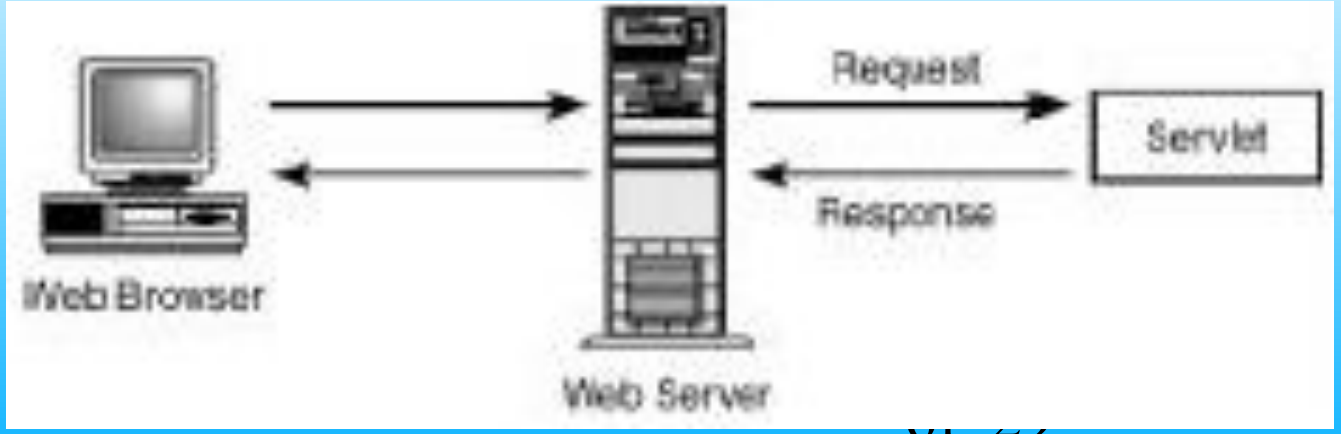
Web application & Servlet

History of Web application

- Web servers began as a mechanism for sending static Web documents to a Web browser using a simple protocol, HTTP
- Before servlets, there were various options of creating web applications:
  - ✔ **Common Gateway Interface (CGI)**
    - CGI enables developers to create separate applications that communicate with the Web server.
    - CGI program will start up a new process for each request then shuts down.This is drawback as it is a performance bottleneck
  - ✔ **FastCGI**
  - ✔ **ASP**
    - An extension to server-side includes from Microsoft Web server
  - ✔ **Server-side Java Script**
    - The Web servers support additional HTML tags that can be dynamically processed by the Web server so that the resulting HTML viewed in the Web browser can be customized by server-side

Concept

- *Servlets* are small Java programs that run on a Web server.
  - ✔ extends the capabilities of a Web server
  - ✔ servlets are to Web server as applets are to Web browser.
  - ✔ A *servlet* is a dynamically loaded module that services requests from a Web server.



Advantages of Servlet

- A *servlet* runs entirely inside the Java Virtual Machine.
  - ✔ Because the servlet is running on the server side, it does not require any Java-support from Web browser
  - ✔ They are safe and portable across operating systems and Web servers
  - ✔ Because servlets are written in Java, they can seamlessly take advantage of other Java technologies such as JDBC, JNDI, and others.
  - ✔ More efficient than CGI
  - ✔ All major web servers support servlets
  - ✔ Servlets are secure since they operate within the context of a security manager.

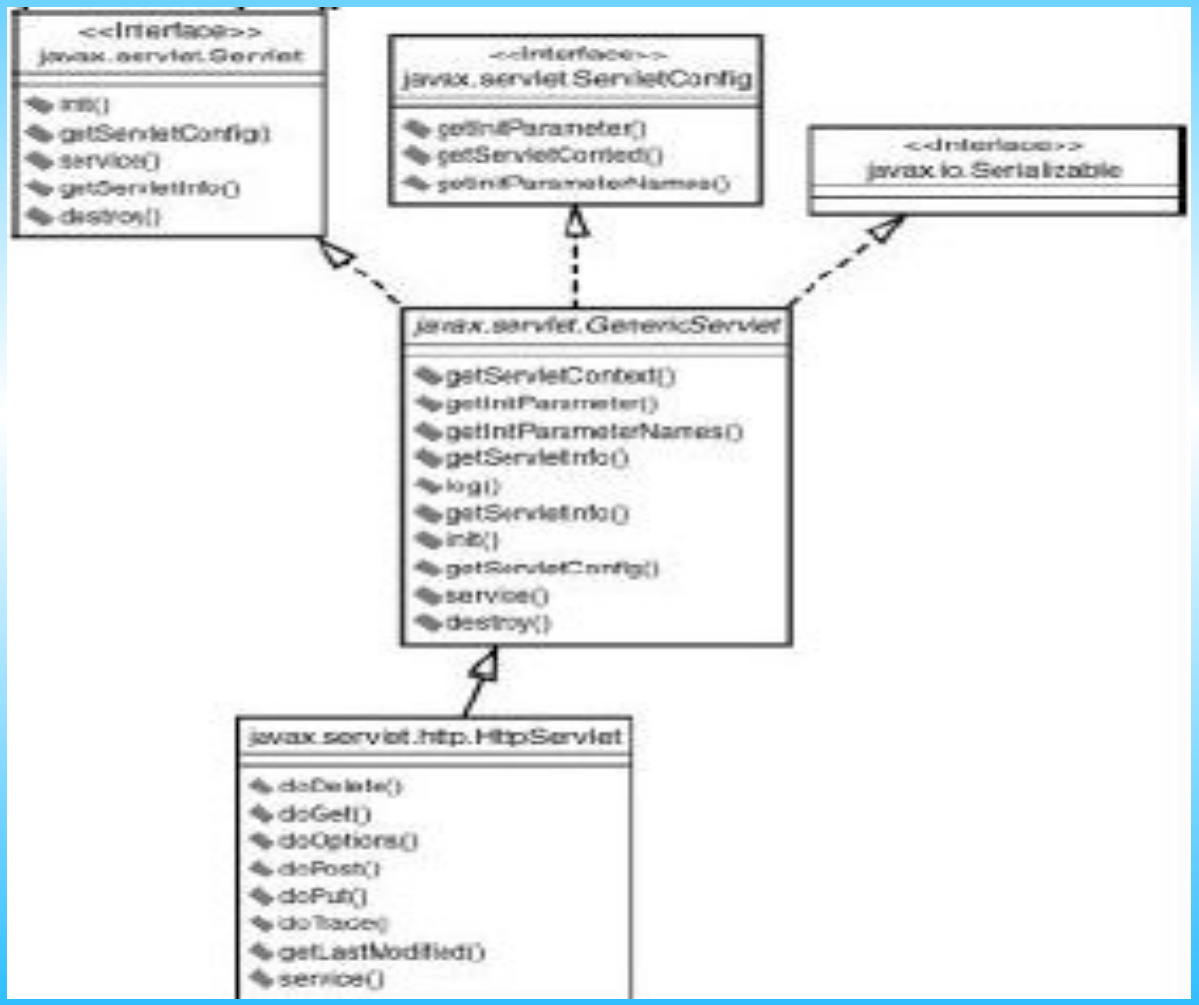
Java Servlet Architecture

- Two packages make up the servlet architecture: the javax.servlet and javax.servlet.http packages.
  - ✔ The javax.servlet package contains the generic interfaces and classes that are implemented and extended by all servlets.
  - ✔ The javax.servlet.http package contains the classes that are extended when creating HTTP-specific servlets.
- They are not part of the core Java API
- The **javax.servlet** and **javax.servlet.http** packages are bundled with the **Java Servlet Development Kit (JSDK)** or any Java Web Server

The Servlet interface

- At the heart of servlet architecture is the interface javax.servlet.Servlet.
  - ✔ It provides the framework for all servlets.
- The Servlet interface defines five methods. The three most important are as follows:
  - ✔ **init()** method—Initializes a servlet
  - ✔ **service()** method—Receives and responds to client requests
  - ✔ **destroy()** method—Performs cleanup
- All servlets must implement this interface, either directly or through inheritance.

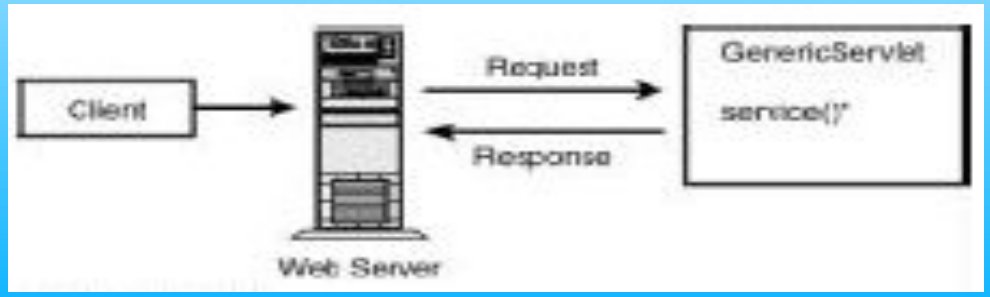
The servlet framework.



*

GenericServlet

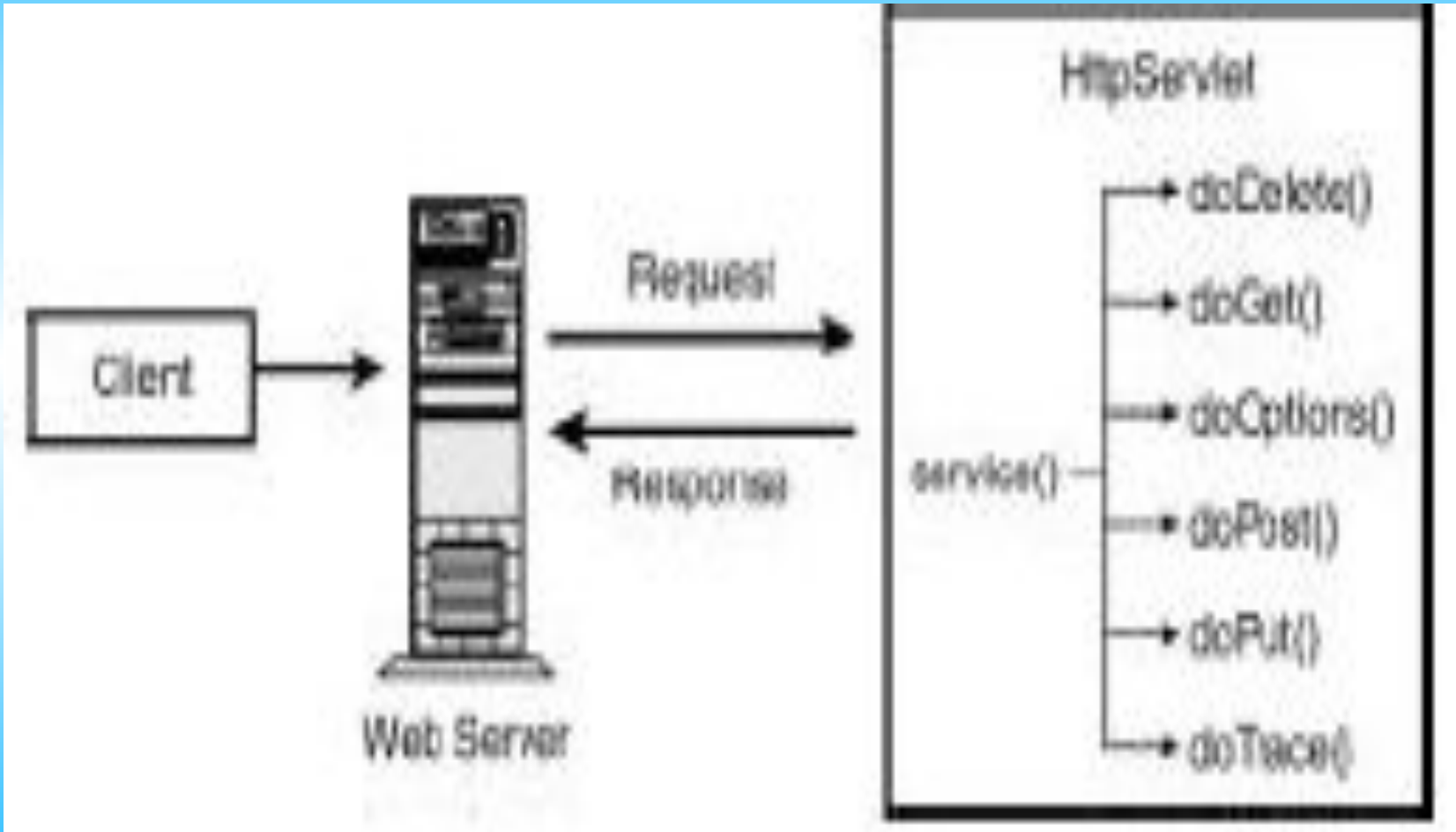
- To extend the GenericServlet class, you must implement the service() method.
  - ✔ *public abstract void service(ServletRequest req, ServletResponse res) throws ServletException, IOException;*
- The two objects that the service() method receives are ServletRequest and ServletResponse.
  - ✔ The **ServletRequest** object holds the information that is being sent to the servlet
  - ✔ The **ServletResponse** object is where you place the data you want to send back to the server.



HttpServlet

- The HttpServlet class has already implemented the service() method for you.
  protected void service(HttpServletRequest req, HttpServletResponse resp)throws ServletException, IOException;
- When the HttpServlet.service() method is invoked, it reads the method type stored in the request and determines which method to invoke based upon this value.
  - ✔ If the method type is GET, it will call doGet(). If the method type is POST, it will call doPost().
  - ✔ These are the methods that you will want to override
  - ✔ Typically, we should not override the service() method, but the doPost() and/or doGet() method

HttpServlet



*

A Basic Servlet

- The first step in defining a servlet is to create an object that implements the javax.servlet.Servlet interface.
- There are two standard ways to implement the Servlet interface.
  - ✔ The first is to subclass javax.servlet.**GenericServlet**.
  - ✔ The second way to implement Servlet is to subclass javax.servlet.http.**HttpServlet**.
- The Web server associates servlets with URLs. As: http://hostserver:8080/greeting/servlet/GreetingServlet
- When the user accesses a URL represented by a servlet, the Web server first checks to see if it has loaded that servlet.
  - ✔ an instance of the servlet's class is created using the default constructor,
  - ✔ The doPost() or doGet() get executed to generate dynamic html

A simple web application

- To deploy a simple web application, let say \greeting we need:
  - ✔ An HTML page with a form to collect client information
    - Put this **index.html** into ..\webapps\greeting
  - ✔ A servlet to process the request and generate HTML to display the response
    - Put the servlet class into \greeting\WEB-INF\classes
    - servlet.jar must be on the CLASSPATH to compile the servlet
  - ✔ A deployment descriptor
    - Put the deployment descriptor web.xml into \greeting\WEB-INF\web.xml
    - (optional –not necessary if your web application located as a sub directory of webapps directory
- Start Tomcat
- Open http://hostserver:8080/greeting to test your servlet

*

Servlet life cycle

- The Servlet interface defines the life cycle methods. These methods are init(), service(), and destroy().
- The init() method is where the servlet's life cycle begins.
  - ✔ It is called by the server immediately after the servlet is instantiated.
  - ✔ It is called only once.
  - ✔ public void init(ServletConfig config) throws ServletException;
  - ✔ In the init() method, the servlet creates and initializes any resources, including data members, that it will be using while handling requests.

Servlet life cycle

- The service() method handles all requests sent by a client.
  - ✔ The entry point for executing application logic in a servlet
  - ✔ It cannot start servicing requests until the init() method has been executed.
  - ✔ The service() method implements a request and response paradigm.
  - ✔ We should never implement this method directly if extending from HttpServlet

Servlet life cycle

- The destroy() method signifies the end of a servlet's life.
  - ✔ When a service is being shut down, it calls the servlet's destroy() method.
  - ✔ This is where any resources that were created in the init() method will be cleaned up.
  - ✔ If you have an open database connection, you should close it here.
  - ✔ This is also a good place to save any persistent information that will be used the next time the servlet is loaded.
- The getServletConfig() method returns the servlet's ServletConfig object, which contains the servlet's startup configuration and initialization parameters.

Invoke a Servlet

- There are two ways to invoke a servlet.
  - ✔ The first is to just reference the servlet by name in the URL.
    - The following URL will execute the servlet on your local server: http://localhost:8080/servlet/BasicServlet
    - Using this method defaults the request method to GET, which will invoke the servlet's doGet() method.
  - ✔ The second way to invoke the servlet is to create an HTML page that will send a request to the servlet using the POST method.
    - This will invoke the servlet's doPost() method.

doGet() and doPost()

- The doGet() method handles GET requests, and the doPost() method handles POST requests.
- Both of these methods receive HttpServletRequest and HttpServletResponse objects.
  - ✔ These objects encapsulate the request/response paradigm.
  - ✔ The HttpServletRequest contains information sent from the client
  - ✔ The HttpServletResponse contains the information that will be sent back to the client.
- response.setContentType("text/html");
  - ✔ can set this response property only once.
  - ✔ must set this property before writing to a Writer or an OutputStream.
- PrintWriter out = response.getWriter();
  - ✔ enable you to write HTML text that will be sent back to the client in the HttpServletResponse object.

The ServletRequest

- The ServletRequest interface defines an object used to encapsulate information about the client's request. Information in the ServletRequest object includes parameter name/value pairs, attributes, and an input stream.
  - ✔ getAttribute(String name) returns value of the object keyed by the name string for the current request.
  - ✔ getParameter(String name) method returns the value of the requested parameter.
  - ✔ If the parameter has or could have more than one value, use the getParameterValues() method.
  - ✔ getRemoteAddress() method returns a String representing the IP address of the client sending the request.

The ServletResponse

- The ServletResponse interface defines an object for sending MIME data back to the client from the servlet's service method.
  - ✔ getWriter() method returns a print writer used for writing formatted text to the response object.
  - ✔ getOutputStream() method returns an output stream used for writing binary data to the response.
  - ✔ setContentType() method sets the content type of the current response.
    - You can only set this property once for the current response.
    - This method must be called before calling the getWriter() or getOutputStream() methods.

The HttpServletRequest

- The HttpServletRequest interface defines an object that provides the HttpServlet.service() to access HTTP header information sent by the client. It has 26 methods
  - ✔ getQueryString() method returns the query string from the request.
  - ✔ getSession(true) method returns the session associated with the request. If there is no valid session and the boolean parameter passed in is true, then it will create a new session.
  - ✔ getRemoteUser() method returns the name of the user making the request. If the name is not available, null is returned.
  - ✔ The getMethod() method returns the HTTP method used by the client request.

The HttpServletResponse

- The HttpServletResponse interface has 39 fields and 10 methods,
  - ✔ The encodeURL() method encodes the passed-in String and returns it. If no changes are necessary, then it simply returns the String.
  - ✔ sendError(int er, String msg) method sends an error to the client in the response object. The error consists of the int status code and a String message.
  - ✔ The sendRedirect() method redirects the client to the passed-in URL, which must be an absolute URL.

The ServletConfig

- The ServletConfig interface defines an object generated by a servlet engine and is used to pass configuration information to a servlet during start up.
  - ✔ The getInitParameter() method returns a String containing the value of the initialization parameter's name/value pair
  - ✔ The getServletContext() method returns a reference to the current ServletContext object.
  - ✔ The getServletName() method returns the registered servlet's name.
- The init parameter and others configuration informationcan be specified in deployment descriptor- the web.xml file

The ServletContext

- The ServletContext interface defines an object, to be created by a servlet engine, that contains information about the servlet's environment, helpful for sharing data
  - ✔ The **getAttribute(String name)** method returns an Object stored in the ServletContext and keyed by the name value passed in.
    - This is one of methods used to share resources between servlets.
    - The returning object must be downcast to its original type before use.
  - ✔ The **setAttribute(String name, Object obj)** method stores an Object in the ServletContext and binds the Object obj to the given name.
    - If the name already exists in the ServletContext, then it is replaced.

The HttpSession

- The HttpSession interface defines an object that provides an association between a client and server persisting over multiple connections, helpful for session tracking
- A HttpSession is associated with a HttpServletRequest and obtain by request.getSession()
- Using HttpSession gives you the ability to maintain state between transactions.
  - ✔ getAttribute() method returns a reference to the named object in the current session. The object must be downcasted to its original type.
  - ✔ getLastAccessedTime() method returns the last time, in milliseconds, the client sent a request with HttpSession.
  - ✔ setAttribute() method binds the passed-in object to the passed-in String and puts the object into the session. If there is an object in the session already bound to the name, it is replaced.

*

Servlet Collaboration

- A servlet can receive a request from client and delegates the processing to another servlet.
- There are 2 solutions for this collaboration:
  - ✔ Using servlet chaining:
    - configure the servlet container to chain the servlet;
    - not supported by servlet API specification
  - ✔ Using Request Dispatching
    - Allow a servlet to dispatch a request to another servlet for generating the response
    - The RequestDispatcher used for this purpose

RequestDispatcher

- A servlet obtains a RequestDispatcher object for a resource by calling:
  - ✔ ServletContext.getRequestDispatcher(String URLpath)
  - ✔ ServletRequest. getRequestDispatcher(String URLpath)
  - ✔ ServletContext.getNameDispatcher(String name)
- void forward(request,response)
  - ✔ Forwards a request from a servlet to another resource
  - ✔ The request and response parameters must be the same objects as were passed to the calling servlet
- void include(request,response)
  - ✔ Includes the content of a resource (servlet, JSP page, HTML file) in the response