Servlets

Jatin Ambasana

SUBJECT:
Advanced Java Programming

TOPIC:
Servlet API

Outline

1. Servlet Introduction, Servlet Life Cycle(SLC)
2. Types of Servlet
3. Servlet Configuration with Deployment Descriptor
4. Working with ServletContext and ServletConfig objects
5. Attributes in Servlet
6. Response and Redirection
    1. Using Request Dispacher
    2. Using sendRedirect Method
7. Session Tracking
    1. Using Cookies
    2. HTTPSession
    3. Hidden Form Field
    4. URL Rewriting
8. Filter API, Manipulating Responses using Filter API
9. Types of Servlet Event: ContextLevel and SessionLevel

Servlets - Introduction

- Used to create web application
  - Resides at server side and generates dynamic web pages.
- **Servlet** technology is robust and scalable because of java language.
  - Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But had many disadvantages.
- There are many interfaces and classes in the servlet API:
  - Servlet, GenericServlet, HttpServlet
  - ServletRequest, ServletResponse, HttpServletRequest, HttpServletResponse
  - ServletInputStream, ServletOutputStream
  - RequestDispatcher, ServletConfig, ServletContext, SingleThreadModel
  - Filter, FilterConfig, FilterChain

Servlets - Introduction

- Servlets run on the HTTP protocol.
  - HTTP is an request-response protocol.
  - Client sends a request message to the server
  - Server returns a response message
- Servlets are server-side programs (running inside a web server)
  - Handle clients' requests
  - Return a customized or dynamic response for each request.
- Servlets → Foundation of the Java server-side technology
  - Extensions of the servlet technology
    - JSP (JavaServer Pages), JSF (JavaServer Faces)
    - Struts, Spring, Hibernate

Servlets - Introduction

- Why Build Web Pages Dynamically?
  - The Web page is based on data submitted by the user
    - Ex: Results returned from search engine OR order-confirmation pages at online stores
  - The Web page is derived from data that changes frequently
    - Ex: A weather report or news headlines page
  - The Web page uses information from databases or other server-side sources
    - Ex: an e-commerce site could use a servlet to build a Web page that lists the current price and availability of each item that is for sale.
- Comparison with applets
  - Big applets require long download time
  - Applets do not have access to all the system resources
  - Server-side Java solves problems that applets face
    - Code executed on the server side and only the results sent to client
    - Servlets can access applications and data sources

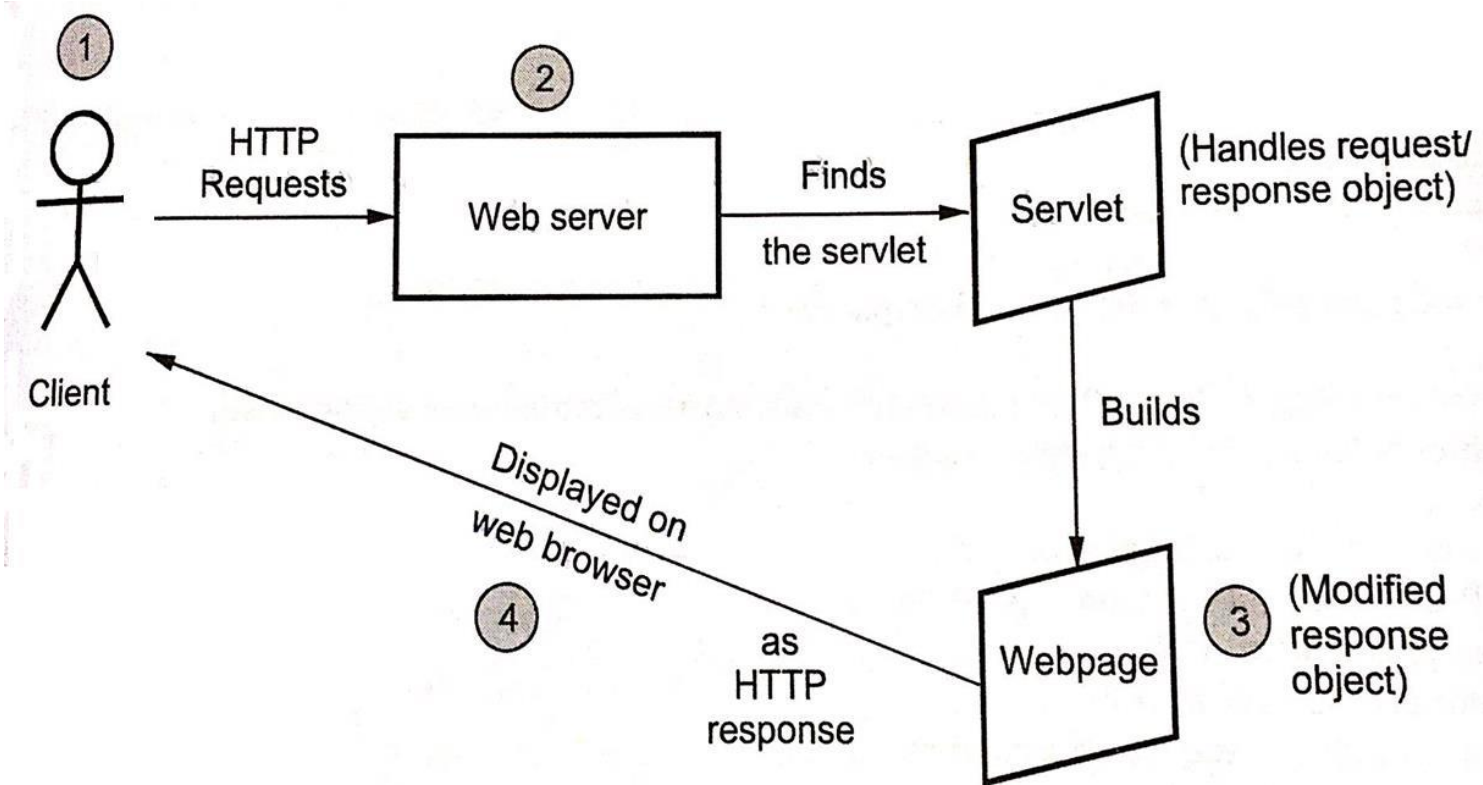
Servlets - Features

Servlets:

- Extensions to Java-enabled servers
- A dynamically loaded module that services requests from a Web server
- Executed within the Java Virtual Machine
- Runs on the server side
  - So independent of the browser and not affected by its compatibility issues

How does a Servlet work? (Execution)

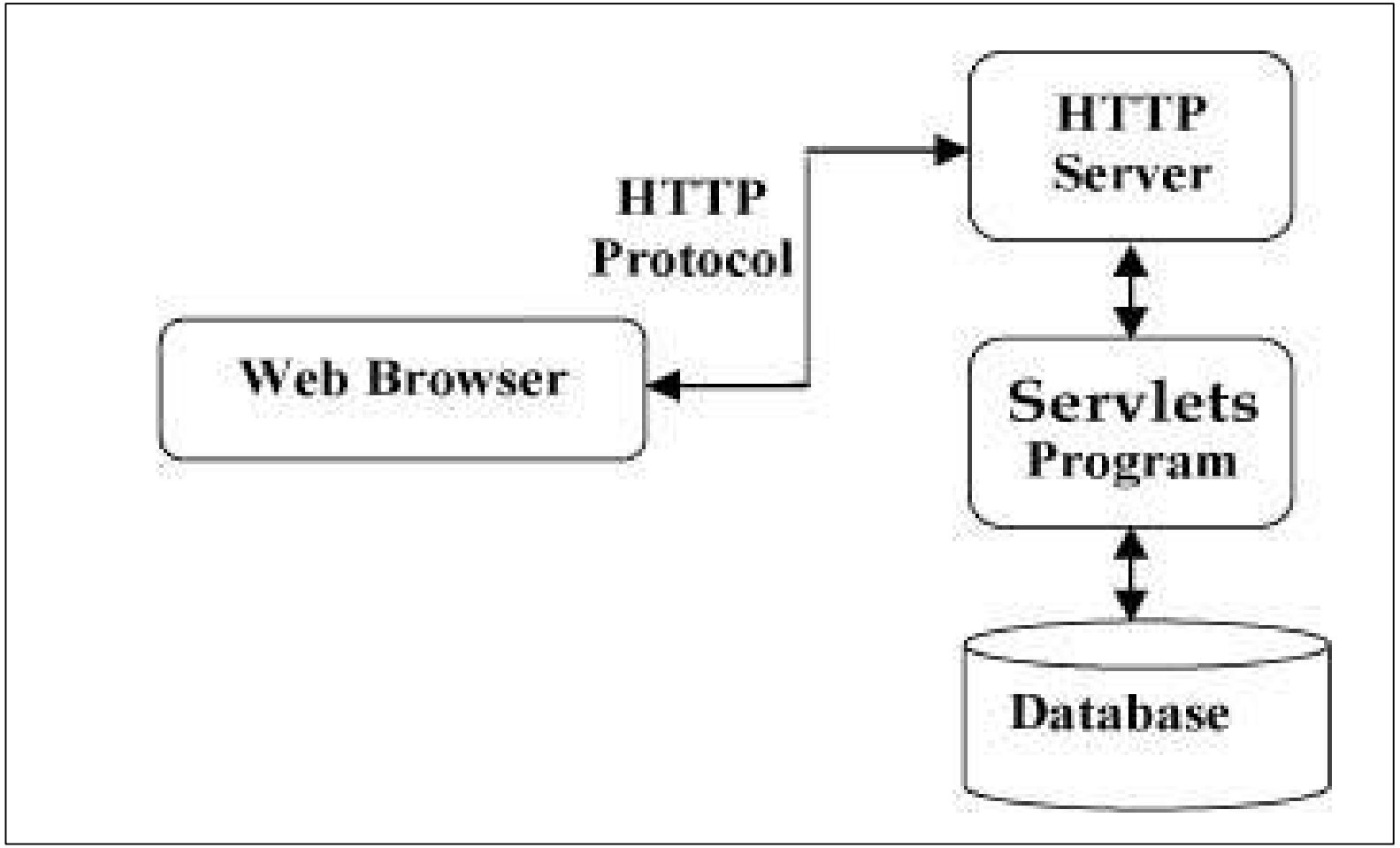
1. A client makes a request for some servlet, using a Web browser through an URL.
2. Web browser sends this request to Web server. Web server searches for the requested servlet.
3. Servlet is executed because of the request and it builds a web page accordingly as a response.
4. This web page is then displayed to the client. Request made by client is satisfied by the servlet.



A Servlet's Job

- Read data sent by client (form data)
- Read data sent by client (request headers)
- Generate the results
- Send the explicit data back to client (HTML)
- Send the implicit data to client (status codes and response headers)

Java Servlet Architecture



Java Servlet Architecture

Two main packages of the servlet architecture:

- **javax.servlet**
  - Generic interfaces and classes that are implemented and extended by all servlets
  - interfaces and classes used by the servlet/web container. Not specific to any protocol.

- **javax.servlet.http**
  - Interfaces and classes used for creating HTTP-specific servlets
  - Interfaces and classes that are responsible for HTTP requests only

Main heart of servlet architecture is the interface

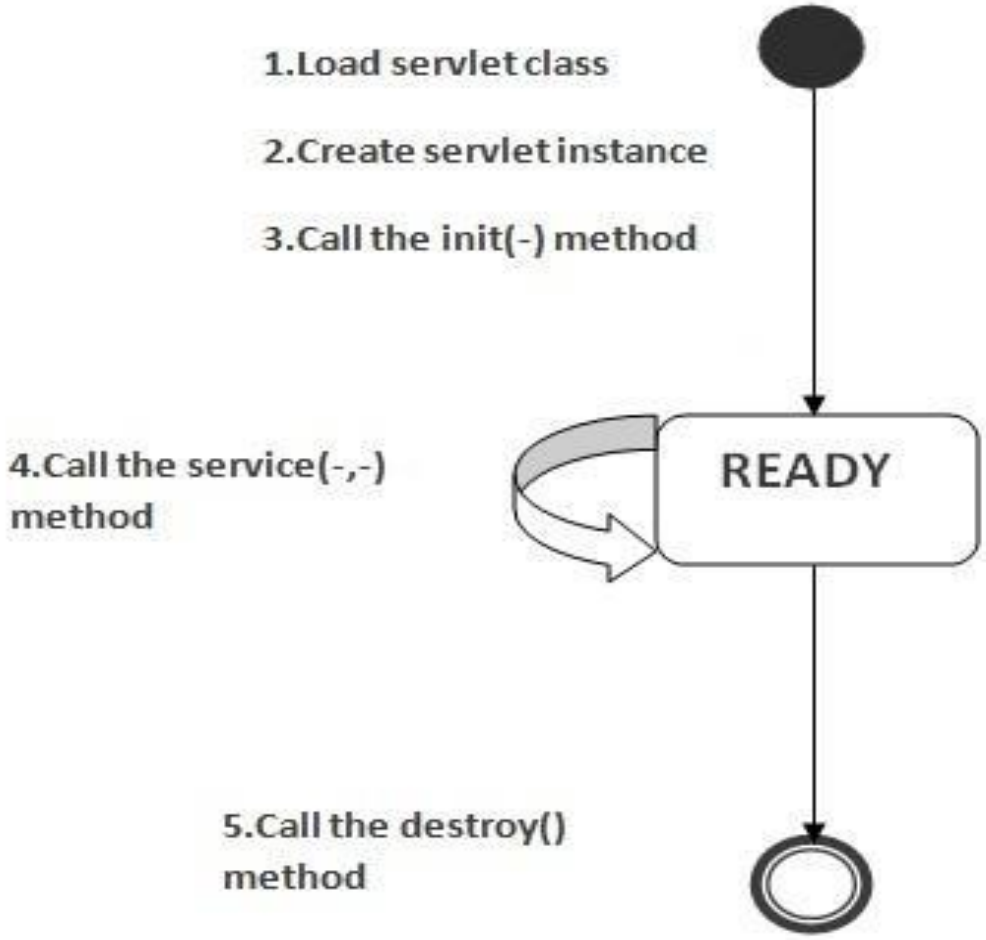
- **javax.servlet.Servlet**
  - It provides the framework for all servlets
  - Defines five basic methods – init, service, destroy, getServletConfig, getServletInfo

Servlet Life Cycle (SLC)

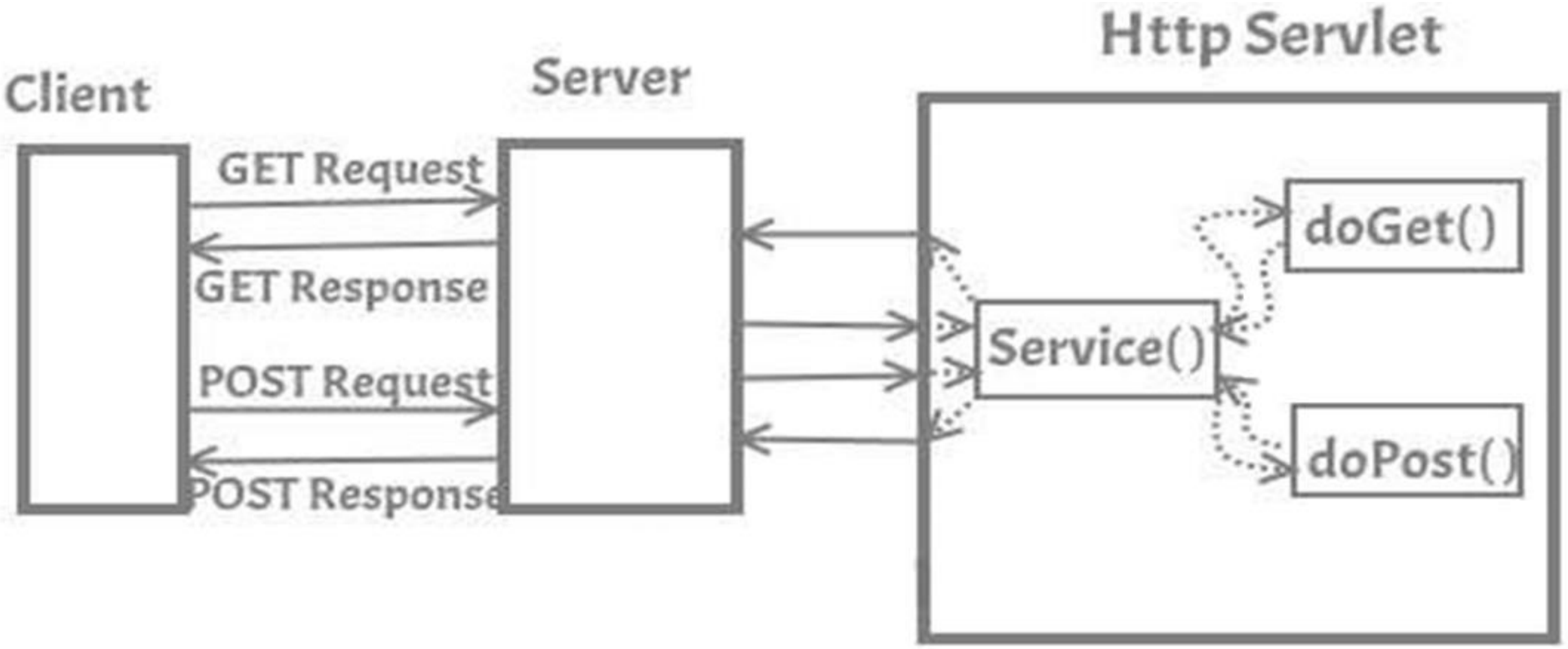
- The web container maintains the life cycle of a servlet instance

SLC Steps:

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



Servlet Life Cycle (SLC)



Servlet Life Cycle (SLC) – Cont.

1. **Servlet class is loaded:** Class loader is responsible to load the servlet class. Servlet class is loaded when a 1st request for it is received by web container.
2. **Servlet instance is created:** Web container creates an instance of a servlet after loading the servlet class. It is created only once in the servlet life cycle.
3. **init method is invoked:** Web container calls the **init** method only once after creating the servlet instance. It is used to initialize the servlet. It is the life cycle method of the **javax.servlet.Servlet** interface.
4. **service method is invoked:** Web container calls the service method each time when a request for the servlet is received.
5. **destroy method is invoked:** Web container calls the destroy method before removing the servlet instance from the service.

Types of Servlets

- A servlet can be created by three ways:

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

- Most used approach → Extending HttpServlet
  - It provides http request specific method such as doGet(), doPost(), doHead() etc.

Methods of different types of Servlets

1. Servlet (Interface)

a) init (ServletConfig) : void

b) service(ServletRequest,
ServletResponse) : void

c) destroy() : void

d) getServletInfo() : String

e) getServletConfig() : ServletConfig

2. GenericServlet (Abstract Class)

a) service(ServletRequest, ServletResponse) : void

3. HttpServlet (Abstract Class)

a) doGet(HttpServletRequest, HttpServletResponse)

b) doPost (HttpServletRequest, HttpServletResponse)

Servlet interface

- Servlet interface provides common behaviorto all the servlets.Servlet interface defines methods that all servlets must implement.

- 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 and **2 non-life cycle methods**.

Servlet interface – 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.

GenericServlet class

- Implements Servlet, ServletConfig and Serializable interfaces.
- Provides the implementation of all the methods of these interfaces except the service method.
- Can handle any type of request so it is protocol-independent.
- Create a generic servlet by **inheriting** the **GenericServlet** class and providing the implementation of the **service method.**

Methods of GenericServlet class:

- public void init(ServletConfig config)
  - Used to initialize the servlet.
- public abstract void service(ServletRequest request, ServletResponse response)
  - Provides service for the incoming request.
  - Invoked at each time when user requests for a servlet.

GenericServlet class (cont.)

Methods of GenericServlet class (cont.):

- public void destroy()
  - Invoked only once throughout the life cycle
  - Indicates that servlet is being destroyed.
- public ServletConfig getServletConfig()
  - Returns the object of ServletConfig
- public ServletContext getServletContext()
  - Returns the object of ServletContext.
- public String getServletInfo()
  - Returns information about servlet such as writer, copyright, version etc.
- public String getInitParameter(String name)
  - Returns the parameter value for the given parameter name
- public Enumeration getInitParameterNames()
  - Returns all the parameters defined in the web.xml file
- public String getServletName()
  - Returns the name of the servlet object

HttpServlet class

- Extends the GenericServlet class
- Implements Serializable interface
- Provides http specific methods → doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class:

- public void service(ServletRequest rq,ServletResponse rs)
  - Dispatches the request to the protected service method by converting the request and response object into http type.
- protected void service(HttpServletRequest rq, HttpServletResponse rs)
  - Receives the request from the service method
  - Dispatches the request to the doXxx() method depending on the incoming http request type

HttpServlet class

Methods of HttpServlet class:

- protected void doGet(HttpServletRequest rq, HttpServletResponse rs)
  - Handles the GET request It is invoked by the web container
- protected void doPost(HttpServletRequest rq, HttpServletResponse rs)
  - Handles the POST request. It is invoked by the web container.
- protected void doHead(HttpServletRequest rq, HttpServletResponse rs)
  - Handles the HEAD request. It is invoked by the web container.
- Similarly → doPut, doOptions, doTrace, doDelete
- protected long getLastModified(HttpServletRequest req)
  - Returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Get Vs Post – methods

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.

Servlet Configuration with Deployment Descriptor

- A **deployment descriptor (DD)** refers to a configuration file for an artifact that is deployed to some container/engine.

- In the Java Platform, Enterprise Edition, a deployment descriptor describes how a component, module or application (such as a web application or enterprise application) should be deployed.

- It directs a deployment tool to deploy a module or application with specific container options, security settings and describes specific configuration requirements.

- XML is used for the syntax of these deployment descriptor files.

Servlet Configuration with Deployment Descriptor

- For **web applications**, the deployment descriptor must be called **web.xml** and must reside in the **WEB-INF** directory in the web application root.

- For **Java EE applications**, the deployment descriptor must be named **application.xml** and must be placed directly in the **META- INF** directory at the top level of the application .ear file

- In Java EE, there are **two types** of deployment descriptors:

1. Java EE deployment descriptors
2. Runtime deployment descriptors

Servlet Configuration with Deployment Descriptor

- The Java EE deployment descriptors are defined by the language specification, whereas the runtime descriptors are defined by the vendor of each container implementation.

- For example, **the web.xml** file is a **standard Java EE** deployment descriptor, specified in the Java Servlet specification, but the **sun-web.xml** file contains configuration data specific to the **Sun GlassFish Enterprise Server** implementation.

- The **web.xml** file is located in the **WEB-INF** directory of your Web application. The first entry, under the root servlet element in web.xml, defines a **name** for the **servlet** and specifies the **compiled class** that executes the **servlet**.

Working with ServletConfig objects

- 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.**

Advantages:

- If the configuration information is modified in the web.xml file
  - No need to change the servlet
- Easier to manage the web application
  - If any specific content is modified regularly

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.
- **How to get the object of ServletConfig?**
- getServletConfig() method of Servlet interface returns the ServletConfig object
- **Syntax of getServletConfig() method**
  - public ServletConfig getServletConfig();

Defining init parameter in web.xml

```
<web-app>
......
 <servlet>
     <init-param>
          <param-name>parametername</param-name>
          <param-value>parametervalue</param-value>
     </init-param>
 </servlet>
......
</web-app>
```

Working with ServletContext objects

- An object of ServletContext is created by the web container **at the 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:

- **Easy to maintain** →If information is changed, no need to modify servlet

Defining context parameter in web.xml

```
<web-app>
......
 <context-param>
     <param-name>parametername</param-name>
     <param-value>parametervalue</param-value>
 </context-param>
......
</web-app>
```

Usage of ServletContext object

ServletContext object:

- Provides an interface between the container and servlet.
- Can be used to get configuration information from the web.xml file.
- Can be used to set, get or remove attribute from the web.xml file.
- Can be used to provide inter-application communication.

Methods of ServletContext interface

- public String getInitParameter(String name)
  - Returns the parameter value for the specified parameter name.
- public Enumeration getInitParameterNames()
  - Returns the names of the context's initialization parameters.
- public void setAttribute(String name, Object object)
  - Sets the given object in the application scope.
- public Object getAttribute(String name)
  - Returns the attribute for the specified name
- public Enumeration getInitParameterNames()
  - Returns names of context's initialization parameters as an Enumeration of String objects
- public void removeAttribute(String name)
  - Removes the attribute with the given name from the servlet context

How to get the ServletContext object?

1. **getServletContext() method** of ServletConfig interface returns the object of ServletContext.

2. **getServletContext() method** of GenericServlet class returns the object of ServletContext.

- **Syntax of getServletContext() method:-**
  - public ServletContext getServletContext()

Difference between ServletConfig and ServletContext

- The **ServletConfig** object refers to the **single servlet.**
- The **ServletContext** object refers to the **whole web application.**

Attributes in Servlet

- Attribute is a variable that the servlet programmers can use to pass any kind of text based information from one servlet to another.
- It is just like passing object from one class to another so that we can reuse the same object again and again.
- An **attribute in servlet** is an object that can be set, get or removed from one of the following scopes:

1. request scope
2. session scope
3. application scope

Attributes Specific Methods in Servlets

- There are 4 Attribute specific methods of ServletRequest, HttpSession and ServletContext interface

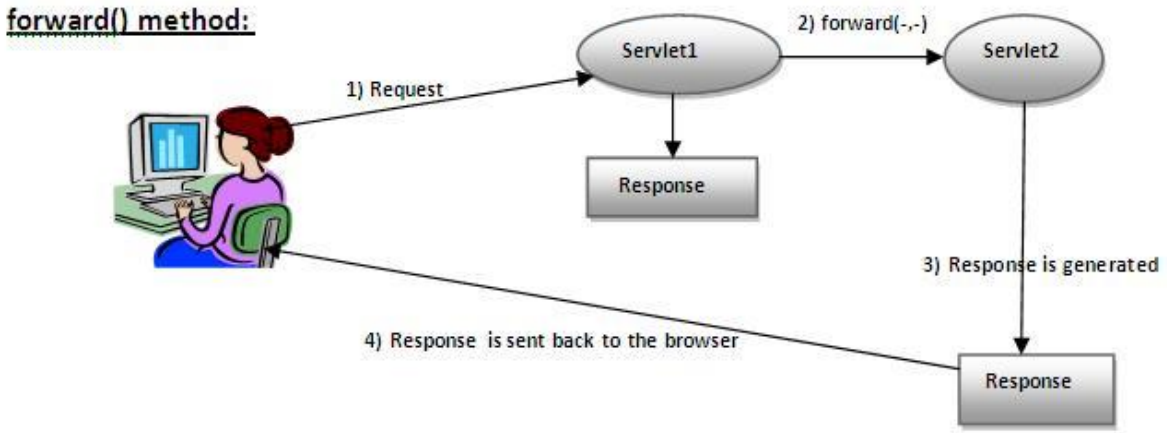
1. public void setAttribute(String name, Object object): sets the given object in the application scope.
2. public Object getAttribute(String name): Returns the attribute for the specified name.
3. public Enumeration getInitParameterNames(): Returns the names of the context's initialization parameters as an Enumeration of String objects.
4. public void removeAttribute(String name): Removes the attribute with the given name from the servlet context.

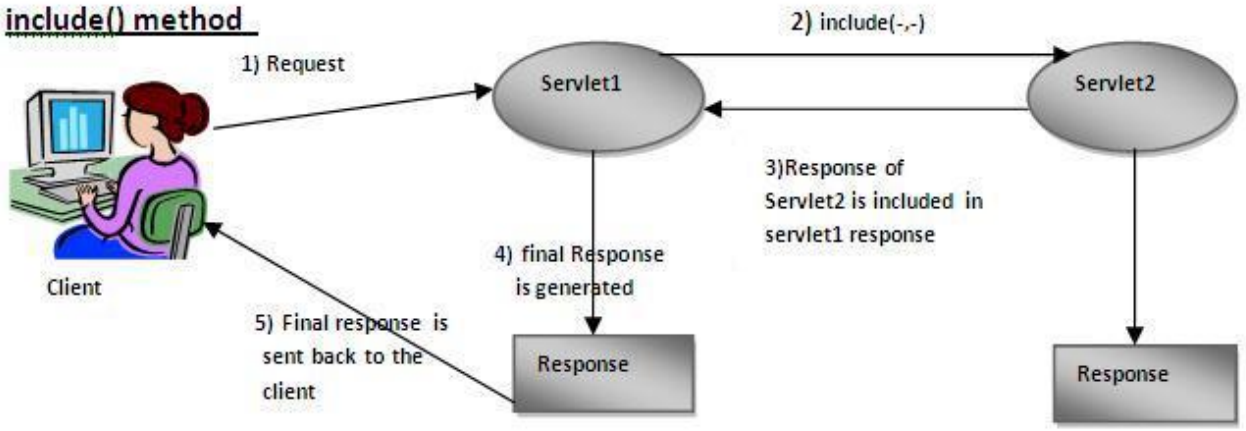
RequestDispatcher Interface

- The RequestDispatcher 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.
- It is one of the way of servlet collaboration.
- There are two methods defined in the RequestDispatcher interface:

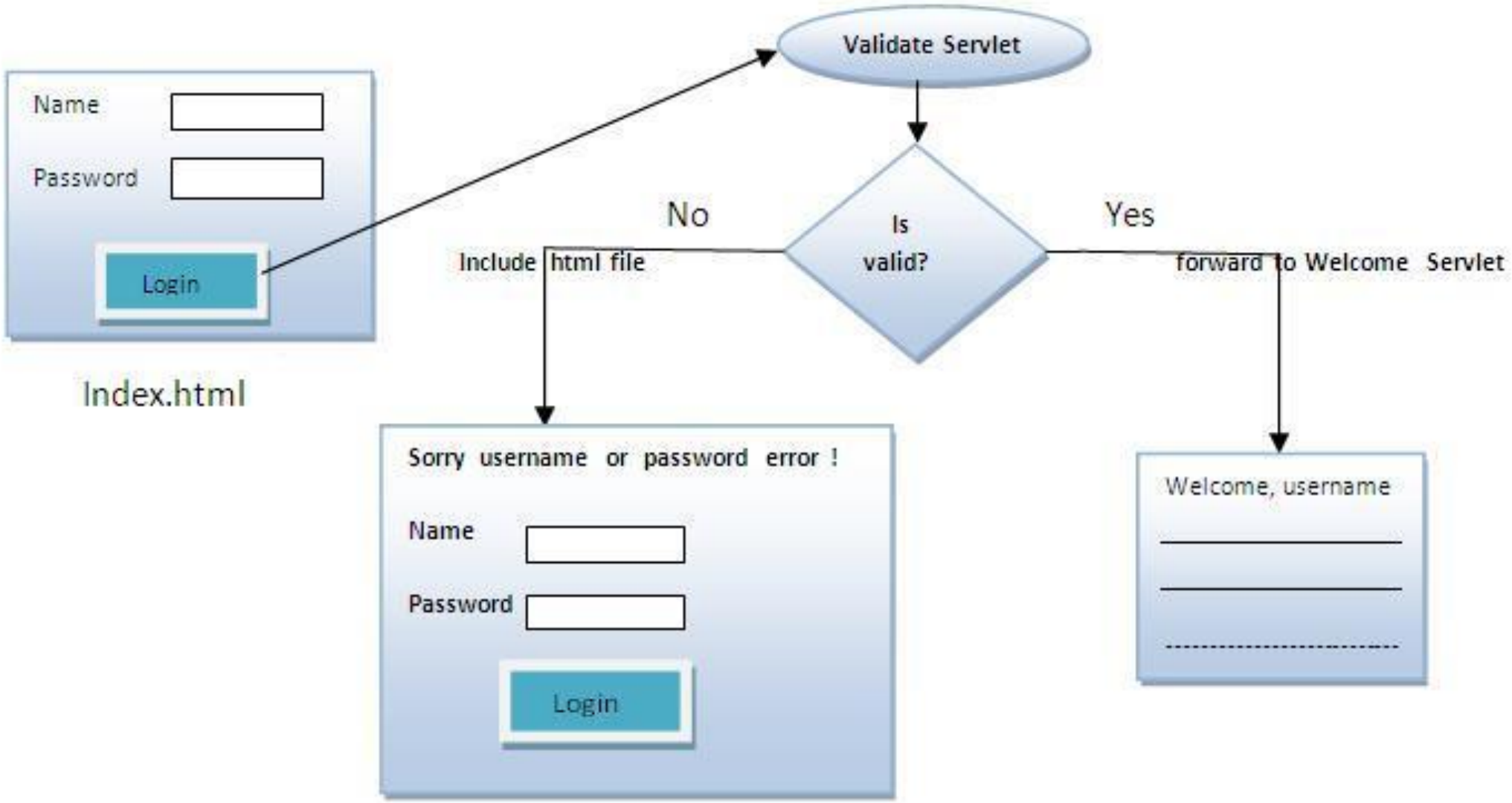
1. public void forward(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException: Forwards a request from a servlet to another resource (servlet, JSP file, or HTML file) on the server.
2. public void include(ServletRequest request, ServletResponse response)throws ServletException,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

Forward() vs Include () method:





Example of Request Dispatcher interface



SendRedirect in servlet

- **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file.
- It accepts relative as well as absolute URL.
- 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.
- Syntax of sendRedirect() method
- **public void** sendRedirect(String URL)**throws** IOException;
- Example of sendRedirect() method
- response.sendRedirect("http://www.marwadiuniversity.ac.in");

Example of sendRedirect method in servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SendRedirectServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
        throws ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter pw=res.getWriter();
        response.sendRedirect("http://www.google.com");
        pw.close();
    }
    }
```

Difference between forward() and sendRedirect() method

- Many differences between the forward() method of RequestDispatcher and sendRedirect() method of HttpServletResponse interface.

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");

Custom google search ex. using sendRedirect

- *index.html*

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>sendRedirect example</title>
</head>
<body>
<form action="GoogleSearchServlet">
<input type="text" name="query">
<input type="submit" value="Google Search">
</form>
</body>
</html>
```

Custom google search ex. using sendRedirect

- *GoogleSearch.java*

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GoogleSearchServlet extends HttpServlet
{
protected void doGet(HttpServletRequest request, HttpServletResponse respon
se)  throws ServletException, IOException
{
      String name = request.getParameter("query");
       response.sendRedirect("https://www.google.co.in/#q="+name);
   }
}
```

Session Tracking in Servlets

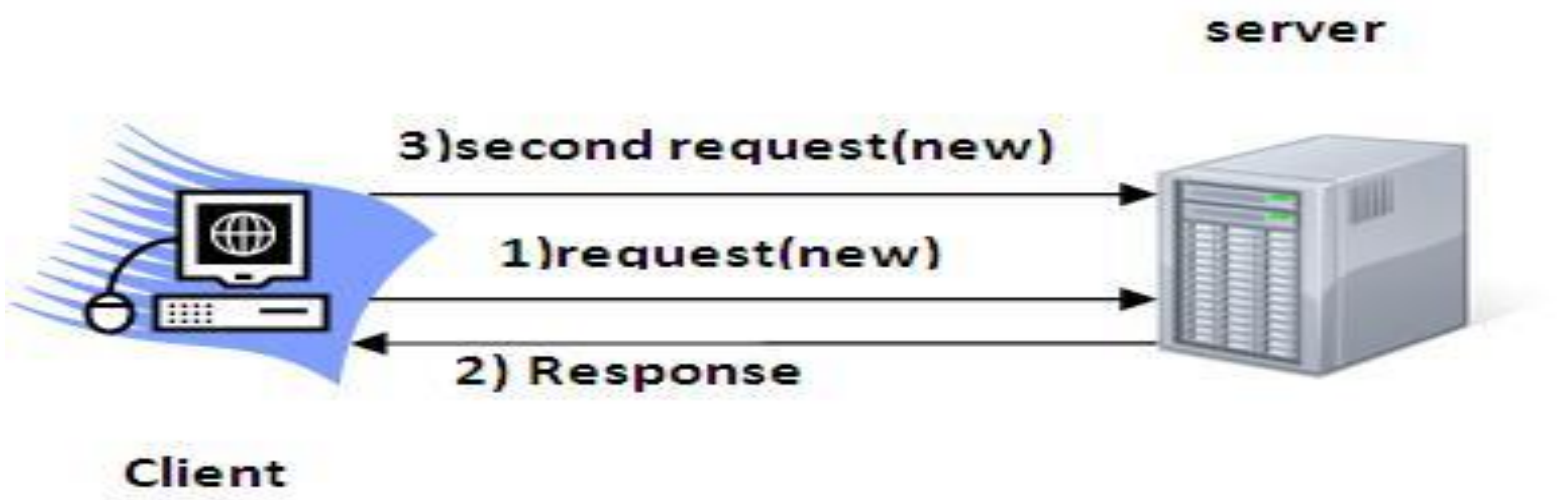
- **Session** simply means a particular interval of time.

- **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 Protocol so we need to maintain state using session tracking techniques.

Session Tracking (Cont.)

- **Http** protocol is a **stateless protocol** 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**.



Session Tracking (Cont.)

Why use Session Tracking?

- To recognize a particular user.

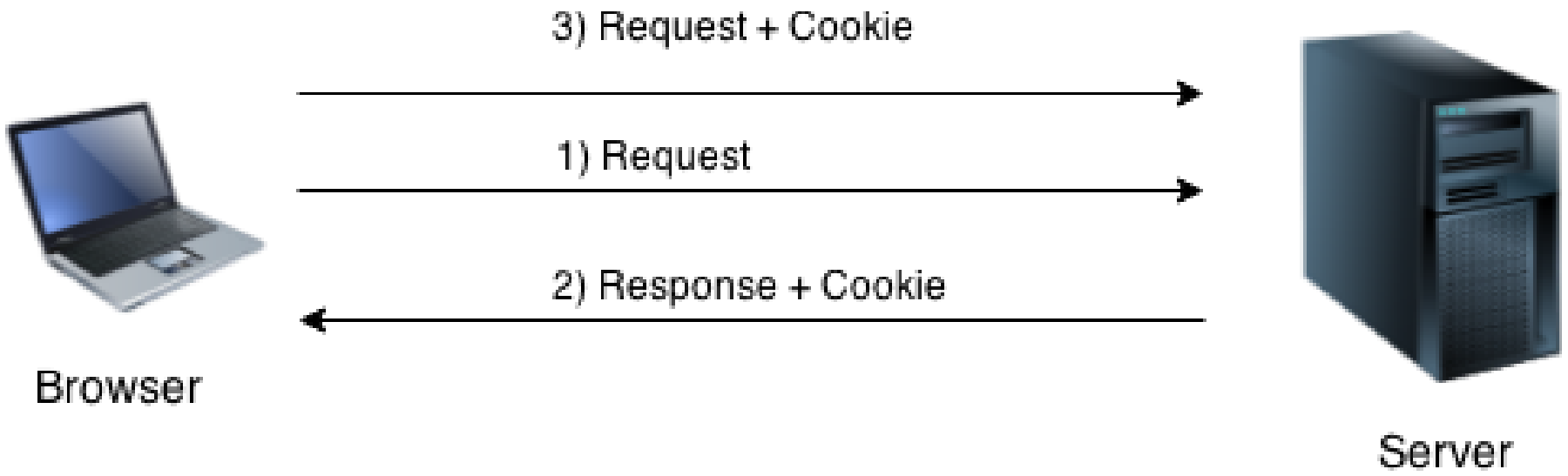
Session Tracking Techniques

- **Cookies**
- **Hidden Form Field**
- **URL Rewriting**
- **HttpSession object**

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.



Cookies (Cont.)

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 (Cont.)

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 signs out.

Cookie class & its constructor

- **javax.servlet.http.Cookie** class provides the functionality of using cookies

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

Useful Methods of Cookie class

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.

How to create Cookie?

```
//creating cookie object
Cookie ck=new Cookie("user","MarwadiUniv");

//adding cookie in the response
response.addCookie(ck);
```

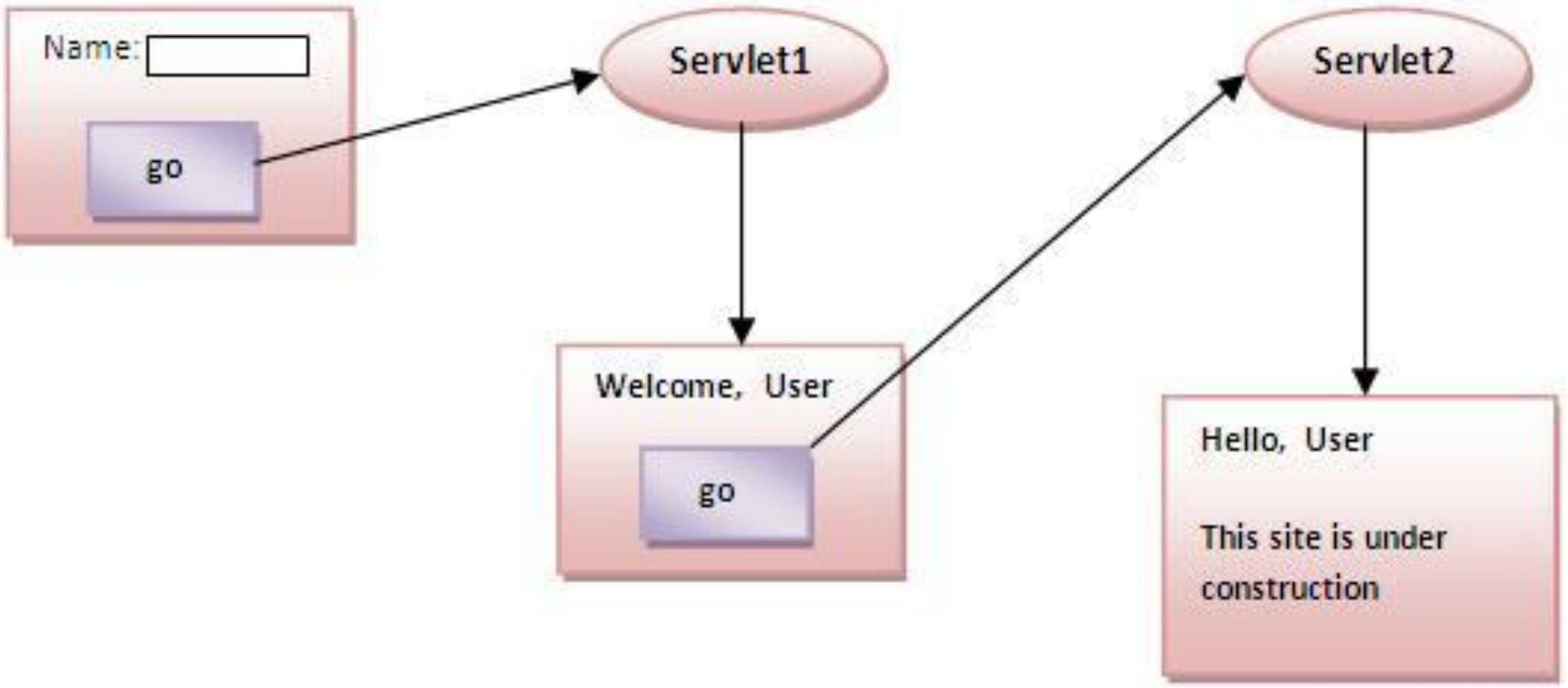
How to delete Cookie?

```
//deleting value of cookie
Cookie ck=new Cookie("user","");

//changing the maximum age to 0 seconds
ck.setMaxAge(0);

//adding cookie in the response
response.addCookie(ck);
```

How to get Cookies?

```
Cookie ck[ ]=request.getCookies();
for( int i = 0 ; i < ck.length ; i++ )
{
        //printing name and value of cookie
        out.print("<br>"+ck[i].getName()+" : "+ck[i].getValue());
}
```

Simple example of Servlet Cookies



Advantage of Cookies

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.

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.

- <input type="hidden" name="uname" value="Vimal Jaiswal">

Hidden Form Field (Cont.)

- **Real application of hidden form field**

- It is widely used in comment form of a website. In such case, we store page id or page name in the hidden field so that each page can be uniquely identified.

Hidden Form Field (Cont.)

- **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:
  - 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.

URL Rewriting (Cont.)

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.

URL Rewriting (Cont.)

- **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 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.

HttpSession interface (Cont.)

- **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.

Filter API

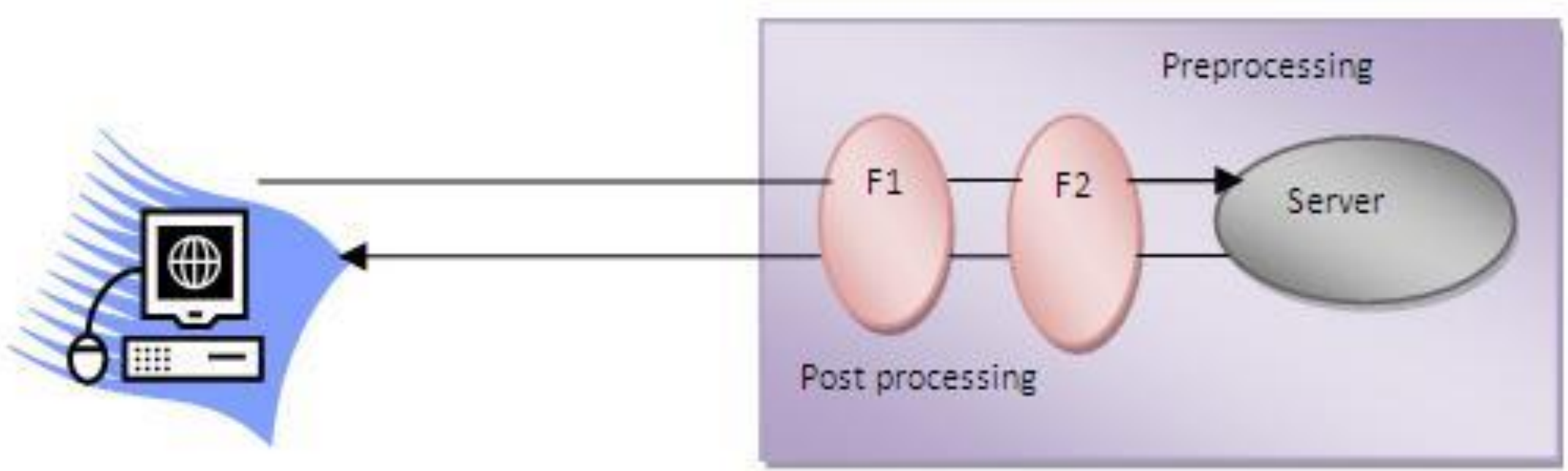
- Like servlet filter have its own API.
- The javax.servlet package contains the three interfaces of Filter API:
  - Filter
  - FilterChain
  - FilterConfig

Servlet Filter

- A **filter** is an object that is invoked at the pre-processing and post-processing of a request.
- It is mainly used to perform filtering tasks such as **conversion**, **logging**, **compression**, **encryption** and **decryption**, **input validation** etc.
- The **servlet filter is pluggable**, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.
- So maintenance cost will be less.

Servlet Filter (Cont.)

- A **filter** is an object that is invoked at the **pre-processing** and **post-processing** of a request.



Exploring the Need of Filters

- Let's take an example of a web application that formats the data to be presented to clients in a specific format, say Excel.

- However at later point of time, the clients may require data in some other format, such as HTML, PDF, or word.

- In such a situation, instead of modifying the code every time to change the format of data, a filter can be created to transform data dynamically in the required formats.

Need of Filters (Cont.)

Usage of Filter

- recording all incoming requests
- logs the IP addresses of the computers from which the requests originate
- conversion
- data compression
- encryption and decryption
- input validation etc.

Advantages of Filter

- Filter is pluggable.
- One filter don't have dependency onto another resource.
- Less Maintenance

Filter Life Cycle

- The filter interface calls the following methods during the life cycle of a filter :
- **The init( ) method :**
  - Refers to the method that is invoked by the web container only once when filter is initialized
- **The doFilter( ) method :**
  - Refers to the method that is invoked each time a user requests a resource, such as a servlet to which the filter is mapped.
  - When the doFilter( ) method is invoked, the servlet container passes the ServletRequest, ServletResponse, and FilterChain objects
- **The destroy( ) method :**
  - Refers to the method that is invoked when the filter instance is destroyed.

Filter Interface

- For creating any filter, you must implement the Filter interface. Filter interface provides the life cycle methods for a filter.

Method	**Description**
public void **init** (FilterConfig config)	init() method is invoked **only once**. It is used **to initialize the filter**.
public void **doFilter** (ServletRequest req, ServletResponse res, FilterChain chain)	doFilter() method is invoked **every time when user request to any resource**, to which the filter is mapped. It is used to perform filtering tasks.
public void **destroy**( )	This is invoked **only once when filter is taken out of the service**.

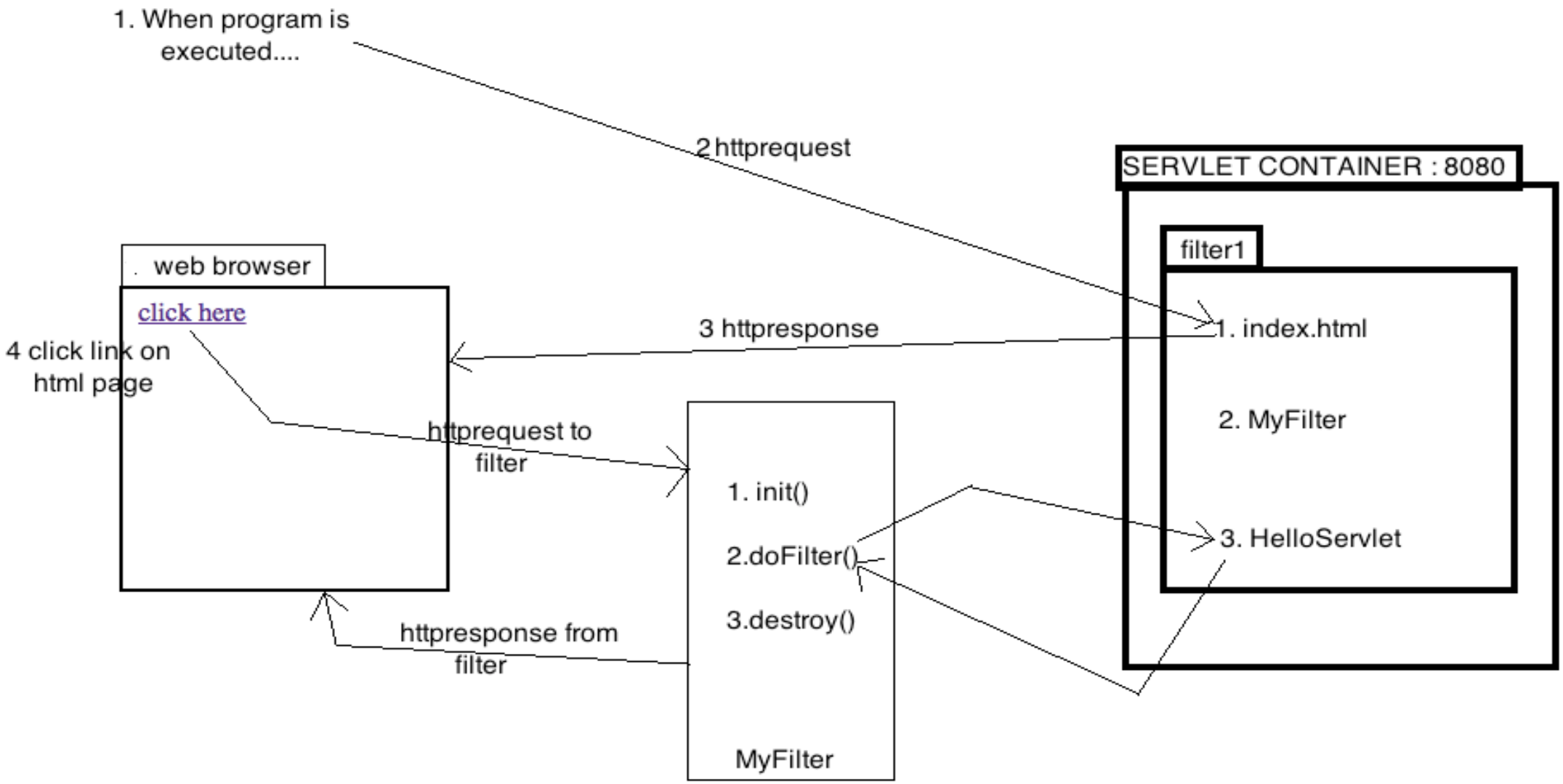
How to define a Filter?

- We can define filter in the same was as we define a servlet.

web.xml

```
<web-app>
 <filter>
    <filter-name>...</filter-name>
    <filter-class>...</filter-class>
 </filter>

 <filter-mapping>
   <filter-name>...</filter-name>
   <url-pattern>...</url-pattern>
</filter-mapping>

</web-app>
```

Simple Example of Filter



FILES NEEDED TO BE CREATED:

- index.html / index.jsp
- MyFilter.java
- HelloServlet.java
- web.xml

1. index.html

- click here

2. Filter (MyFilter)

```
import java.io.*;
import javax.servlet.*;
public class MyFilter implements Filter{
    public void init (FilterConfig arg0) throws ServletException { }
    public void doFilter (ServletRequest req, ServletResponse resp, FilterChain chain)
        throws IOException, ServletException {
                PrintWriter out=resp.getWriter();
                out.print("filter is invoked before");
                chain.doFilter(req, resp);//sends request to next resource
                out.print("filter is invoked after");
    }
    public void destroy( ) { }
}
```

3. Servlet : HelloServlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html"); // For setting MIME TYPE
        PrintWriter out = response.getWriter(); // creating a writer object
        out.print("<br>welcome to servlet<br>");
    }
}
```

4. web.xml

```
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<filter>
    <filter-name>f1</filter-name>
    <filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>f1</filter-name>
    <url-pattern>/servlet1</url-pattern>
</filter-mapping>
```

Same as given in html file

Same in both the filter and filter mapping tags

4. web.xml

```
<servlet>
    <servlet-name>HelloServlet</servlet-name>
    <servlet-class>HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>HelloServlet</servlet-name>
    <url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<filter>
    <filter-name>f1</filter-name>
    <filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>f1</filter-name>
    <url-pattern>/servlet1</url-pattern>
</filter-mapping>
```

Same as given in html file

Same in both filter and filter mapping tag

FilterChain interface

- The object of FilterChain is responsible to invoke the next filter or resource in the chain. This object is passed in the doFilter method of Filter interface. The FilterChain interface contains only one method:

- **public void doFilter(HttpServletRequest request, HttpServletResponse response):** it passes the control to the next filter or resource.

FilterConfig

- An object of FilterConfig is created by the web container. This object can be used to get the configuration information from the web.xml file.

Methods of FilterConfig interface

- **public void init(FilterConfig config):**
  - init() method is invoked only once it is used to initialize the filter.
- **public String getInitParameter(String parameterName):**
  - Returns the parameter value for the specified parameter name.
- **public java.util.Enumeration getInitParameterNames():**
  - Returns an enumeration containing all the parameter names.
- **public ServletContext getServletContext():**
  - Returns the ServletContext object

Event handling in Servlets

- When a request is mapped to a servlet, the Web container creates the instance of the servlet class and then initializes the servlet by calling the init() method.
- Once the init() method is executed, the service() method gets executed with the Request and Response parameters.
- After the service() method execution, the web container may call the destroy method if the Web container does not require the servlet.
- Different events of the Servlet life cycle can now received and particular methods can be called on the basis of these event.
- Depending on the type of event, a listener class (context listener or session listener or request listener) is defined.

Servlet Events

Following events can occur with Servlets

- Initializing and destroying Servlets
- Adding, removing or replacing attributes in Servlet Context
- Creating, activating, invalidating a session
- Adding, removing or replacing attributes in a Servlet session

Event classes and interfaces

Event classes

- ServletRequestEvent
- ServletContextEvent
- ServletRequestAttributeEvent
- ServletContextAttributeEvent
- HttpSessionEvent
- HttpSessionBindingEvent

Event interfaces

- ServletRequestListener
- ServletRequestAttributeListener
- ServletContextListener
- ServletContextAttributeListener
- HttpSessionListener
- HttpSessionAttributeListener
- HttpSessionBindingListener
- HttpSessionActivationListener

Types of Event Listeners

- Context Listeners are used to notify a class when the context is initialized or destroyed or when an attribute is added or removed to the web context.

- Session Listeners are used to notify a class when the session is initialized, destroyed, activated or when an attribute is added, replaced or removed.

- Request Listeners are used to notify a class when the request is coming into scope for a servlet or the request is getting out of scope for a servlet. A request is defined as coming into scope when it is about to enter the first servlet or servlet filter.

Request Level events

Request Level events: There are two event listeners for request level events

- **ServletRequestListener** interface is implemented to notify the request coming in scope and going out of scope for a servlet.

- **ServletRequestAttributeListener** interface is implemented to notify the changes (addition, replacement or removal) in the request attributes.

Servlet Context Level events

- **ServletContextListener** interface is implemented to notify the initialization or destruction of the Servlet.

- **ServletContextAttributeListener** interface is implemented to notify the changes (addition, replacement or removal) in the Servlet Context attributes

Servlet Session Level events

- The Servlet session level events refer to events that are used to maintain the client's session.

- **HttpSessionListener** interface is implemented to notify the initialization or destruction of the Http Session.

- **HttpSessionActivationListener** interface is implemented to notify when a sessions object change from one VM to another.

- **HttpSessionAttributeListener** interface is implemented to notify the changes (addition, replacement or removal) in the HttpSession attributes

HttpSessionEvent and HttpSessionListener

- HttpSessionEvent is notified when session object is changed.
  - Corresponding Listener interface for this event is HttpSessionListener.
- What operations can we perform?
  - Counting total and current logged-in users,
  - Maintaining a log of user details such as login time, logout time etc.

Methods of HttpSessionListener interface

- Two methods declared in the HttpSessionListener interface which must be implemented by the servlet programmer to perform some action:
  - public void sessionCreated(HttpSessionEvent e): is invoked when session object is created.
  - public void sessionDestroyed(ServletContextEvent e): is invoked when session is invalidated.

Example of ServletContextEvent and ServletContextListener

- index.html
- MyListener.java
- FetchData.java

index.html

- fetch records

MyListener.java

```
import javax.servlet.*;

import java.sql.*;

public class MyListener implements ServletContext
Listener

{

public void contextInitialized(ServletContextEvent e
vent) {

try{

Class.forName("oracle.jdbc.driver.OracleDriver");

Connection con=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:xe","system","orac
le");

//storing connection object as an attribute in Servlet
Context

    ServletContext ctx=event.getServletContext();

ctx.setAttribute("mycon", con);


}catch(Exception e){e.printStackTrace();}

}


public void contextDestroyed(ServletContextEve
nt arg0) {}

}
```

FetchData.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class FetchData extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
try{
//Retrieving connection object from ServletContext object
ServletContext ctx=getServletContext();
Connection con=(Connection)ctx.getAttribute("mycon");
//retieving data  from emp32 table
PreparedStatement ps=con.prepareStatement("select * from emp32",
ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=ps.executeQuery();
while(rs.next()){
out.print("<br>"+rs.getString(1)+" "+rs.getString(2));
}
con.close();
}catch(Exception e){e.printStackTrace();}
out.close();
}
}
```

Example of HttpSessionEvent and HttpSessionListener

Aim: To count total and current logged-in users

- index.html: to get input from the user.
- MyListener.java: A listener class that counts total and current logged-in users and stores this information in ServletContext object as an attribute.
- First.java: A Servlet class that creates session and prints the total and current logged-in users.
- Logout.java: A Servlet class that invalidates session.

index.html

```
<form action="servlet1">
        Name:<input type="text" name="username"><br>
        Password:<input type="password" name="userpass"><br>
        <input type="submit" value="login"/>
</form>
```

MyListener.java

```
import javax.servlet.*;
import javax.servlet.http.*;
public class CountUserListener implements
HttpSessionListener
{
   ServletContext ctx=null;
   static int total=0,current=0;
public void sessionCreated(HttpSessionEvent e)
{
   total++;
   current++;
   ctx = e.getSession().getServletContext();
       ctx.setAttribute("totalusers", total);
       ctx.setAttribute("currentusers", current);
}
public void sessionDestroyed(HttpSessionEvent e)
{
       current--;
       ctx.setAttribute("currentusers",current);
   }
}
```

First.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class First extends HttpServlet {
public void doGet(HttpServletRequest request,
 HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String n=request.getParameter("username");
      out.print("Welcome "+n);
      HttpSession session=request.getSession();
      session.setAttribute("uname",n);
      //retrieving data from ServletContext object
      ServletContext ctx=getServletContext();
      int t=(Integer)ctx.getAttribute("totalusers");
      int c=(Integer)ctx.getAttribute("currentusers");
      out.print("<br>total users= "+t);
      out.print("<br>current users= "+c);
      out.print("<br><a href='logout'>logout</a>");
      out.close();
   }
}
```

Logout.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class LogoutServlet extends HttpServlet {
public void doGet(HttpServletRequest request,
 HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        HttpSession session=request.getSession(false);
        session.invalidate();//invalidating session
        out.print("You are successfully logged out");
        out.close();
    }
```