
JAVA
J2EE


Java EE™


JAKARTA™ EE

J2SE VS J2EE

Feature	Java SE	Java EE
Definition	Standard edition with core functionalities	Enterprise edition with extended functionalities
Components	Core language, basic APIs	Core + enterprise APIs
Use Cases	Standalone applications	Enterprise, scalable, distributed applications
Technologies Included	JavaFX, JDBC, etc.	EJB, Servlets, JSP, JMS, JPA, CDI, etc.
Integration	Limited integration capabilities	Extensive support for various integration scenarios

J2EE OR JEE OR….

Version	Date
J2EE 1.2	December 1999
J2EE 1.3	September 2001
J2EE 1.4	November 2003
Java EE 5	May 2006
Java EE 6	December 2009
Java EE 7	April 2013
Java EE 8	August 2017
Jakarta EE	February 2018*

What are the differences between JVM vs JIT vs JDK vs JRE?

JVM	JIT	JDK	JRE
Java Virtual Machine: Introduced for managing system memory and also to provide a portable execution environment for Java applications.	Just in Time Compilation: This is a part of JVM and was developed for improving JVM performance.	Java Development Kit: JDK is a cross-platformed software development environment offering various collections of libraries and tools required for developing Java applications and applets.	Java Runtime Environment: JRE is part of JDK that consists of JVM, core classes and support libraries. It is used for providing a runtime environment for running Java programs.
Used for compiling byte code to machine code completely.	This is used for compiling only reusable byte code to machine code.	JDK is essential for writing and running programs in Java.	JRE is a subset of JDK and is like a container that consists of JVM, supporting libraries and other files. It doesn't have development tools such as compilers and debuggers.
This is used for providing platform independence.	This is used for improving the performance of JVM.	JDK is used for developing Java programs as it provides libraries and compiler tools as well as JRE for running the code.	JRE is not suitable for development. It is used for running Java Programs as it provides runtime environments and tools for supporting execution.

J2EE CONTAINER

- J2EE/Java EE applications aren't self contained. In order to be executed, they need to be deployed in a **container**.
- In other words, the container provides an execution environment on top of the JVM.
- The role of the **EE compliant container** is to provide a standard implementation of APIs like JPA, EJB, servlet, JMS, JNDI, etc.
- In J2EE specification the following are server-side containers:
- 1) **A Web container** that manages servlets and JavaServer Pages.
- 2) 1) **An EJB container** that manages EJB components

J2EE container :**Web Server**

- Web servers are computer programs that accept requests and returns responses based on that.

It constitutes the web container.

These are useful for getting static content for the applications.

This consumes and utilizes fewer resources.

Provide an environment for running web applications.

Web servers don't support multithreading.

This server makes use of HTML and HTTP protocols.

J2EE container :**Application Server**

- Application servers are used for installing, operating and hosting associated applications and services.

It constitutes both a web container and an EJB container.

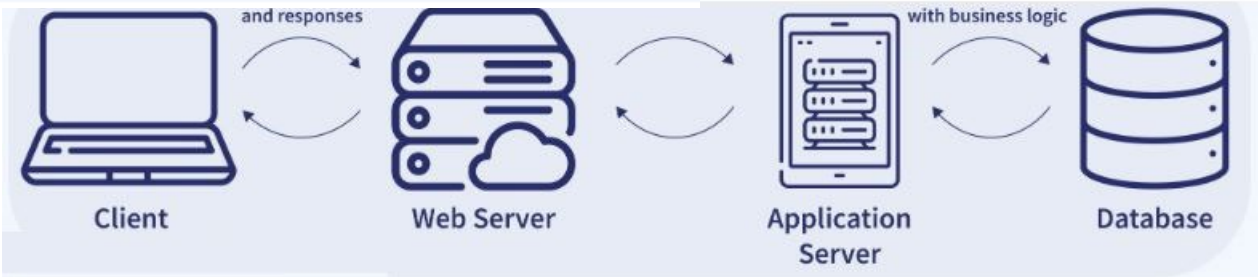
These are useful for getting dynamic content.

It makes use of more resources.

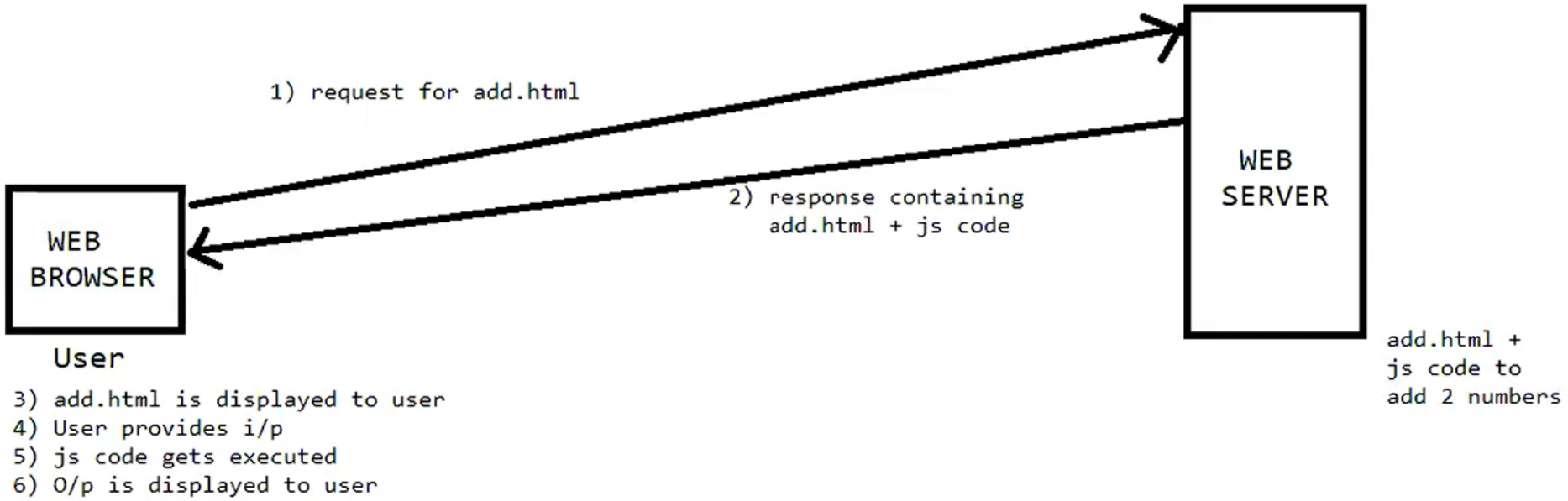
Provide an environment for running enterprise applications.

Multithreading is supported in application servers.

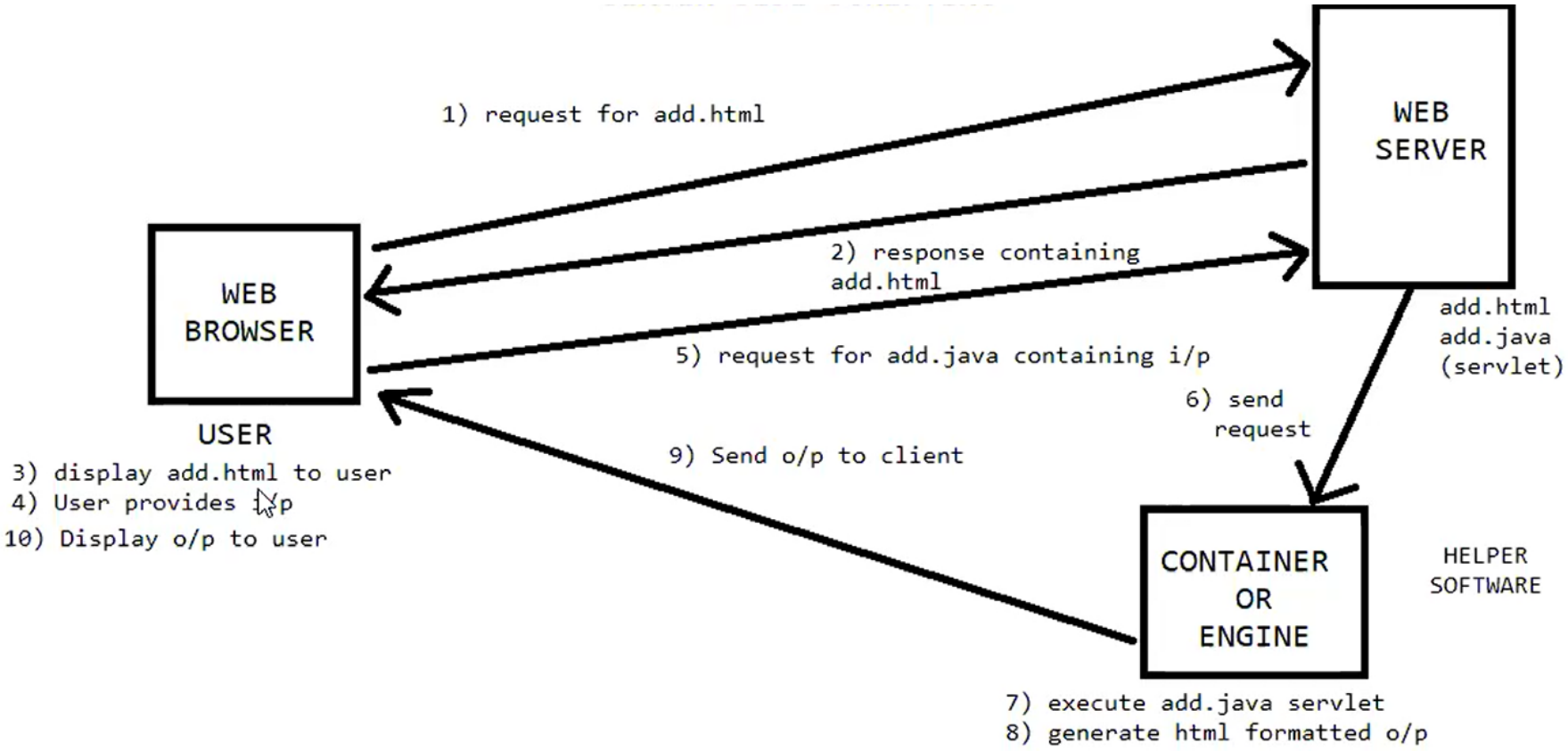
The application server has GUI (Graphical User Interface) and also supports HTTP, RPC/RMI protocols.



Scripting at client side



Scripting at server side



Web Application Life Cycle

- The process for creating, deploying, and executing a web application can be summarized as follows:

1. Develop the web component code.
2. Develop the web application deployment descriptor, if necessary.
3. Compile the web application components and helper classes referenced by the components.
4. Optionally, package the application into a deployable unit.
5. Deploy the application into a web container.
6. Access a URL that references the web application

Java Servlets

What is a Webapp?

- A **web application** is a program that runs on a computer with a web server, while its users interact with it via a web browser or similar user agent.

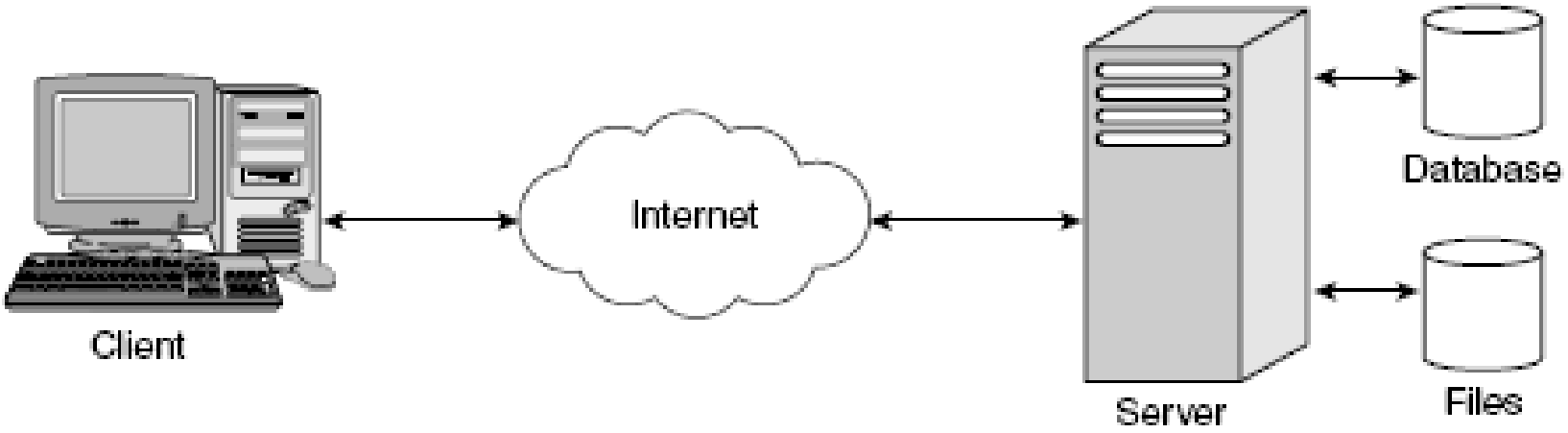
Example

- Gmail is a webapp. All users need is a web browser. They login, create and organize filters, read messages, reply, forward, send, and delete, and logout.
- Of course the "pages" include a fair number of scripts that the browser knows how to execute, but note that these scripts are kept on the server and downloaded on demand.

Introduction to Web

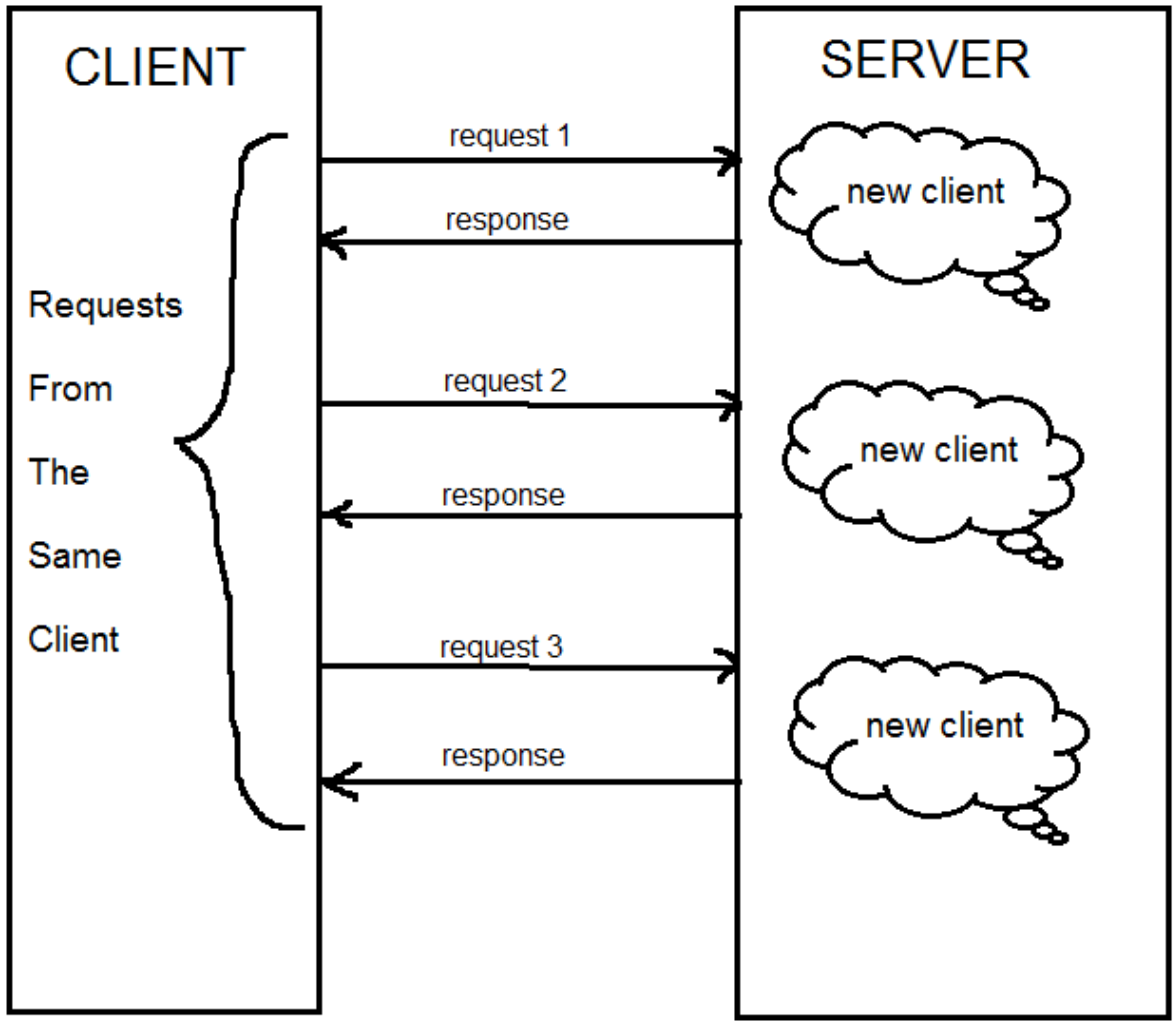
- Web consists of billions of clients and server connected through wires and wireless networks
- The web clients make requests to web server.
- The web server receives the request, finds the resources and return the response to the client.
- When a server answers a request, it usually sends some type of content to the client.
- The client uses web browser to send request to the server.
- The server often sends response to the browser with a set of instructions written in HTML(HyperText Markup Language).
- All browsers know how to display HTML page to the client.

Web Application



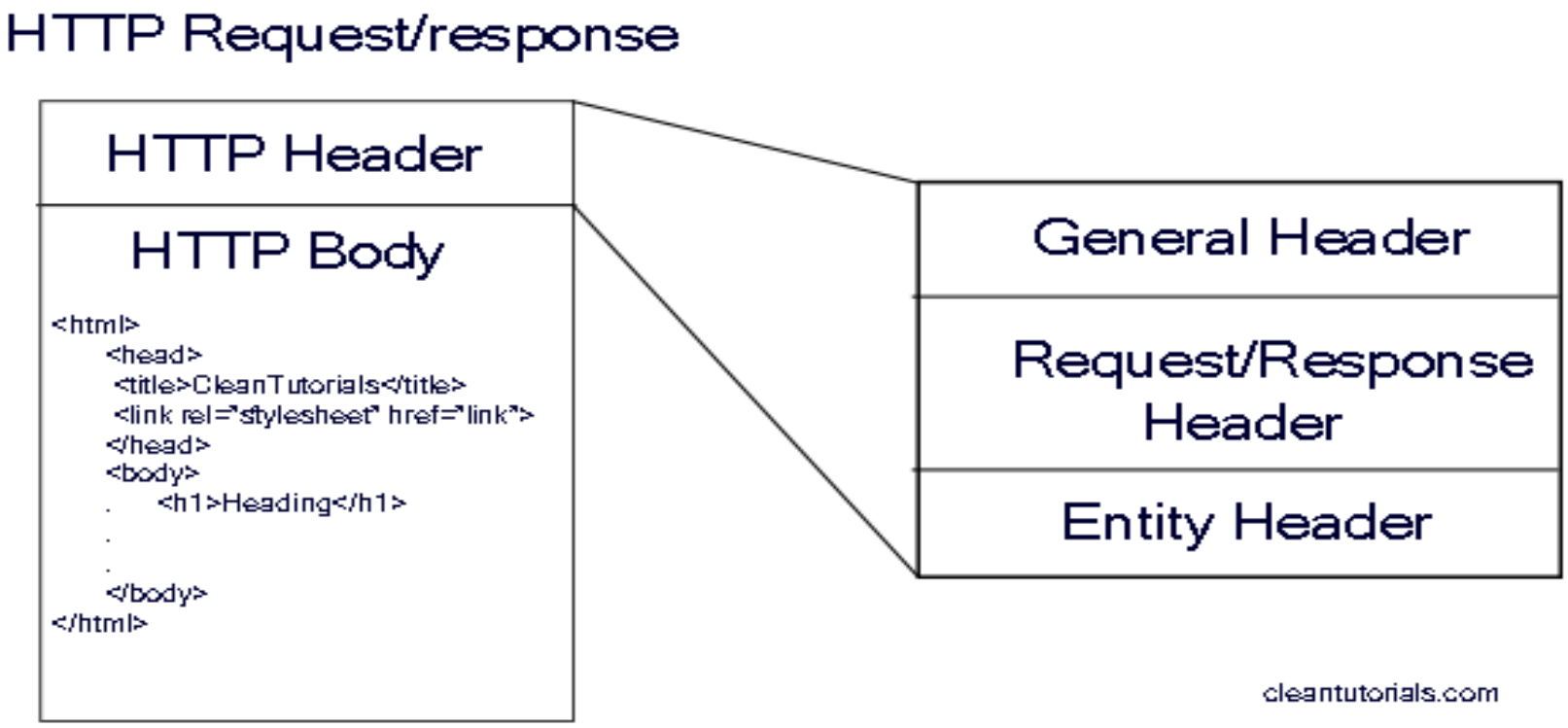
HTTP (Hypertext Transfer Protocol)

- HTTP is a protocol that clients and servers use on the web to communicate.
- It is similar to other internet protocols such as SMTP(Simple Mail Transfer Protocol) and FTP(File Transfer Protocol) but there is one fundamental difference.
- HTTP is a **stateless protocol** i.e. HTTP supports only one request per connection. This means that with HTTP the clients connect to the server to send one request and then disconnects. This mechanism allows more users to connect to a given server over a period of time.
- The client sends an HTTP request and the server answers with an HTML page to the client, using HTTP.


CLIENT
SERVER
request 1
response
Requests
From
The
Same
Client
request 2
response
request 3
response
new client
new client
new client

HTTP REQUEST FORMAT

- Both the request and response message have a common format, they both contain a HTTP Header and a HTTP Body.



HTTP Header

- HTTP Header contains information about the HTTP Body and the Request/Response.
- Information about the body is related to the content of the Body such as the length of the content inside the body.
- The information about Request/Response is the general information about the request/response like what time the Request was made or the browser used to make the request.
- The properties in header are specified as name-value pair which are separated from each other by a colon “:” .

What are the types of HTTP Headers?

1. General Header
2. Request/Response Header
3. Entity Header

General Header :

Contains general information about the communication such as the date and time on which the request/response was generated.

Request/Response Headers :

Request Header is present when you make a request to the server and the response header is present when the server sends a response back to the client/browser.

- ***Request Header*** contains information about the request such as the URL that you have requested, the method(GET, POST, HEAD) ,the browser used to generate the request and other info.

- ***Response Header*** contains information such as the encoding used in the content, the server software that is used on the server machine to generate the response and other information.

- **Entity headers**:

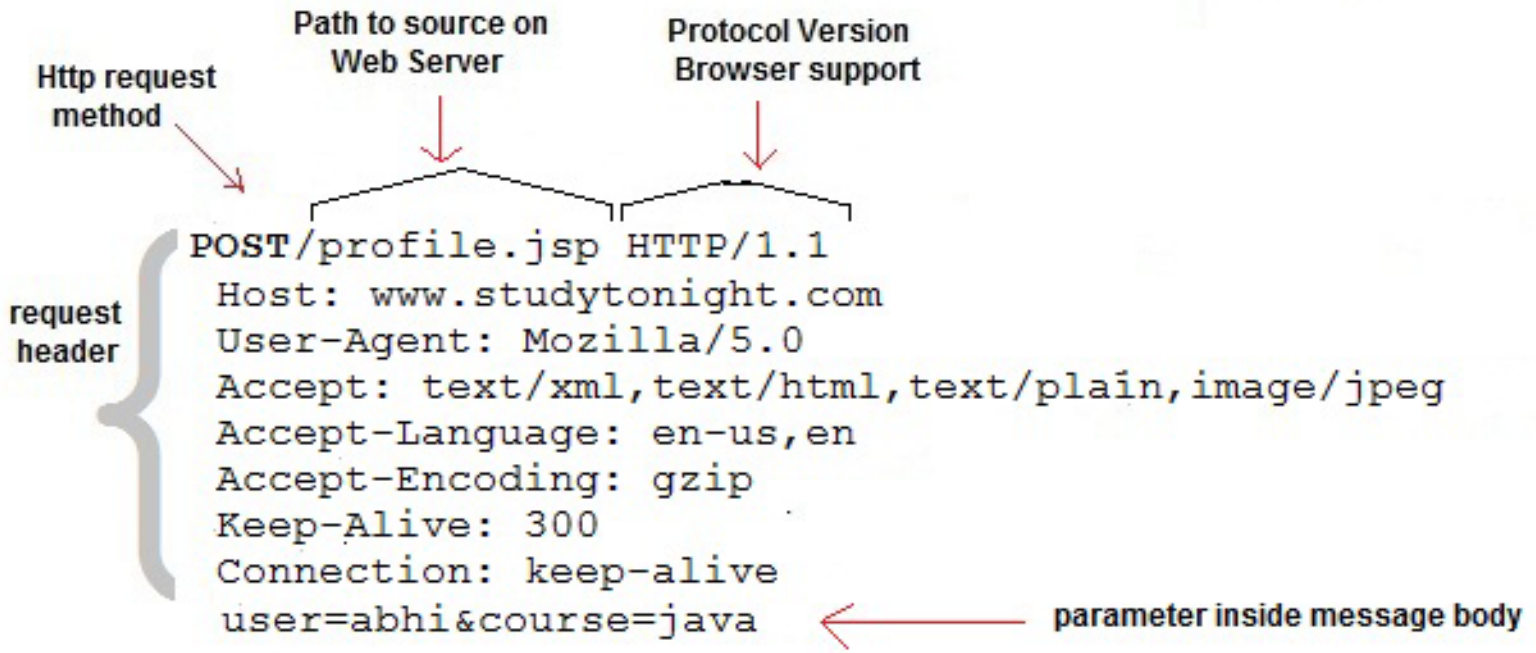
  contains information about the actual message or the HTTP body that is being sent. Information such as content length, the language of the content, encoding, expiration date and other important stuff.

HTTP BODY

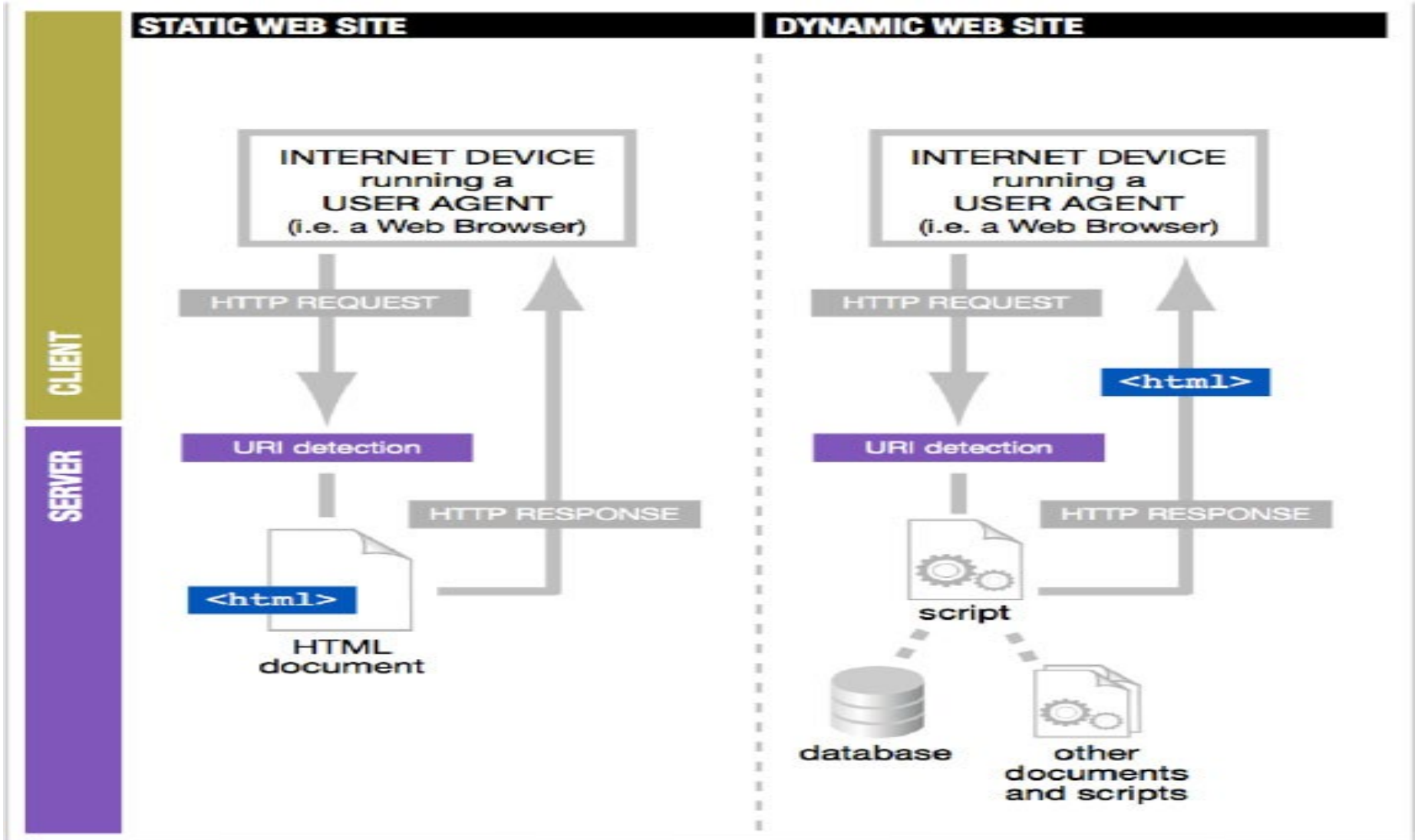
- Now comes the content/message body which is the actual data you want to fetch. The content can contain your HTML code or an image or your CSS stylesheets, JavaScript files depending on the resource for which you have made the request for.

HTTPRequest Sample

Http Response



Static Vs Dynamic Pages



CGI (Common Gateway Interface)

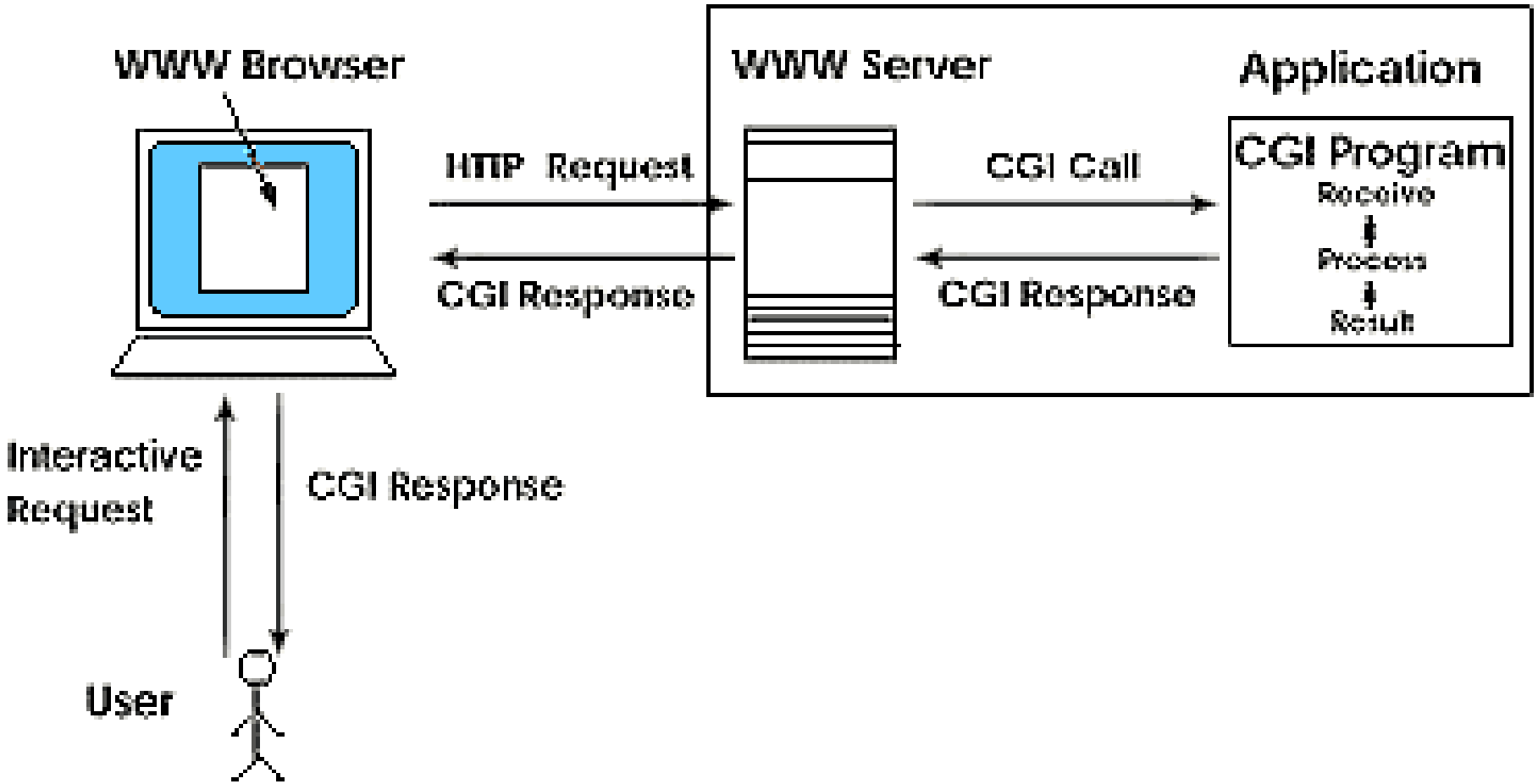
- Before Servlets, CGI(Common Gateway Interface) programming was used to create web applications.
- The common gateway interface (CGI) is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user.
- When a user fills out a form on a Web page and sends it in, it usually needs to be processed by an application program.
- The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message.
- This method or convention for passing data back and forth between the server and the application is called the common gateway interface (CGI). It is part of the Web's Hypertext Transfer Protocol (HTTP).

CGI (Common Gateway Interface)

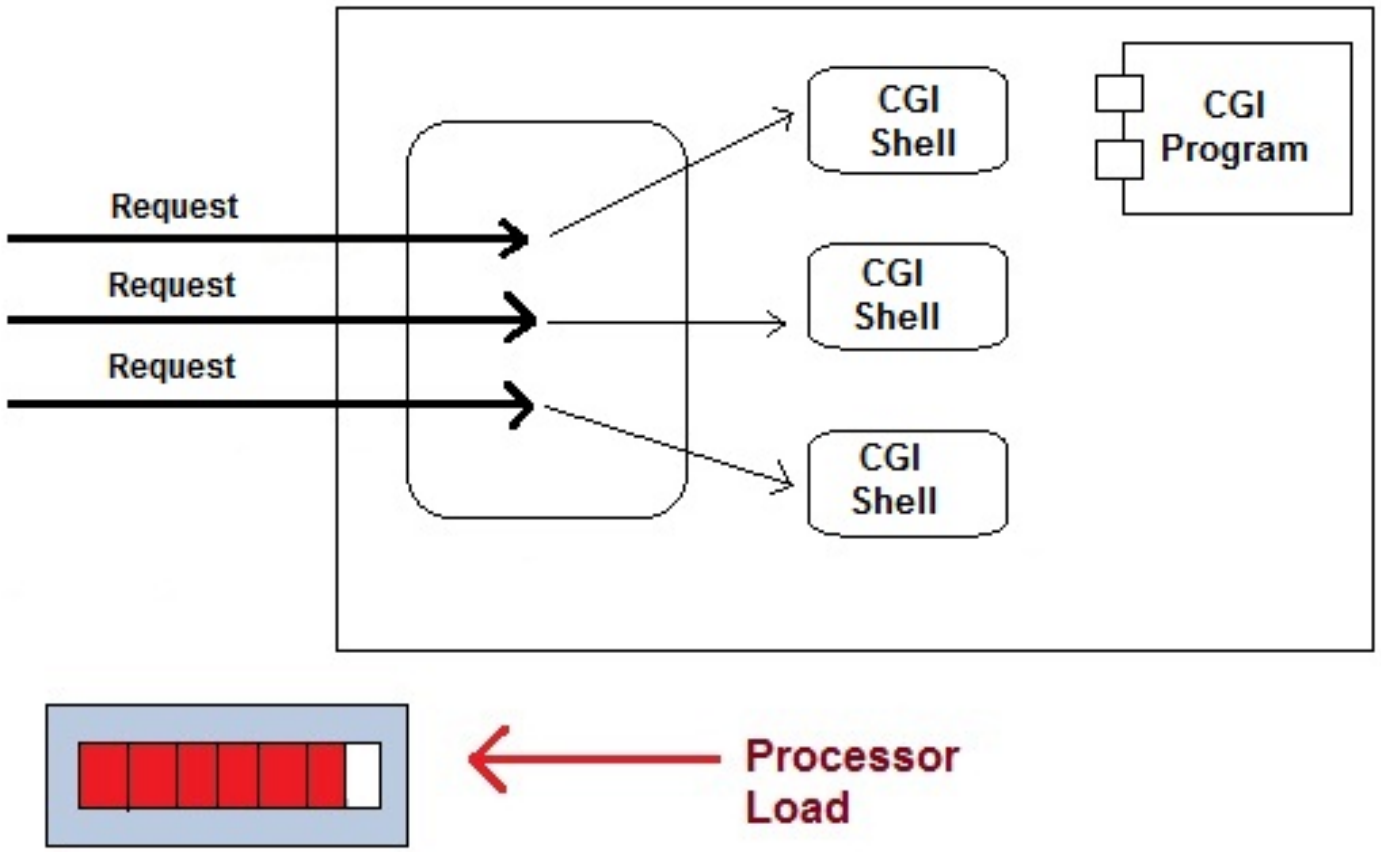
- Here's how a CGI program works :

  <FORM METHOD=POST ACTION=http://www.mybiz.com/cgi-bin/formprog.pl>

  - Locate the requested webpage(CGI application) using the request URL.
  - The URL decides which CGI program to execute.
  - Create a new process to service the client request.
  - Invoke the CGI application with in that process and passes the request information such as HTTP Header and data that the client has send as input.
  - Web Servers run the CGI program in seperate OS shell.
  - The shell includes OS enviroment and the process to execute code of the CGI program.
  - Collect the response from CGI application along with the appropriate headers
  - Destroy the process, and send HTTP response to the client.


WWW Browser
WWW Server
Application
HTTP Request
CGI Call
CGI Program
Receive
Process
Result
CGI Response
CGI Response
Interactive
Request
CGI Response
User

CGI


Request
Request
Request
CGI Shell
CGI Shell
CGI Shell
CGI Program
Processor Load

Drawbacks of CGI programs

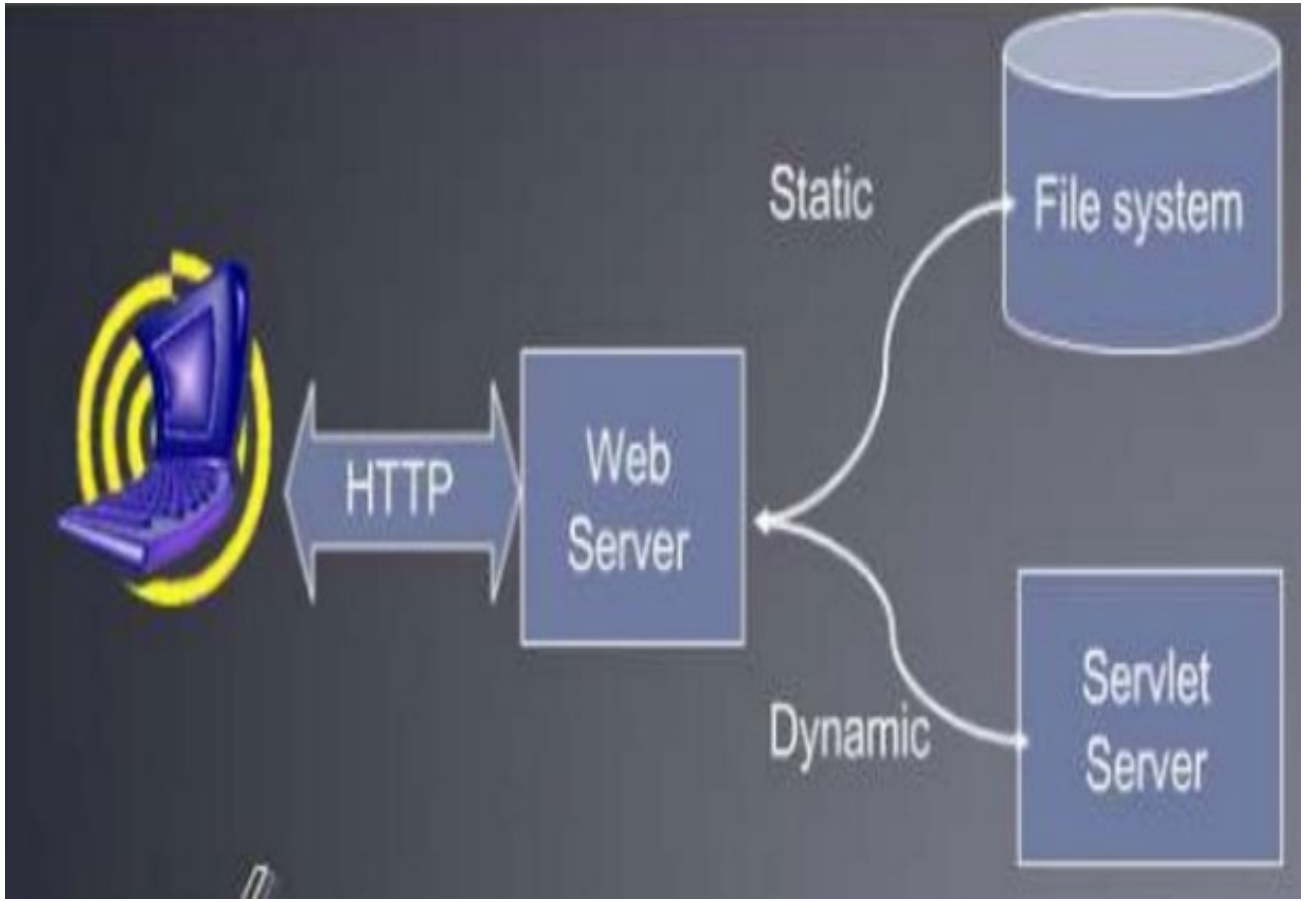
- High response time because CGI programs execute in their own OS shell.
- Cannot process large no of clients.
- It is process based.

Because of these disadvantages, developers started looking for better CGI solutions.

And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

Why Servlet?

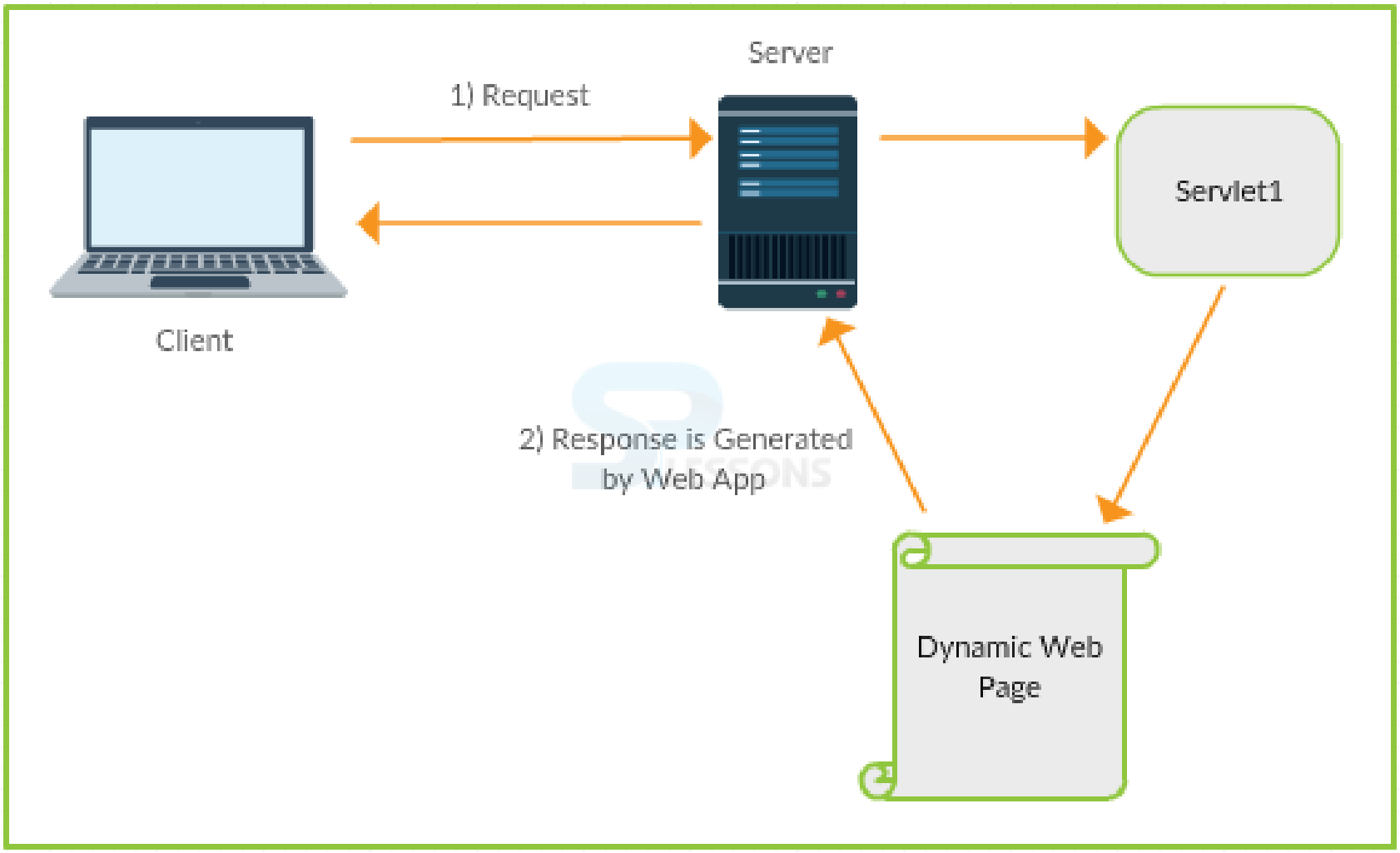
- Java Servlets are an efficient and powerful solution for creating dynamic content for the Web.



Introduction to Servlet

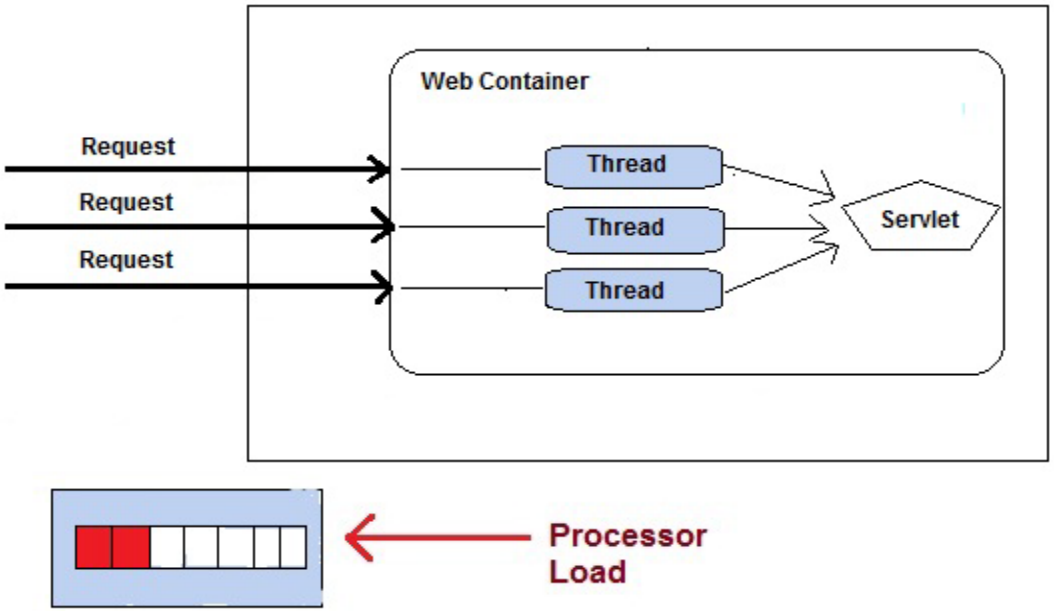
- A Java Servlet is a simple class.
- The Servlet object that generate dynamic content after processing the request originated from the web browser.
- Servlet Technology is used for server side extensions.
- The Serverside extensions are technologies to create dynamic web pages.
- Web pages need a container or web server to host the dynamic pages.
- To meet this requirement independent web servers provides solutions in form of API.
- These API allow us to build programs that can run with a web server.

Introduction to Servlet



Advantages of using Servlets

- Less response time because each request runs in a separate thread.
  - Faster than CGI.
  - Remove overhead of creating new process.
  - Servlets are platform independent.



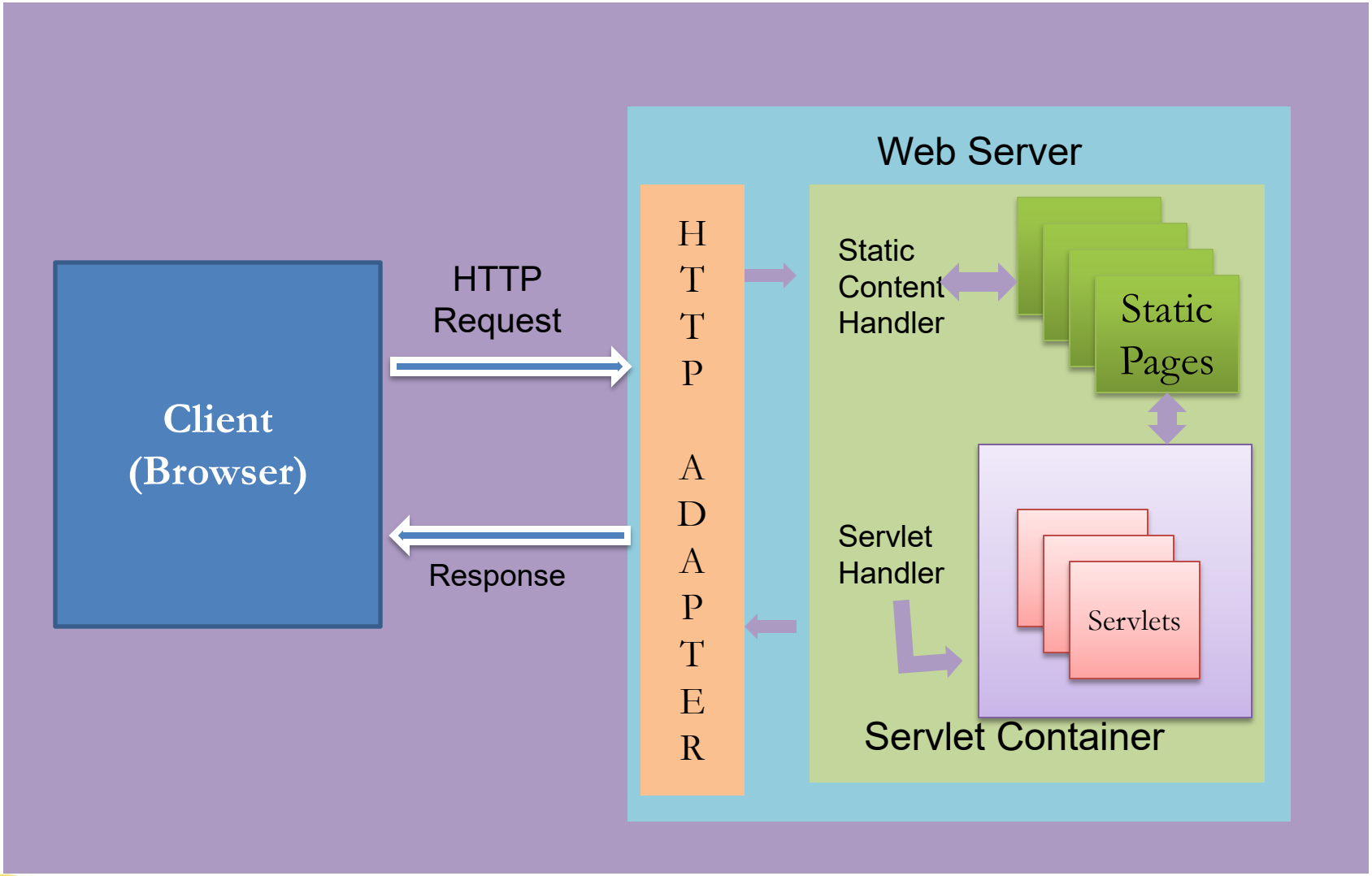
Servlet Container

- Servlet Container also known as Servlet Engine, is a set of objects that provides runtime environment for Servlets.

Functions of Servlet Container

  - Loads Servlet Class.
  - Provides the service of encoding and decoding MIME based messages.
  - Manages Life Cycle of Servlet.
  - Manages static and dynamic resources
  - Session Management.

Request Processing Mechanism



Request Processing Mechanism

- The web server receives a request and dispatches it to the Servlet container.
- The Servlet Container determines which Servlet to invoke on the basis of the request URL.
- The Servlet container calls the servlet whose request URL pattern matches with the pattern provided in the web.xml configuration file.
- The servlet uses the request object to determine the remote user and retrieves the parameters values sent as part of this request.

Request Processing Mechanism

- The servlet then generate the data to send response back to the client.
- It sends this data to the web server through the response object.
- Servlet uses the HTTPRequest and HTTP Response interface to manage user request and response.
- After the servlet has finished processing the request, The servlet container ensures that the response is properly flushed and returns the control back to the host web server.
- The web server packs the response content into the response message and sends it back to the client.

Servlet API

- Servlet API is Supported by the Servlet containers like Tomcat and weblogic.
- Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

- **javax.servlet**

  The javax.servlet package contains classes to support generic, protocol-independent servlets.

- **javax.servlet.http**

  javax.servlet.http package to add HTTP-specific functionality.

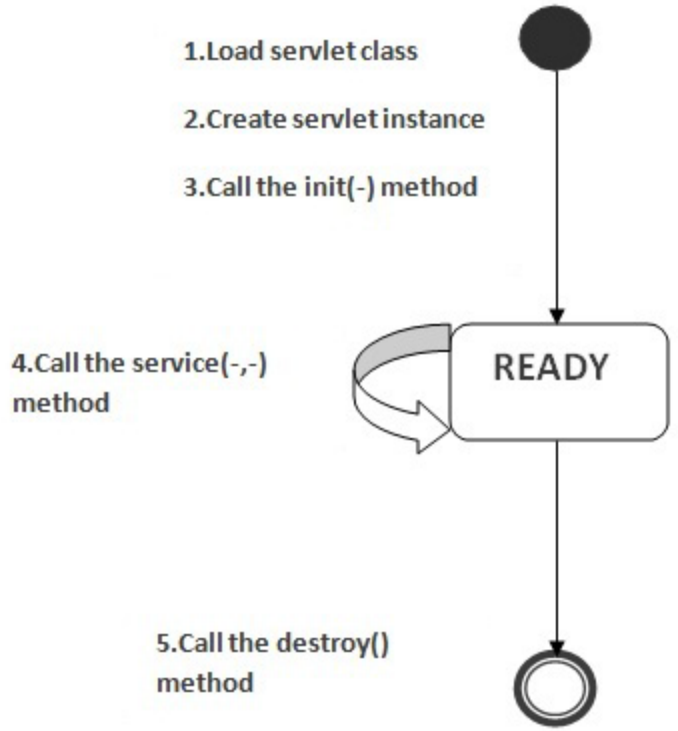
Servlet Interface

- Servlet Interface provides five methods.
- Out of these five methods, three methods are **Servlet life cycle** methods and rest two are non life cycle methods.

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,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	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.

Servlet Life Cycle

- The web container maintains the life cycle of a servlet instance.
  - Servlet class is loaded.
  - init method is invoked.
  - service method is invoked.
  - destroy method is invoked.



Step 1 :Loading Servlet

- First stage of servlet lifecycle is loading and initializing the servlet by container.
- The web container may delay this stage until the web container determines which servlet is needed to service a request.

- Loading:
  - Servlet container loads the servlet class as normal as loading java class.
  - Servlet class may be loaded from the local file system, remote file system or network services.
  - If the servlet container fails to load the class, it terminates this process.

Lifecycle

- Initialization:
  - After servlet loaded , the container creates an instance of servlet.

No. of Instance created by Servlet

If the servlet is not hosted in distributed environment and does not implement the SingleThreadModel interface, the container creates only one instance of the servlet declared inside the deployment descriptor.

If it is in an distributed environment , the servlet container create on servlet instance per JVM.

If it implements the SingleThreadModel , then also multiple instance of the servlet will be created.

Step 2 : Initializing Servlet

- After the servlet is instantiated successfully, container will initialize the servlet object.
- The container initializes the object b invoking the init() method, which accept *ServletConfig* object reference as parameter.
- This object allows the servlet instance to access name-value configured initialization parameters etc.
- Servlet container creates one *ServletConfig* object per servlet declaration in the deployment descriptor.
- If the container fails to initialize , it will throw Unavailable Exception or ServletException as a result.
- If init() method throws such exception , Servlet container does not support the servlet instance does not put into active state, which means it removes/destroy the instance.

Step 3: Request Handling

- Container creates ServletRequest and ServletResponse objects.
- In case of Http request , creates HTTServletRequest and HTTPServletResponse object.
- Invokes Service() and pass request response object to it.
- While processing the request, the servlet may throw ServletException ,UnavilableException or IOException.
- If an exception thrown was
  - ServletException : Indicates that an error occurs while processing the request, therefore container cleanup the request and send error response to the client.
  - UnavialableException : Servlet is currently not available temporarily or permanently.

    if temporarily– return 503 SERVICE UNAVILABLE response

    if permenantly– calls destroy() and remove instance.

Step 4: Destroy Servlet

- When a servlet container decide to destroy the servlet do following
  - Allows to complete the thread with the servlet to finish their job. Meanwhile , it makes the instance unavailable to service new process.
  - After finishing job servlet container calls destroy() on servlet instance.
  - Now it is ready for garbage collection.

Lifecycle Example

Index.html

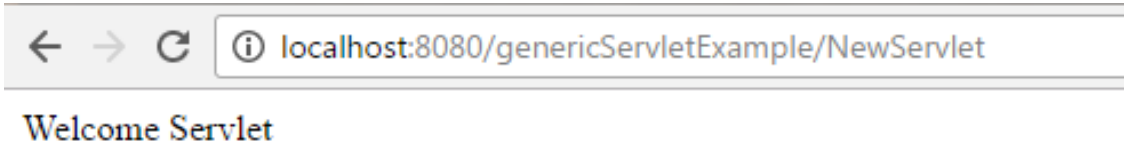
```
<html>
   <head>
      <title>TODO supply a title</title>
</head>
   <body>
      <form action="NewServlet" method="post">
      <input type="submit" value="SUBMIT"/>
      </form>
   </body>
</html>
```

NewServlet.java

```
public class NewServlet implements Servlet {

    PrintWriter out;
    ServletConfig config;

    @Override
    public void init(ServletConfig config) throws ServletException{
        this.config = config;
        System.out.println("Servlet Initialization Happens Here");
    }

    @Override
    public ServletConfig getServletConfig(){
        return config;
    }
```

NewServlet.java

```
@Override
  public void service(ServletRequest req, ServletResponse res) throws ServletException,
    IOException {
     out = res.getWriter();
     out.print("<body>Welcome Servlet</body>");
  }

  @Override
  public String getServletInfo() {
   return "copyright 2016-2017";
  }

  @Override
  public void destroy() {
    System.out.println("In destroy");
  }
}
```



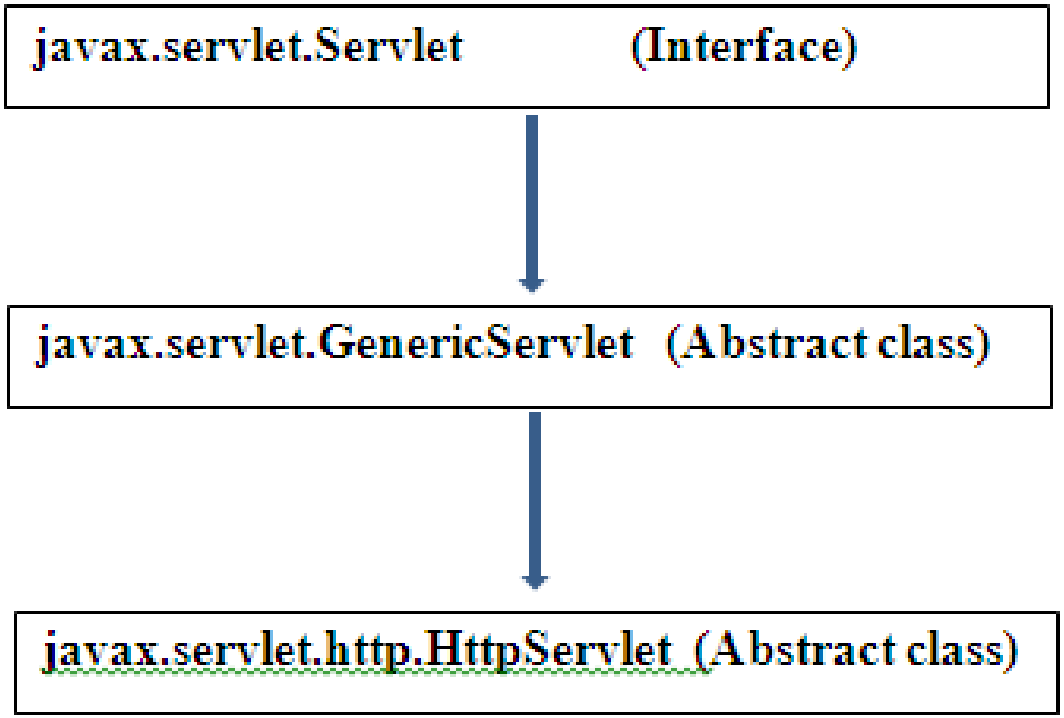
- Generally servlet implements *javax.servlet.Servlet interface* indirectly, by extending one of the below two classes

  javax.servlet.GenericServlet: Defines a generic, protocol-independent servlet

  javax.servlet.http.HttpServlet: It is an abstract class that needs to be extended(inherit) to create a HttpServlet suitable for a web application(website or online application).

Servlet doesn't have a main() method so certain methods (servlet's service() method) of servlet invoked by server.

Servlet Heirarchy



GenericServlet Working

- GenericServlet class has abstract service method. Which means the subclass of GenericServlet must have to override the service() method.

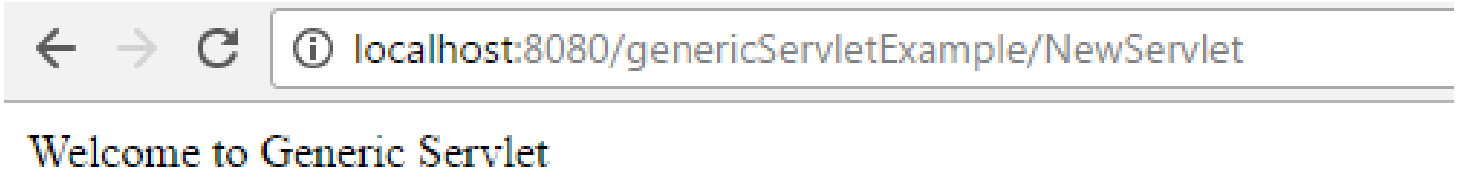
- Signature of service() method:

  public abstract void service(ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException

- As you can see here we have two arguments request and response. Object request receives the request from client and response sends the response back to client.

Lifecycle example using GenericServlet

```
public class NewServlet extends GenericServlet {
    @Override
    public void service(ServletRequest req, ServletResponse res) throws
     ServletException, IOException {
        PrintWriter out = res.getWriter();
        out.println("<body>Welcome to Generic Servlet</body>");
    }
}
```



HTTPServlet Working

- It is an abstract class. A subclass of HttpServlet must override at least one method, usually one of these:

  doGet, if the servlet supports HTTP GET requests

  doPost, for HTTP POST requests

- init and destroy, to manage resources that are held for the life of the servlet.
- getServletInfo, which the servlet uses to provide information about itself.
- There's almost no reason to override the service method. service handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (doXXX methods)

- **Flow for GET request:**
  Web Server made a GET request -> HTTP Servlet receives the request ->it calls the overridden doGet() method -> Sends response back to WebServer.

- **Flow for POST request:**
  Web Server makes a POST request -> HTTP Servlet receives the POST request -> it calls the overridden doPost() method -> Sends response back to Server.

```java
public class NewServlet extends HttpServlet {
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample code. */
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet NewServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Welcome Http Servlet</h1>");
            out.println("</body>");
            out.println("</html>");
        }
    }
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }
```

deployment descriptor (web.xml file)

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

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javae
    <servlet>
        <servlet-name>NewServlet</servlet-name>
        <servlet-class>NewServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>NewServlet</servlet-name>
        <url-pattern>/NewServlet</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

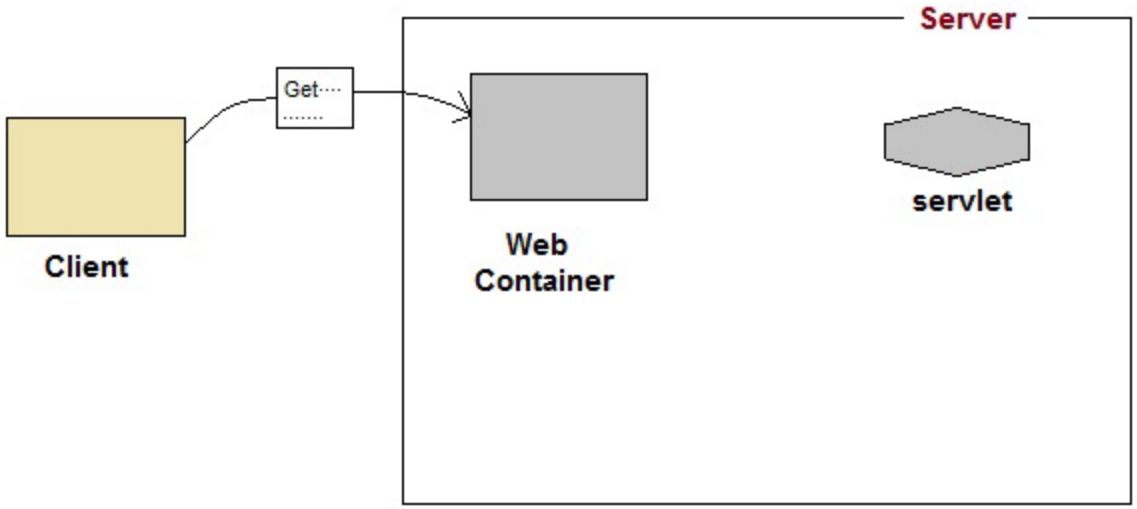
Description of the elements of web.xml file

- 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.

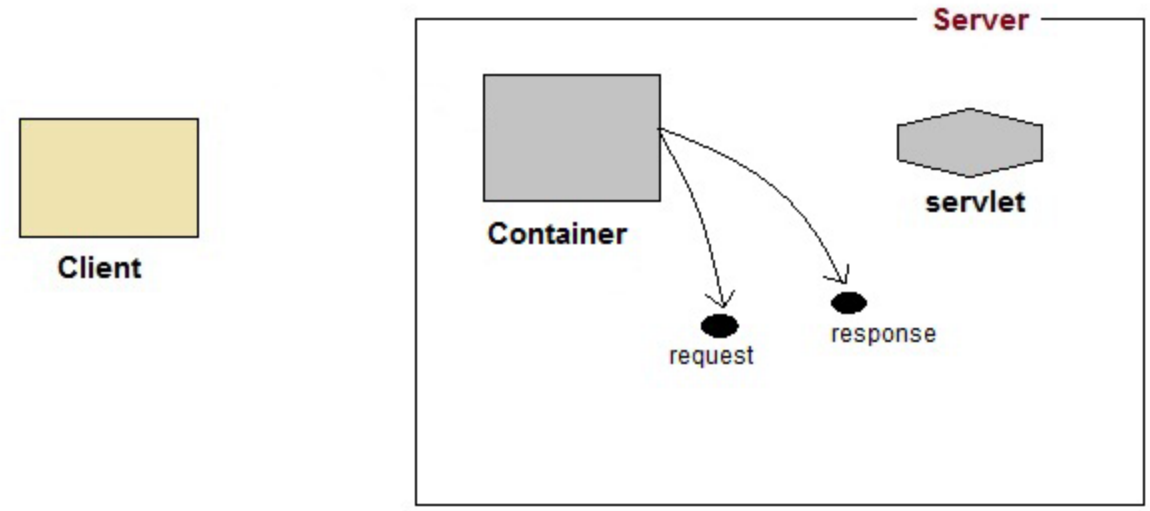
Quick Revision on How a Servlet works

1. User sends request for a servlet by clicking a link that has URL to a servlet.

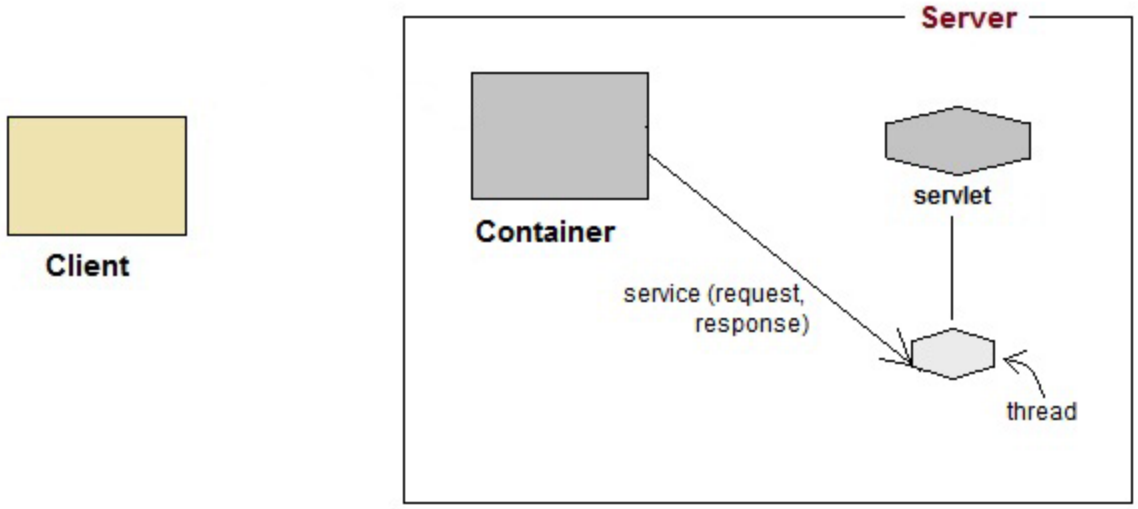


2. The container finds the servlet using **deployment descriptor** and creates two objects :

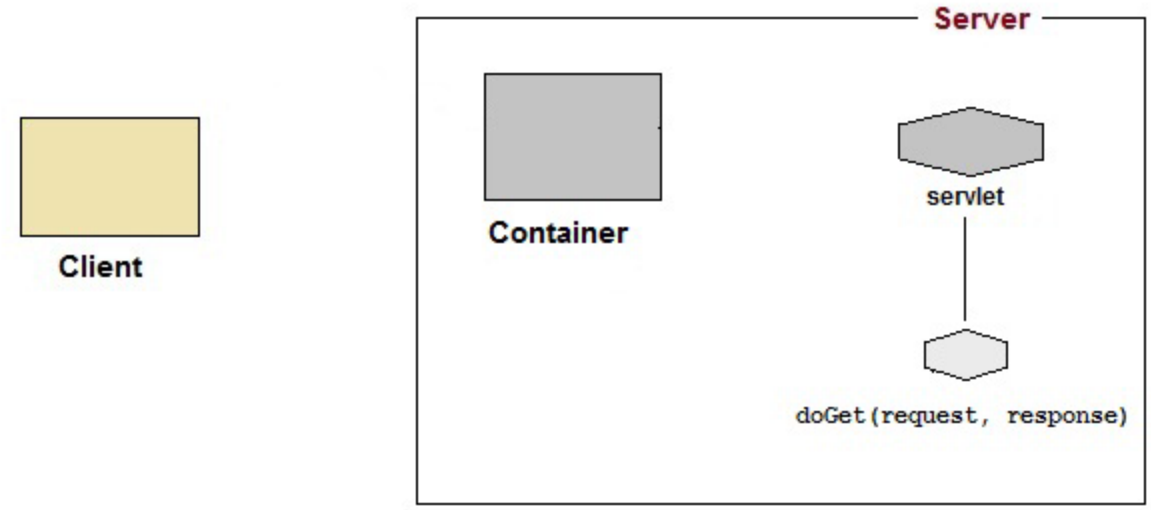
- **HttpServletRequest**
- **HttpServletResponse**



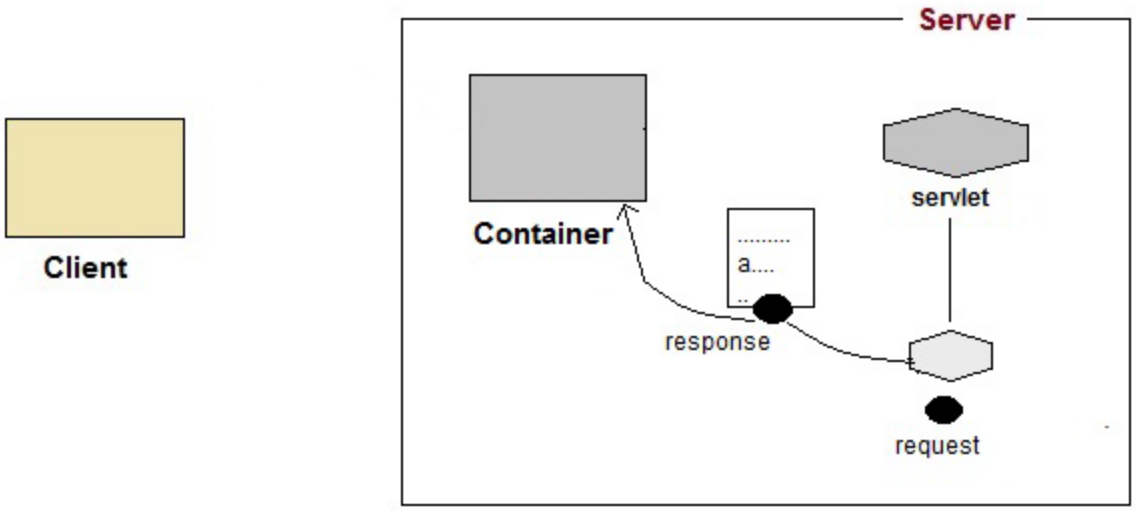
3. Then the container creates or allocates a thread for that request and calls the Servlet's service()method and passes the **request, response** objects as arguments.



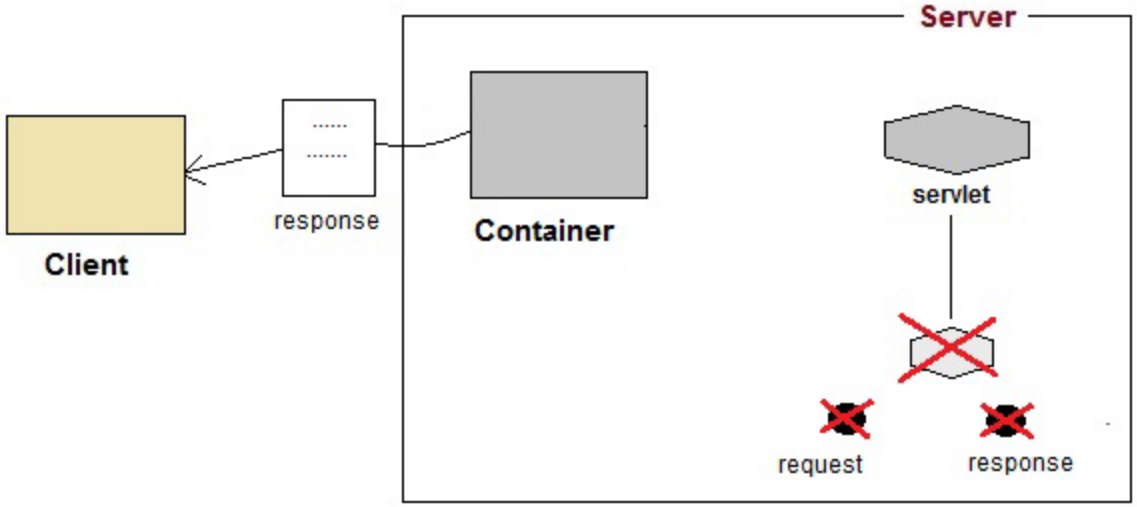
4. The service() method, then decides which servlet method, doGet() or doPost() to call, based on**HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.



5. Then the Servlet uses response object to write the response back to the client.



6. After the service() method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.




thank
you