface in ***javax.servlet.http*** package. To understand better, let's look at some real-time examples.

Example 1: Nowadays, there are so many online shopping sites, where we can purchase goods. Once we select the product, are ready to purchase, and click on pay, the browser will redirect to the respective online payment page. Here, the response from the shopping website redirects it to the payment page and a new URL can be seen in the browser.

Example 2: In some online education applications, if we want to include a google search operation or to include a link to another website for more information on the topic, we need to redirect the response to the specific URL.

HttpServletResponse Interface

HttpServletResponse extends the ServletResponse interface to provide functionality specific to HTTP requests and responses. It provides methods to access HTTP headers and cookies.

public interface HttpServletResponse extends ServletResponse

sendRedirect() Method

sendRedirect() method redirects the response to another resource, inside or outside the server. It makes the client/browser to create a new request to get to the resource. It sends a temporary redirect response to the client using the specified redirect location URL.

Syntax:

void sendRedirect(java.lang.String location) throws java.io.IOException

- sendRedirect() accepts the respective URL to which the request is to be redirected.
- Can redirect the request to another resource like Servlet, HTML page, or JSP page that are inside or outside the server.
- It works on the HTTP response object and always sends a new request for the object.
- A new URL that is being redirected can be seen in the browser.

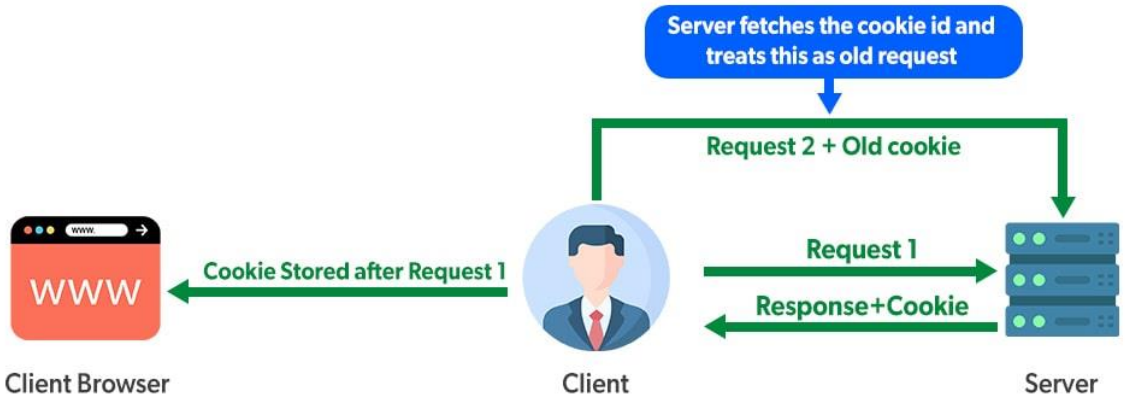
Cookies in Servlet

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.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

It is **valid for multiple session** . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.
3. Space character and image are considered invalid.
4. Security is less.

Note: Gmail uses cookie technique for login. If you disable the cookie, gmail won't work.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.