Servlets

Introduction

A servlet is a Jakarta technology-based web component, managed by a container, that generates dynamic content.

Current Version 4.0

What are Servlets?

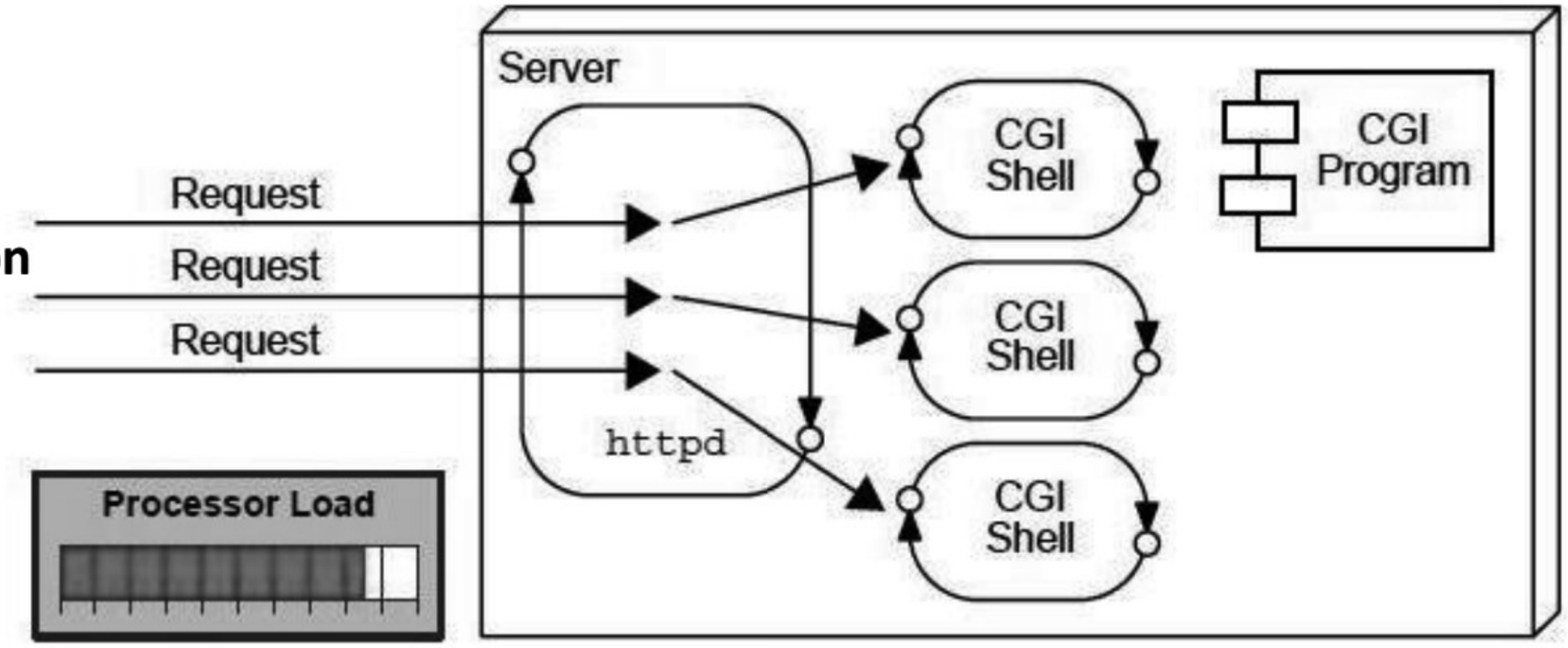
- Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.
- Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI).

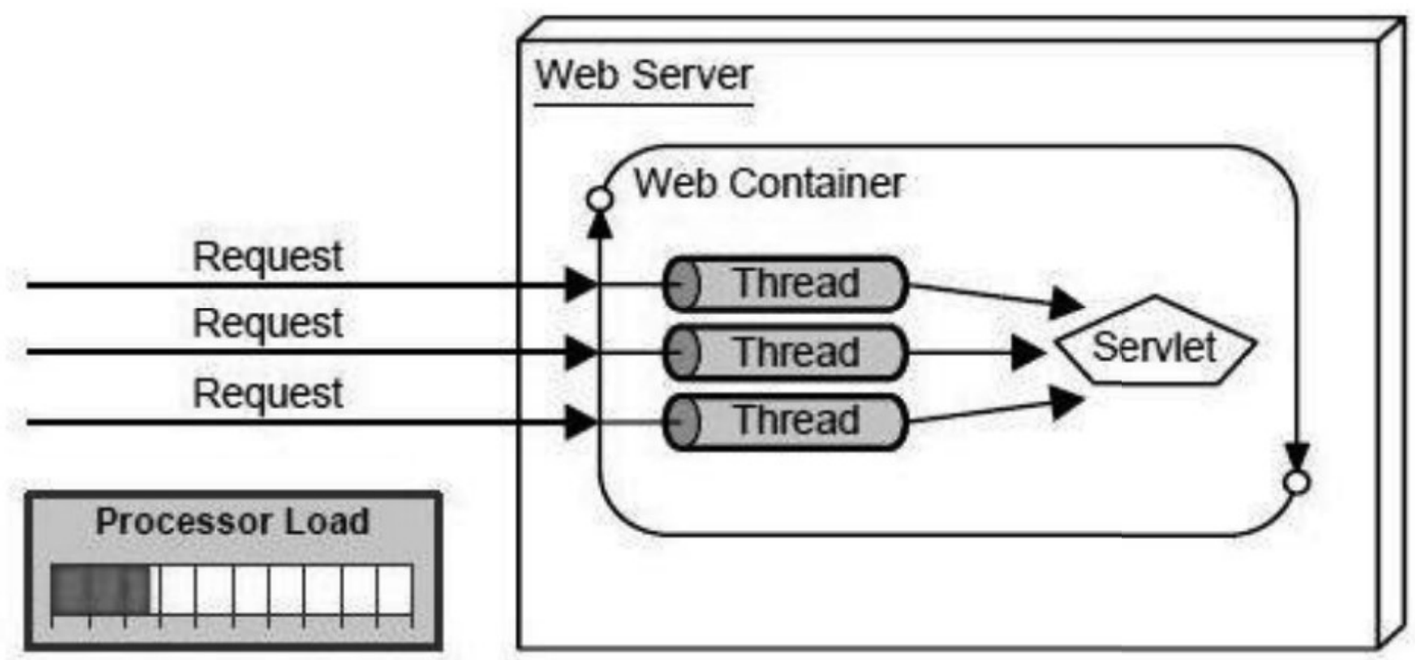
Advantages

Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

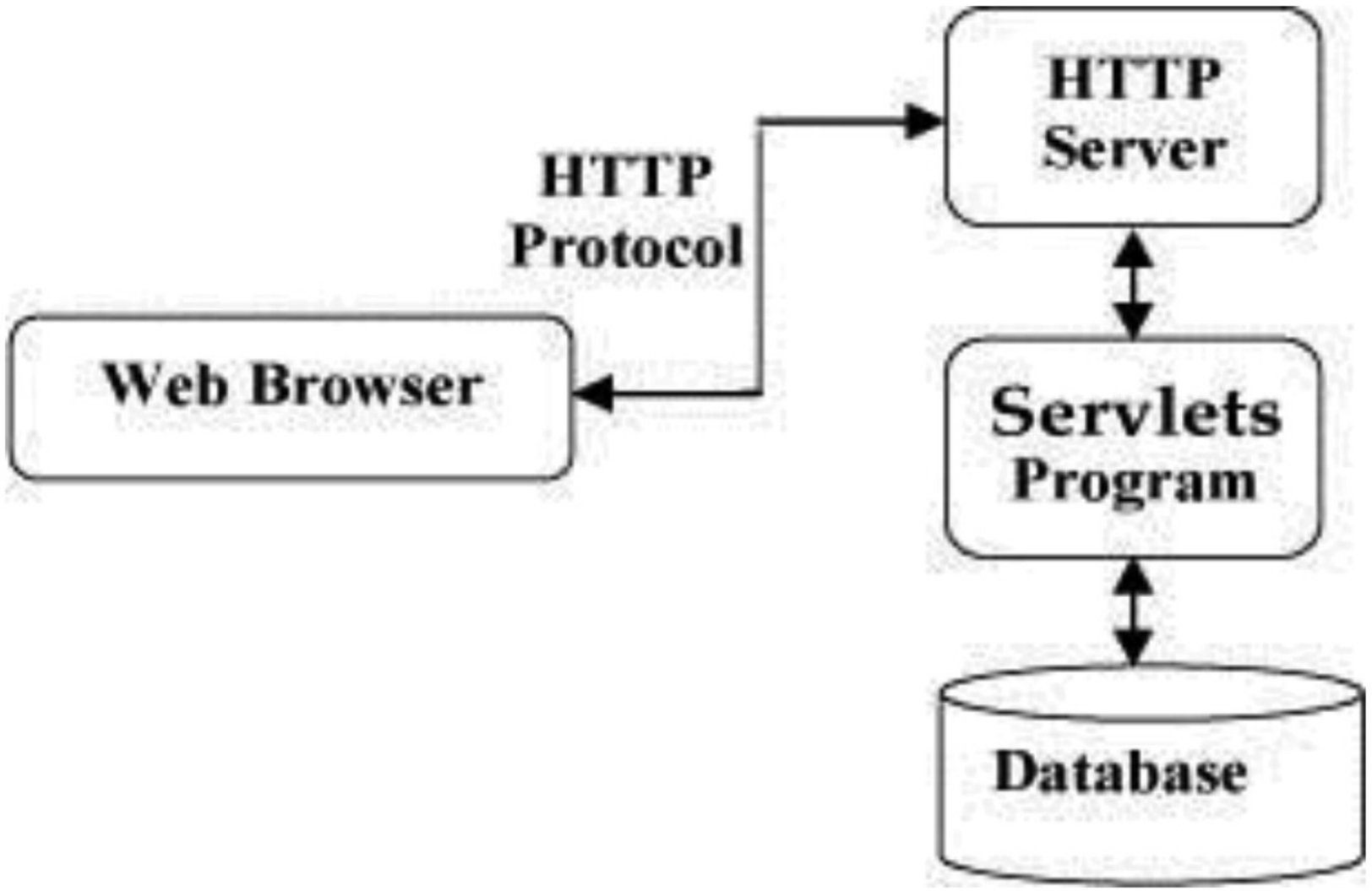
CGI Applicatiion





Servlet Application

Servlets Architecture



What is a Servlet Container?

- The servlet container is a part of a web server or application server that provides the network services over which requests and responses are sent.

Servlets Tasks

Servlets perform the following major tasks:

- ***Read the explicit data sent by the clients*** (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- ***Read the implicit HTTP request data sent by the clients*** (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- ***Process the data and generate the results***. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- ***Send the explicit data (i.e., the document) to the clients*** (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- ***Send the implicit HTTP response to the clients*** (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Packages

- Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

- The **javax.servlet** and **javax.servlet.http** packages provide interfaces and classes for writing servlets.

- All servlets must implement the Servlet interface, which defines lifecycle methods.

- When implementing a generic service, you can use or extend the GenericServlet class provided with the Java Servlet API.

- The HttpServlet class provides methods, such as **doGet** and **doPost**, for handling HTTP-specific services.

Servlet Interface

- **Servlet interface** provides common behaviour to all the servlets.

- Servlet interface needs to be implemented for creating any servlet (either directly or indirectly).

- It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet.

Methods of Servlet interface

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,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.

GenericServlet class:

- GenericServlet class implements **Servlet**, **ServletConfig** and **Serializable** interfaces. It provides the implementaion of all the methods of these interfaces except the service method.
- GenericServlet class can handle any type of request so it is protocol-independent.
- You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method

HttpServlet class:

- The HttpServlet class extends the GenericServlet class and implements Serializable interface.
- HttpServlet defines a HTTP protocol specific servlet

Servlet LifeCycle

- The lifecycle of a servlet is controlled by the container in which the servlet has been deployed.
- When a request is mapped to a servlet, the container performs the following steps.

1. If an instance of the servlet does not exist, the web container
   - Loads the servlet class.
   - Creates an instance of the servlet class.
   - Initializes the servlet instance by calling the **init()** method.
2. Invokes the **service()** method, passing request and response objects.
3. If it needs to remove the servlet, the container finalizes the servlet by calling the servlet's **destroy()** method.

Init() method

- **Init():** is designed to be called only once. It is called when the servlet is first created, and not called again for each user request.

- When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate.

- The init() method simply creates or loads some data that will be used throughout the life of the servlet.

Service() method

- The **service**() method is the main method to perform the actual task.
- The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.
- Each time the server receives a request for a servlet, the server spawns a new thread and calls service.
- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

- **doGet() Method**

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

  // Servlet code

  }

- **doPost() Method**

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

  // Servlet code

  }

- **destroy() method:** The destroy() method is called only once at the end of the life cycle of a servlet.
  - This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities

  public void destroy() { // Finalization code... }

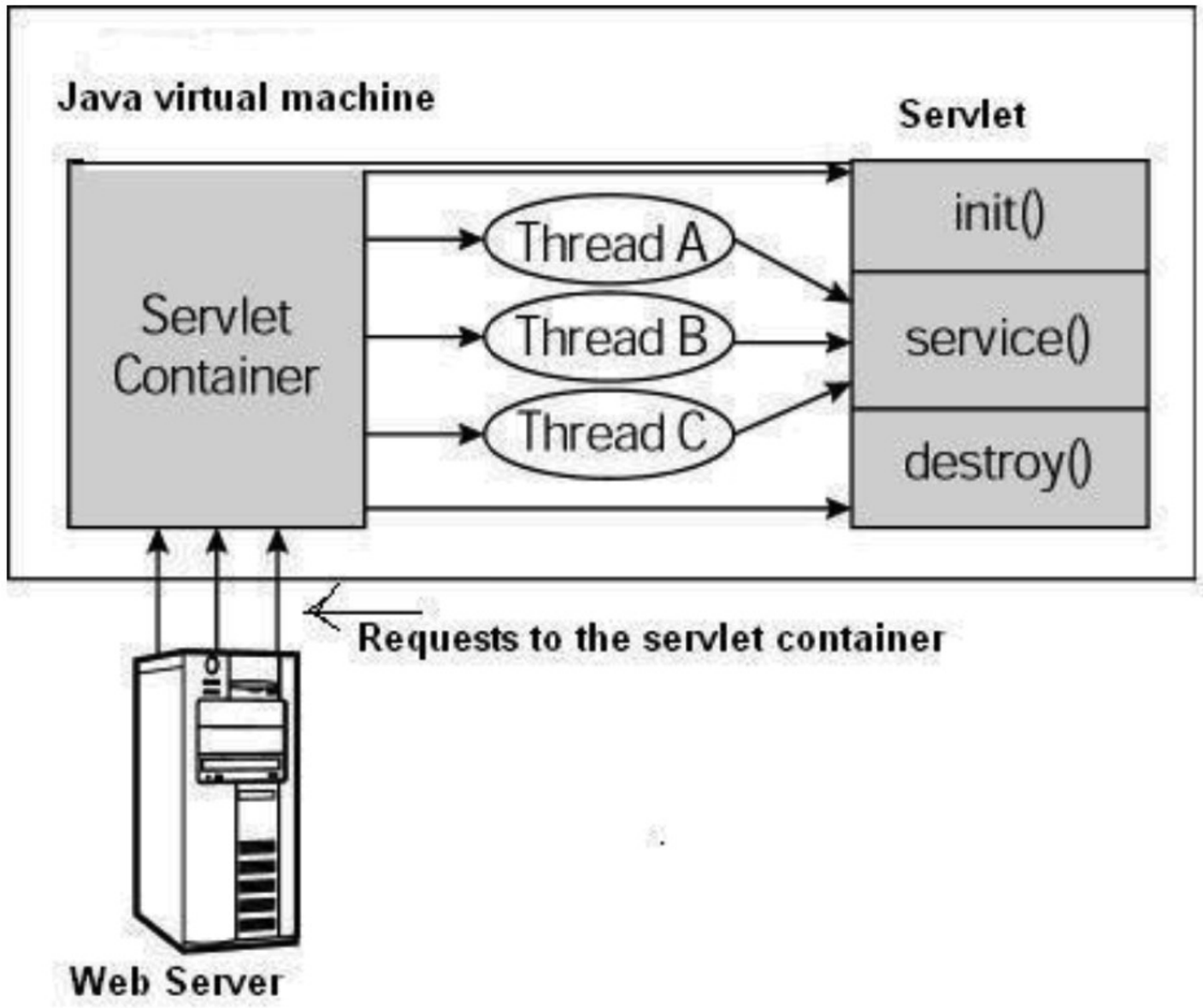
HTTP Specific Request Handling Methods

The HttpServlet abstract subclass adds additional methods beyond the basic *Servlet interface* that are automatically called by the service method in the HttpServlet class to aid in processing HTTP-based requests.

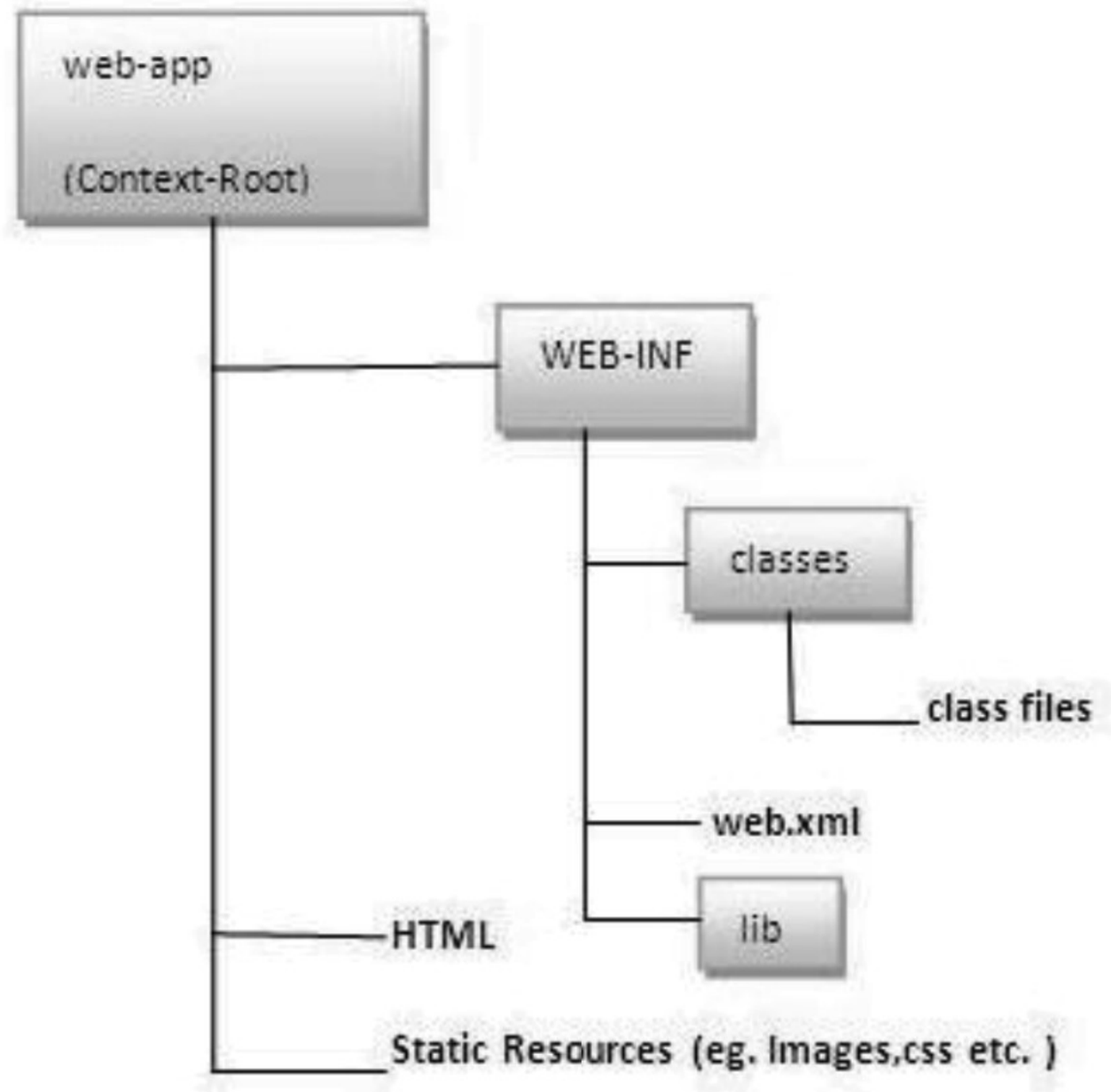
These methods are

- **doGet** for handling HTTP GET requests
- **doPost** for handling HTTP POST requests
- **doPut** for handling HTTP PUT requests
- **doDelete** for handling HTTP DELETE requests
- **doHead** for handling HTTP HEAD requests
- **doOptions** for handling HTTP OPTIONS requests
- **doTrace** for handling HTTP TRACE requests

1. First the HTTP requests coming to the server are delegated to the servlet container.
2. The servlet container loads the servlet before invoking the service() method.
3. Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet



Web application directory structure



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.

- **<web-app>**
- **<servlet>**
- **<servlet-name>**Hello**</servlet-name>**
- **<servlet-class>**DemoServlet**</servlet-class>**
- **</servlet>**
-
- **<servlet-mapping>**
- **<servlet-name>**Hello**</servlet-name>**
- **<url-pattern>**/welcome**</url-pattern>**
- **</servlet-mapping>**
- **</web-app>**

How web container handles the servlet request?

The web container is responsible to handle the request. Let's see how it handles the request.

- maps the request with the servlet in the web.xml file.
- creates request and response objects for this request
- calls the service method on the thread
- The public service method internally calls the protected service method
- The protected service method calls the doGet method depending on the type of request.
- The doGet method generates the response and it is passed to the client.
- After sending the response, the web container deletes the request and response objects. The thread is contained in the thread pool or deleted depends on the server implementation.

Get () Method

- The GET method sends the encoded user information appended to the page request.
- The page and the encoded information are separated by the ? character as follows:

Example: http://www.dummy.in/hello?key1=value1&key2=value2

  - The GET method is the defualt method to pass information from browser to web server and it produces a long string that appears in your browser.
  - Never use the GET method if you have password or other sensitive information to pass to the server.
  - The GET method has size limitation: only **1024** characters can be in a request string.

POST() Method

- A generally more reliable method of passing information to a backend program is the POST method.
- This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a ? in the URL it sends it as a separate message.
- This message comes to the backend program in the form of the standard input which you can parse and use for your processing.

GET() & POST()

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.

Reading Form data using Servlet

- Servlets handles form data parsing automatically using the following methods depending on the situation:
  - **getParameter():** You call request.getParameter() method to get the value of a form parameter.
  - **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
  - **getParameterNames():** Call this method if you want a complete list of all parameters in the current request.

Http Status Codes

100-199 Information

200-299 Successfully processed

300-399 Redirection

400-499 Problem with Client Request

500-599 Internal problem of server

Ex: 403-Access denied, 404-File not found

Client HTTP Request

Header	Description
Accept	This header specifies the MIME types that the browser or other clients can handle. Values of **image/png** or **image/jpeg** are the two most common possibilities.
Accept-Charset	This header specifies the character sets the browser can use to display the information. For example ISO-8859-1.
Accept-Encoding	This header specifies the types of encodings that the browser knows how to handle. Values of **gzip** or **compress** are the two most common possibilities.
Accept-Language	This header specifies the client's preferred languages in case the servlet can produce results in more than one language. For example en, en-us, ru, etc.
Authorization	This header is used by clients to identify themselves when accessing password-protected Web pages.
Connection	This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser to retrieve multiple files with a single request. A value of **Keep-Alive** means that persistent connections should be used
Content-Length	This header is applicable only to POST requests and gives the size of the POST data in bytes.

Client HTTP Request

Header	Description
Cookie	This header returns cookies to servers that previously sent them to the browser.
Host	This header specifies the host and port as given in the original URL.
If-Modified-Since	This header indicates that the client wants the page only if it has been changed after the specified date. The server sends a code, 304 which means **Not Modified** header if no newer result is available.
If-Unmodified-Since	This header is the reverse of If-Modified-Since; it specifies that the operation should succeed only if the document is older than the specified date.
Referer	This header indicates the URL of the referring Web page. For example, if you are at Web page 1 and click on a link to Web page 2, the URL of Web page 1 is included in the Referer header when the browser requests Web page 2.
User-Agent	This header identifies the browser or other client making the request and can be used to return different content to different types of browsers.

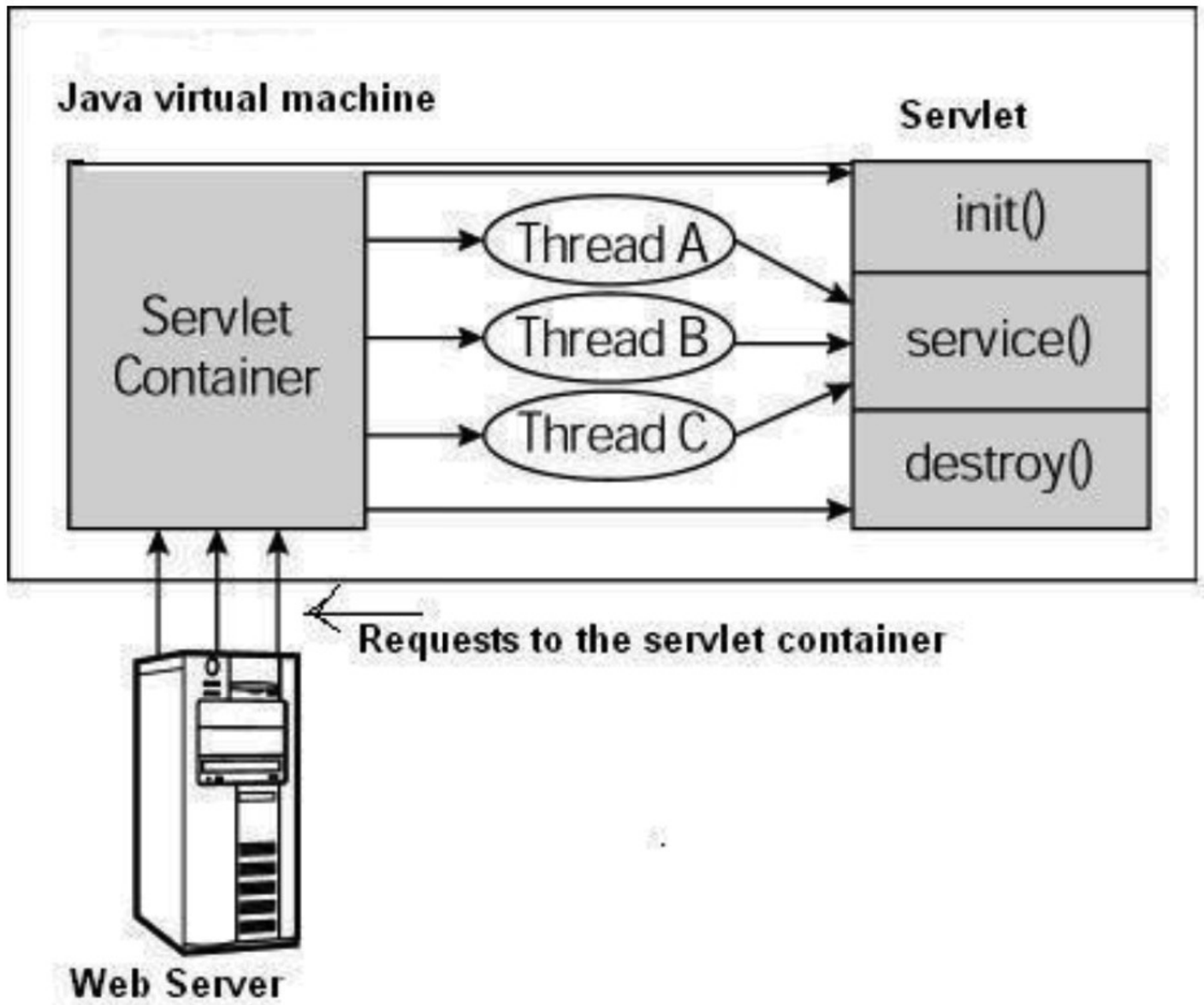
Server HTTP Response

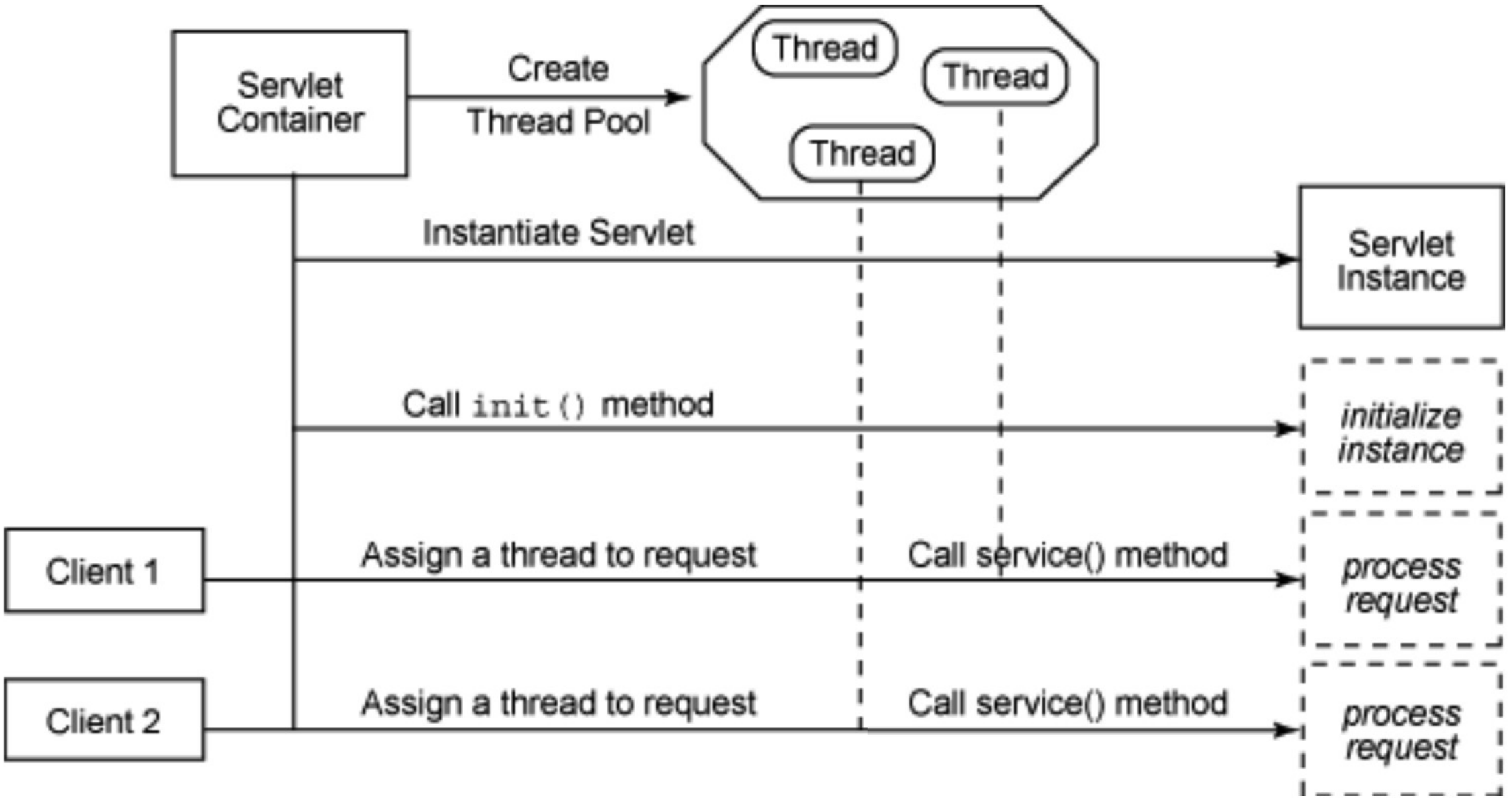
Header	Description
Allow	This header specifies the request methods (GET, POST, etc.) that the server supports.
Cache-Control	This header specifies the circumstances in which the response document can safely be cached. It can have values **public, private** or **no-cache** etc. Public means document is cacheable, Private means document is for a single user and can only be stored in private (nonshared) caches and no-cache means document should never be cached.
Connection	This header instructs the browser whether to use persistent in HTTP connections or not. A value of **close** instructs the browser not to use persistent HTTP connections and **keep-alive** means using persistent connections.
Content-Disposition	This header lets you request that the browser ask the user to save the response to disk in a file of the given name.
Content-Encoding	This header specifies the way in which the page was encoded during transmission.
Content-Language	This header signifies the language in which the document is written. For example en, en-us, ru, etc.

Server Header	Description
Content-Length	This header indicates the number of bytes in the response. This information is needed only if the browser is using a persistent (keep-alive) HTTP connection.
Content-Type	This header gives the MIME (Multipurpose Internet Mail Extension) type of the response document.
Expires	This header specifies the time at which the content should be considered out-of-date and thus no longer be cached.
Last-Modified	This header indicates when the document was last changed. The client can then cache the document and supply a date by an **If-Modified-Since** request header in later requests.
Location	This header should be included with all responses that have a status code in the 300s. This notifies the browser of the document address. The browser automatically reconnects to this location and retrieves the new document.
Refresh	This header specifies how soon the browser should ask for an updated page. You can specify time in number of seconds after which a page would be refreshed.
Retry-After	This header can be used in conjunction with a 503 (Service Unavailable) response to tell the client how soon it can repeat its request.
Set-Cookie	This header specifies a cookie associated with the page.

Part-2

1. First the HTTP requests coming to the server are delegated to the servlet container.
2. The servlet container loads the servlet before invoking the service() method.
3. Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet




Servlet Container
Create Thread Pool
Thread
Thread
Thread
Instantiate Servlet
Servlet Instance
Call init() method
initialize instance
Client 1
Assign a thread to request
Call service() method
process request
Client 2
Assign a thread to request
Call service() method
process request

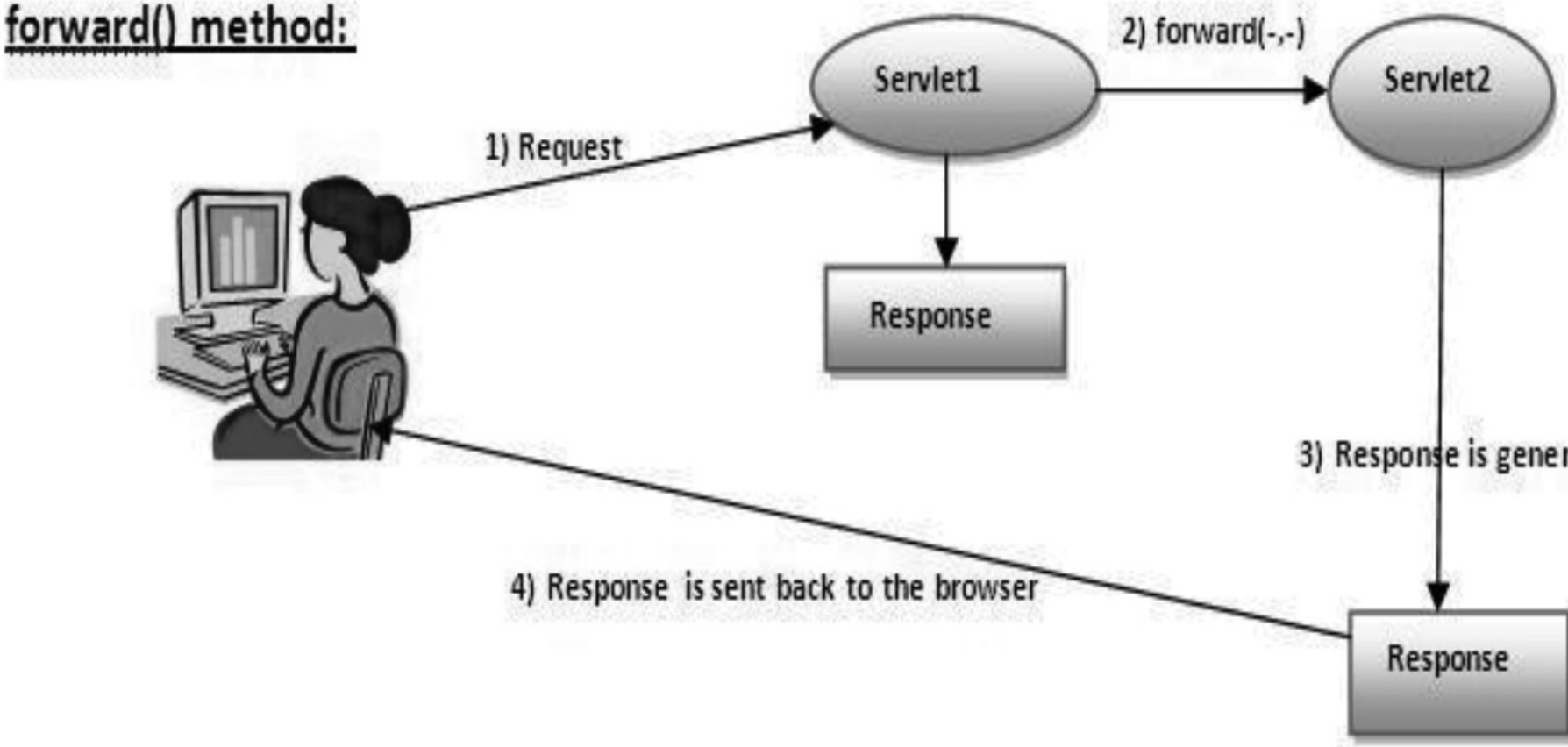
RequestDispatcher

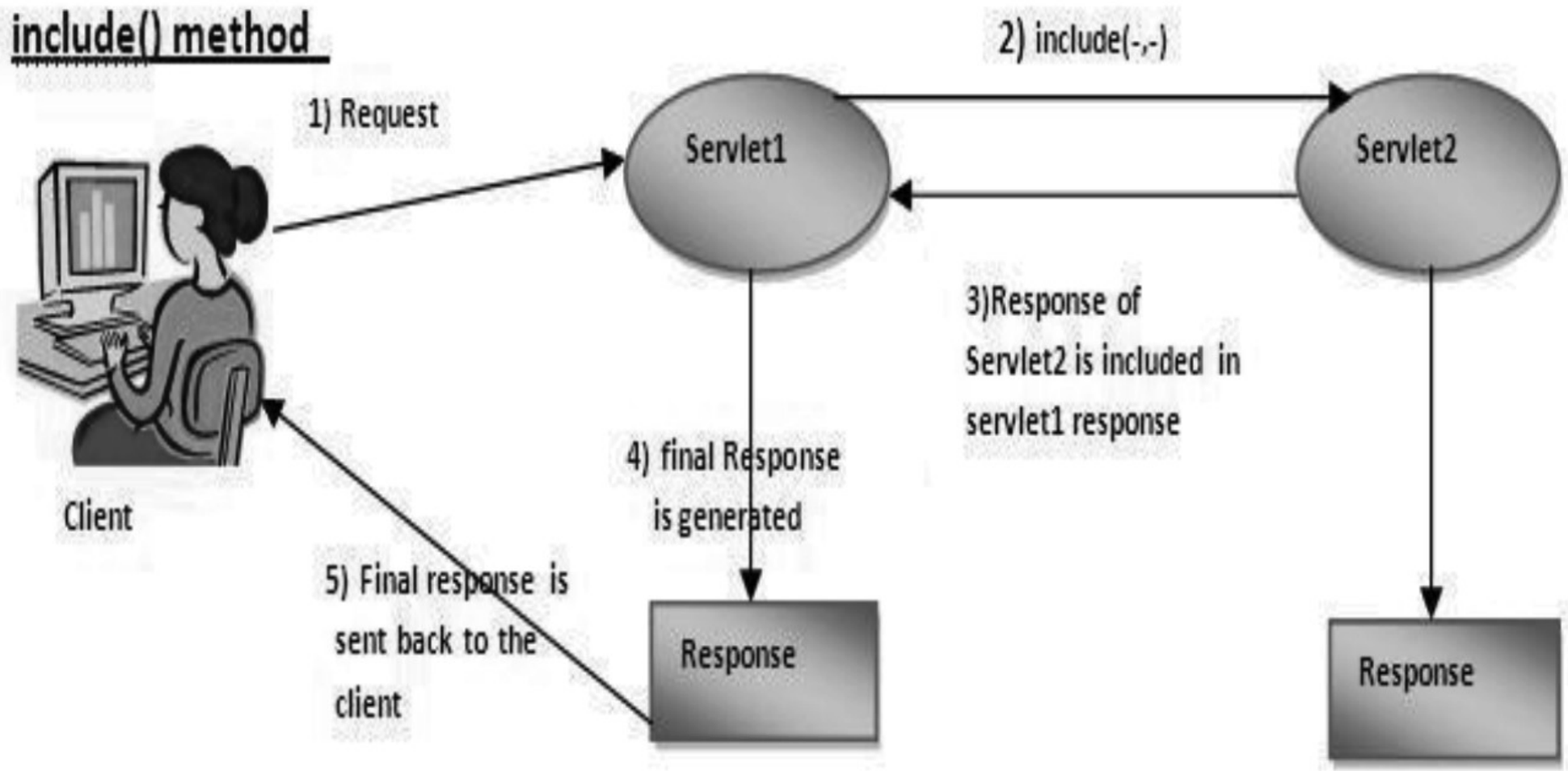
- The **RequestDispacher** interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp.
- This interface can also be used to include the content of another resource also.

The RequestDispatcher interface provides two methods. They are:

- ***public void forward(ServletRequest request,ServletResponse response)throws ServletException, java.io.IOException:*** Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
- ***public void include(ServletRequest request,ServletResponse response)throws ServletException, java.io.IOException:*** Includes the content of a resource (servlet, JSP page, or HTML file) in the response.

forward() method:




include() method
1) Request
2) include(-,-)
Servlet1
Servlet2
3)Response of
Servlet2 is included in
servlet1 response
Client
4) final Response
is generated
5) Final response is
sent back to the
client
Response
Response

SendRedirect in servlet

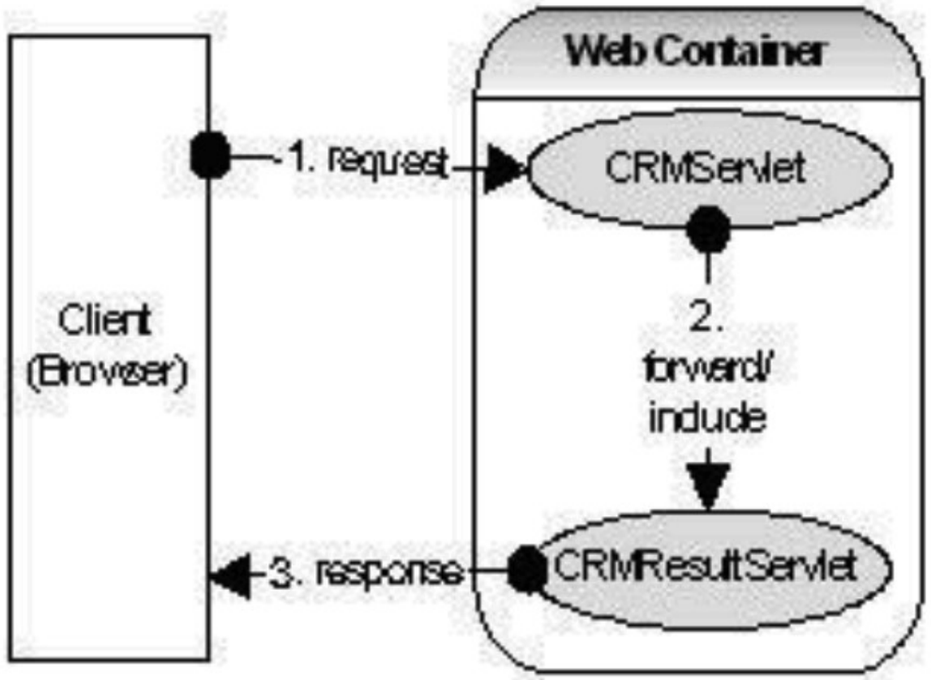
- The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.

- It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

forward() method	sendRedirect() method
The forward() method works at server side.	The sendRedirect() method works at client side.
It sends the same request and response objects to another servlet.	It always sends a new request.
It can work within the server only.	It can be used within and outside the server.
Example: request.getRequestDispacher("servlet2").forward(request,response);	Example: response.sendRedirect("servlet2");

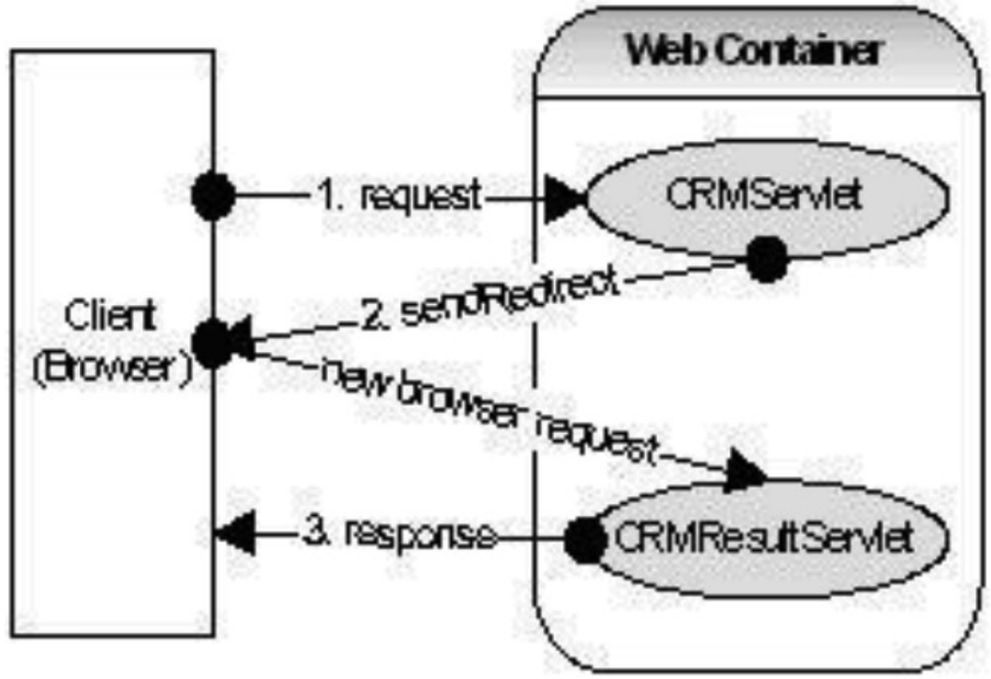
forward() or include() vs sendRedirect()

forward() or include()



Note: path supplied to RequestDispatcher will be something like "/CRMResultServlet"

sendRedirect()



Note: path supplied to RequestDispatcher will be something like "http://myserver:8080/myContext/CRMResultServlet".

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

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 servlets, 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 servlets. We provide this information from the web.xml file, so if the information is changed, we don't need to modify the servlet.

Example of getServletContext() method

//We can get the ServletContext object from ServletConfig object

- ServletContext application = getServletConfig().getServletContext();

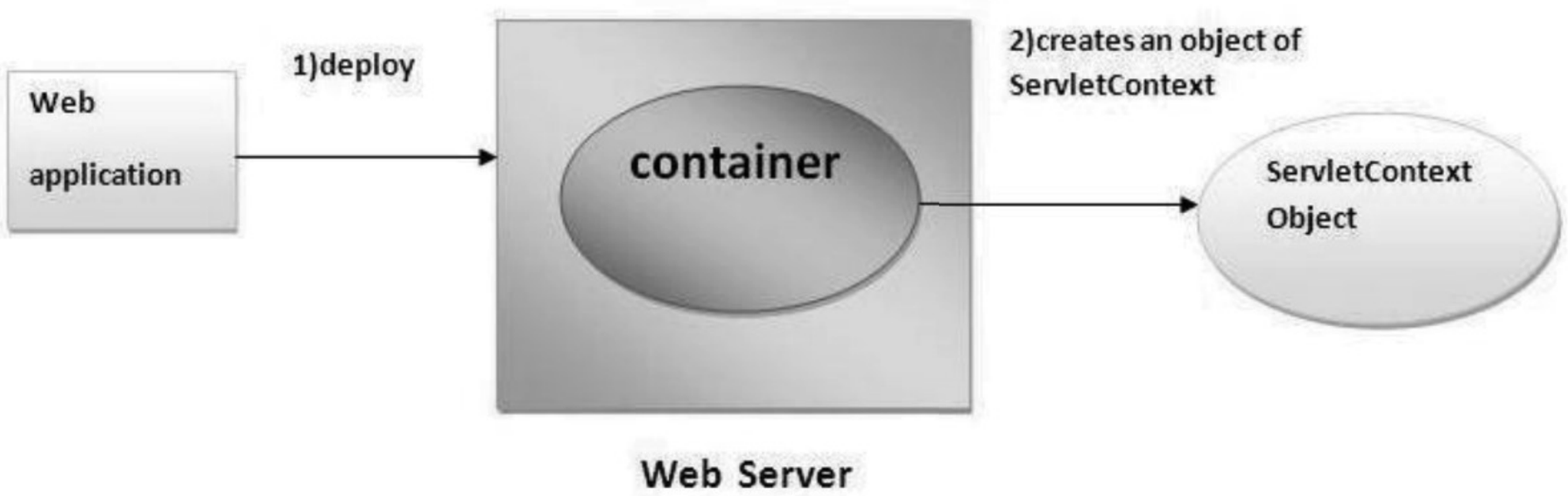
//Another convenient way to get the ServletContext object

- ServletContext application = getServletContext();

Usage of ServletContext

Usage of ServletContext Interface

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



Difference between ServletConfig and ServletContext

The **servletconfig** object refers to the single servlet whereas **servletcontext** object refers to the whole web application.

Scope

An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:

- request scope
- session scope
- application scope

- The servlet programmer can pass information from one servlet to another using attributes. It is just like passing object from one class to another so that we can reuse the same object again and again.

Session Tracking in Servlets

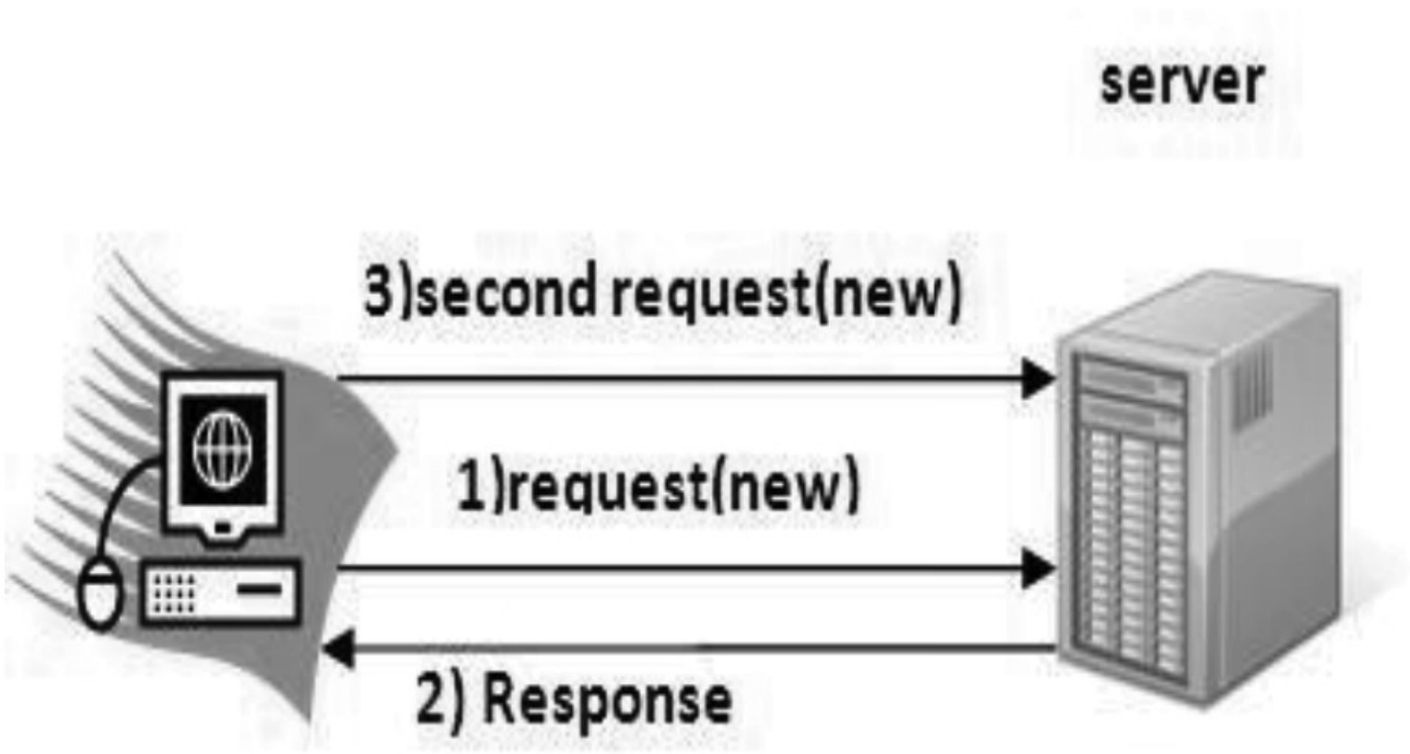
- **Session** simply means a particular interval of time.

- **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet.

- Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user.

Why use Session Tracking?

It is used to **recognize** the particular user.



Session Tracking Techniques

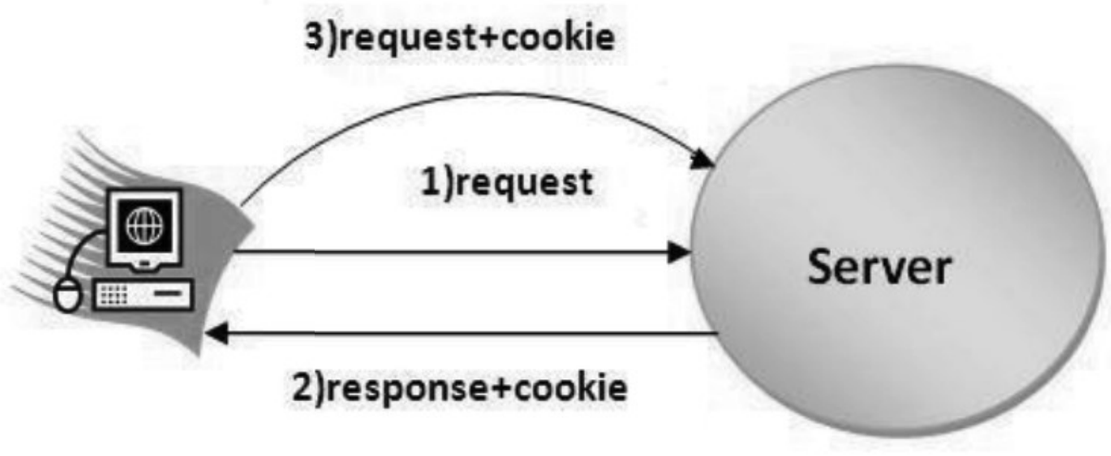
- There are four techniques used in Session tracking:
  - **Cookies**
  - **Hidden Form Field**
  - **URL Rewriting**
  - **HttpSession**

Cookies in Servlet

- A **cookie** is a kind of information that is stored at client side.
- 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



Cookies in Servlet

Types of Cookie

- There are 2 types of cookies in servlets.
  - Non-persistent cookie
  - 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

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

Disadvantage of Cookies

- It will not work if cookie is disabled from the browser.
- Only textual information can be set in Cookie object.

Cookie class

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

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.
public void addCookie(Cookie ck)	method of HttpServletResponse interface is used to add cookie in response object
public Cookie[] getCookies()	method of HttpServletRequest interface is used to return all the cookies from the browser

Hidden Form Field

- In case of Hidden Form Field **a hidden (invisible) textfield** is used for maintaining the state of an user.
- In such case, we store the information in the hidden field and get it from another servlet.
- This approach is better if we have to submit form in all the pages and we don't want to depend on the browser.

Example:

<input type="hidden" name="uname" value="CDAC">

Advantage of Hidden Form Field

- It will always work whether cookie is disabled or not.

Disadvantage of Hidden Form Field:

- It is maintained at server side.
- Extra form submission is required on each pages.
- Only textual information can be used.

URL Rewriting

- In URL rewriting, we append a token or identifier to the URL of the next Servlet or the next resource.
- We can send parameter name/value pairs using the following format:

Example:

url?name1=value1&name2=value2&??

- A name and a value is separated using an equal = sign, a parameter name/value pair is separated from another parameter using the ampersand(&).
- When the user clicks the hyperlink, the parameter name/value pairs will be passed to the server. From a Servlet, we can use getParameter() method to obtain a parameter value.

Advantage of URL Rewriting

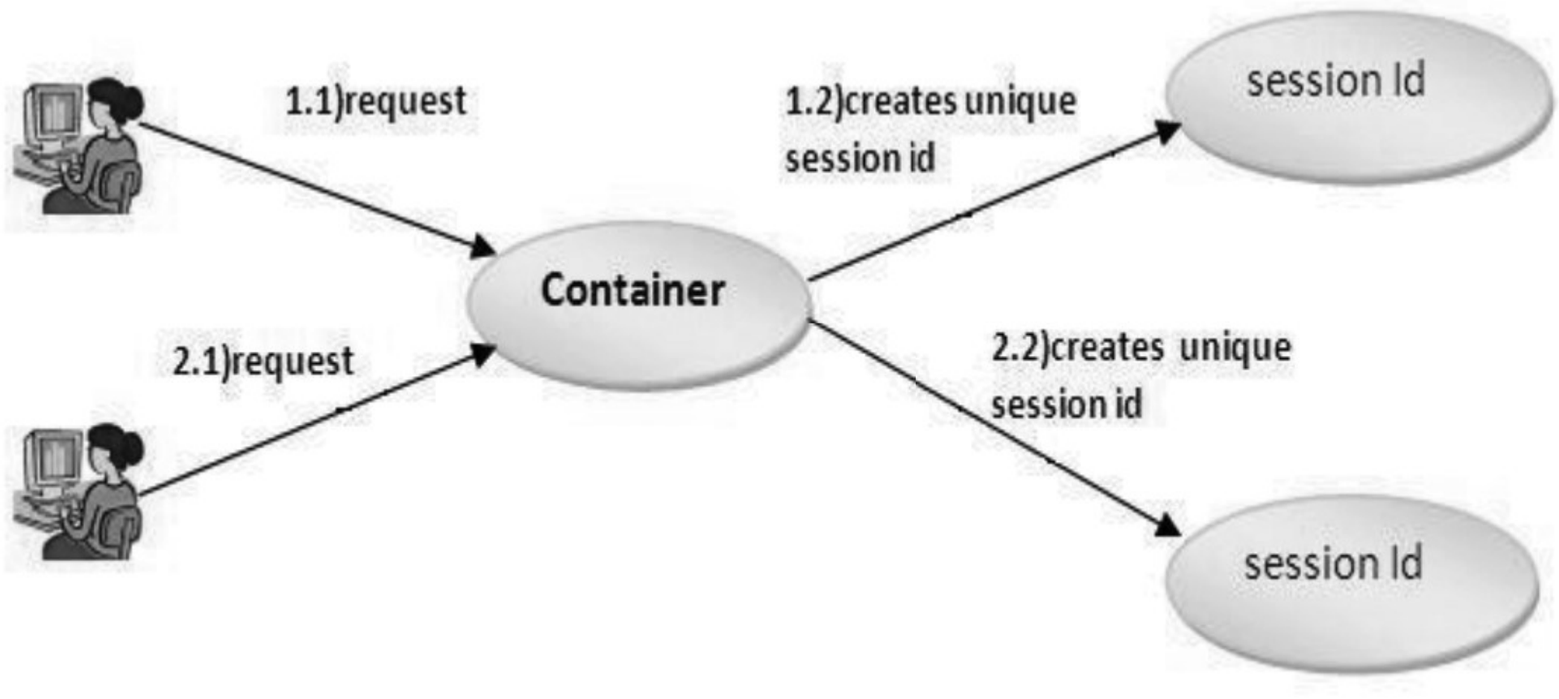
- It will always work whether cookie is disabled or not (browser independent).
- Extra form submission is not required on each pages.

Disadvantage of URL Rewriting

- It will work only with links.
- It can send Only textual information.

HttpSession interface

- In such a case, container creates a session id for each user.
- The container uses this id to identify the particular user.
- An object of HttpSession can be used to perform two tasks:
  - bind objects
  - view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?

- The HttpServletRequest interface provides two methods to get the object of HttpSession:
  - **public HttpSession getSession():** Returns the current session associated with this request, or if the request does not have a session, creates one.
  - **public HttpSession getSession(boolean create):** Returns the current HttpSession associated with this request or, if there is no current session and create is true, returns a new session.

Commonly used methods of HttpSession interface

- **public String getId():** Returns a string containing the unique identifier value.
- **public long getCreationTime():** Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
- **public long getLastAccessedTime():** Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
- **public void invalidate():** Invalidates this session then unbinds any objects bound to it.

Creating a new session

Returns the current session associated with this request, or if the request does not have a session, creates one.

```
HttpSession session = request.getSession();
```

```
HttpSession session = request.getSession(true);
```

Returns the current HttpSession associated with this request (or), if there is no current session and create is true, returns a new session.

Getting a pre-existing session

and create is false, returns null.

```
HttpSession session = request.getSession(false);
```

Destroying a session

```
session.invalidate();
```

destroy a session