CS –29 MAJOR - 11
Advance Java Programming(J2EE)

Presented by : Dhruvita Savaliya

UNIT 2 - Servlet

Topics of Servlet

- Servlet Introduction
- Architecture of a Servlet
- Servlet API (Javax.servlet and Javax.servlet.http)
- Servlet Life Cycle
- Servlet Configuration with Deployment Descriptor
- Developing and Deploying Servlets
- Handling Servlet Requests and Responses
- Reading Initialization Parameters
- Session Tracking Approaches (URL Rewriting, Hidden Form Fields, Cookies, Session API)

Introduction of Servlet :

- **Java Servlet** is a Java program that runs on a Java-enabled web server or application server. It handles client requests, processes them, and generates responses dynamically. Servlets are the backbone of many server-side Java applications due to their efficiency and scalability.
- **Key Features:**
- Servlets are used to develop dynamic web applications.
- Servlets are nothing but the Java programs which reside on the server side and their main purpose is to serve the client request.
- Servlets are fully compatible with Java. You can use any of the available Java APIs like JDBC inside the servlets.
- As servlets are written in Java, they are platform independent, robust, and secured.
- In Servlets, a thread is created for each request unlike in CGI where a process is created for each request. Hence, servlets give better performance than CGI.
- Servlets are protocol independent. i.e. they support FTP, SMTP, HTTP etc. protocols.

- Using servlets web developers can create fast and efficient server side application which can run on any servlet enabled web server.
- Servlets run entirely inside the Java Virtual Machine.
- Servlets can access the entire family of Java APIs, including the JDBC API to access enterprise databases.
- Servlets are not designed for a specific protocols. It is different thing that they are most commonly used with the HTTP protocols Servlets uses the classes in the java packages javax.servlet and javax.servlet.http.
- Its standard framework and use the highly portable java language.

What is Servlet ?

- Servlet can be described in many ways, depending on the context.
- **Servlet is a technology** i.e. used to create web application.
- **Servlet is an API** that provides many interfaces and classes including documentations.
- **Servlet is an interface** that must be implemented for creating any servlet.
- **Servlet is a class** that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- **Servlet is a web component** that is deployed on the server to create dynamic web page.
- **Benefits of Java Servlets :**

  Faster execution as Servlets do not create new processes for each request.

  Write-once, run-anywhere feature of Java.

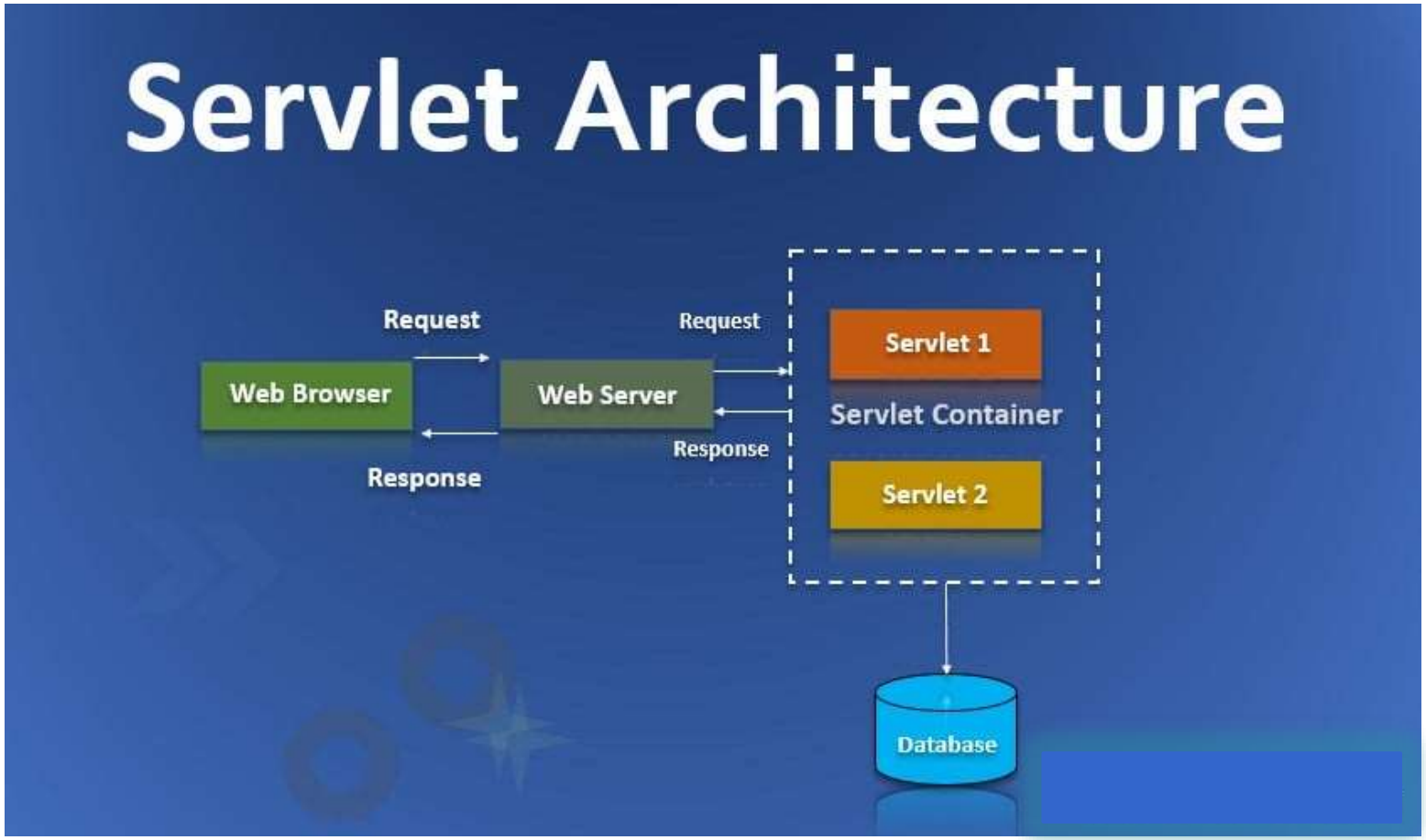
  Single instance handles multiple requests.

  Easily integrates with databases using JDBC.

  It inherits robust security features from web servers.

  Many web servers like Apache Tomcat are free to use with Java Servlets.

Servlet Architecture :



Components of Architecture :

- **1. Client :**
  In this architecture, the web browser acts as a Client.
  Client or user connected with a web browser.
  The client is responsible for sending requests or HttpRequestto the web server and processing the Web server's responses.
- **2. Web Server :**
  The web server controls how web users access hosted files and is responsible for processing user requests and responses.
  Here server is software it understands URLs and HTTP protocol.
  Whenever a browser needs to host a file on the web server, it processes a client request using an HTTP request; if it finds the requested file, it sends it back to the browser through HTTP Response.
  Static web servers send the file as it is, while dynamic web servers update the server-hosted file before sending it to the browser.

- **3. Web Container or Servlet Container :**

  A web container is a web server component that interacts with Java servlets.

  A web container manages the servlets' lifecycle and performs the URL mapping task.

  Web container handles the server-side requests of servlets, JSP, and other files.

  The critical tasks performed by servlets are loading and unloading servlets, creating and managing requests and response objects, and performing servlet management's overall tasks.

- **Advantages :**

  There are many advantages of servlet over CGI.

  The web container creates threads for handling the multiple requests to the servlet.

  Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low.

Servlet API :

- Servlets are the Java programs that run on the Java-enabled web server or application server.
- They are used to handle the request obtained from the webserver, process the request, produce the response, then send a response back to the webserver.
- In Java, to create web applications we use Servlets.
- To create Java Servlets, we need to use Servlet API which contains all the necessary interfaces and classes.
- Servlet API has 2 packages namely,

1. javax.servlet
2. javax.servlet.http

- **javax.servlet :**
- This package provides the number of interfaces and classes to support Generic servlet which is protocol independent.
- These interfaces and classes describe and define the contracts between a servlet class and the runtime environment provided by a servlet container.

Classes available in javax.servlet package:

Class Name	Description
GenericServlet	To define a generic and protocol-independent servlet.
ServletContextAttributeEvent	To generate notifications about changes to the attributes of the servlet context of a web application.
ServletContextEvent	To generate notifications about changes to the servlet context of a web application.
ServletInputStream	This class provides an input stream to read binary data from a client request.
ServletOutputStream	This class provides an output stream for sending binary data to the client.
ServletRequestAttributeEvent	To generate notifications about changes to the attributes of the servlet request in an application.
ServletRequestEvent	To indicate lifecycle events for a ServletRequest.
ServletRequestWrapper	This class provides the implementation of the ServletRequest interface that can be subclassed by developers to adapt the request to a Servlet.
ServletResponseWrapper	This class provides the implementation of the ServletResponse interface that can be sub classed by developers to adapt the response from a Servlet.

Interfaces available in javax.servlet package:

Interface Name	Description
Filter	To perform filtering tasks on either the request to a resource, or on the response from a resource, or both.
FilterChain	To provide a view into the invocation chain of a filtered request for a resource to the developer by the servlet container.
FilterConfig	To pass information to a filter during initialization used by a servlet container.
RequestDispatcher	It defines an object to dispatch the request and response to any other resource, means it receives requests from the client and sends them to a servlet/HTML file/JSP file on the server.
Servlet	This is the main interface that defines the methods in which all the servlets must implement. To implement this interface, write a generic servlet that extends javax.servlet.GenericServlet or an HTTP servlet that extends javax.servlet.http.HttpServlet.
ServletConfig	It defines an object created by a servlet container at the time of servlet instantiation and to pass information to the servlet during initialization.
ServletContext	It defines a set of methods that a servlet uses to communicate with its servlet container. The information related to the web application available in web.xml file is stored in ServletContext object created by container.
ServletContextAttributeListener	The classes that implement this interface receive notifications of changes to the attribute list on the servlet context of a web application.
ServletContextListener	The classes that implement this interface receive notifications about changes to the servlet context of the web application they are part of.

- **javax.servlet.http :**
- This package provides the number of interfaces and classes to support HTTP servlet which is HTTP protocol dependent.
- These interfaces and classes describe and define the contracts between a servlet class running under HTTP protocol and the runtime environment provided by a servlet container.

Classes available in javax.servlet.http package:

Class Name	Description
Cookie	Creates a cookie object. It is a small amount of information sent by a servlet to a Web browser, saved by the browser, and later sent back to the server used for session management.
HttpServlet	Provides an abstract class that defines methods to create an HTTP suitable servlet for a web application.
HttpServletRequestWrapper	This class provides implementation of the HttpServletRequest interface that can be subclassed to adapt the request to a Servlet.
HttpServletResponseWrapper	This class provides implementation of the HttpServletResponse interface that can be subclassed to adapt the response from a Servlet.
HttpSessionBindingEvent	This events are either sent to an object that implements HttpSessionBindingListener when it is bound or unbound from a session, or to a HttpSessionAttributeListener that has been configured in the deployment descriptor when any attribute is bound, unbound or replaced in a session.
HttpSessionEvent	To represent event notifications for changes to sessions within a web application.

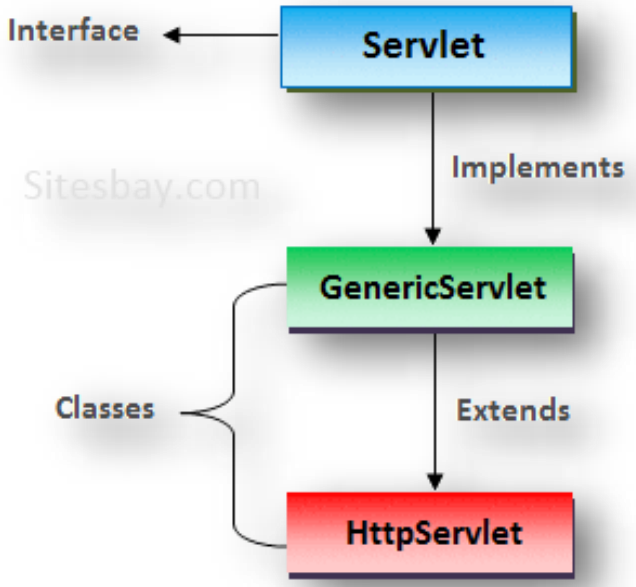
Interfaces available in javax.servlet.http package:

Interface Name	Description
HttpServletRequest	To provide client HTTP request information for servlets. It extends the ServletRequest interface.
HttpServletResponse	To provide HTTP-specific functionality in sending a response to client. It extends the ServletResponse interface.
HttpSession	It provides a way to identify a user across web application/web site pages and to store information about that user.
HttpSessionActivationListener	Container to notify all the objects that are bound to a session that sessions will be passivated and that session will be activated.
HttpSessionAttributeListener	To get notifications of changes to the attribute lists of sessions within this web application, this listener interface can be implemented.
HttpSessionBindingListener	It causes an object to be notified by an HttpSessionBindingEvent object, when it is bound to or unbound from a session.
HttpSessionListener	To receive notification events related to the changes to the list of active sessions in a web application.

How to create Servlet :

- According to servlet API we have three ways to creating a servlet class.

1. By implementing servlet interface
2. By extending GenericServlet class
3. By extending HttpServlet class



Servlet Interface :

- Servlet interface provides common behavior to 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.

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.

```
import javax.servlet.*;
import java.io.*;
public class LifeCycleServlet
   implements Servlet {
   ServletConfig config = null;
public void init(ServletConfig sc)
   {
      config = sc;
      System.out.println("in init");
   }
public void service(ServletRequest
   req, ServletResponse res)
      throws ServletException,
   IOException
   {
   res.setContenttype("text/html");
   PrintWriter pw =res.getWriter();
      pw.println("<h2>hello from
   life cycle servlet</h2>");
}
public void destroy()
   {
      System.out.println("destroy");
   }
public String getServletInfo()
   {
      return "LifeCycleServlet";
   }
public ServletConfig
   getServletConfig()
   {
      return config
   }
}
```

GenericServlet Class :

- GenericServlet implements the Servlet interface and provides an implementation for all its method except the service() method hence it is abstract.
- GenericServletclass implements Servlet, ServletConfig and Serializable interfaces.
- GenericServlet class defines a protocol- independent(HTTPless) servlet.
- You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class. They are as follows:

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

Example of Generic Servlet :

```
import java.io.*;
import javax.servlet.*;
public class First extends GenericServlet{
   public void service(ServletRequest req,ServletResponse
   res)
   throws IOException,ServletException{
         res.setContentType("text/html");
         PrintWriter out=res.getWriter();
         out.print("<html><body>");
         out.print("<b>hello generic servlet</b>");
         out.print("</body></html>");
   }
}
```

HttpServlet Class :

- The HttpServlet class extends the GenericServlet class and implements Serializable interface.
- It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class

There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

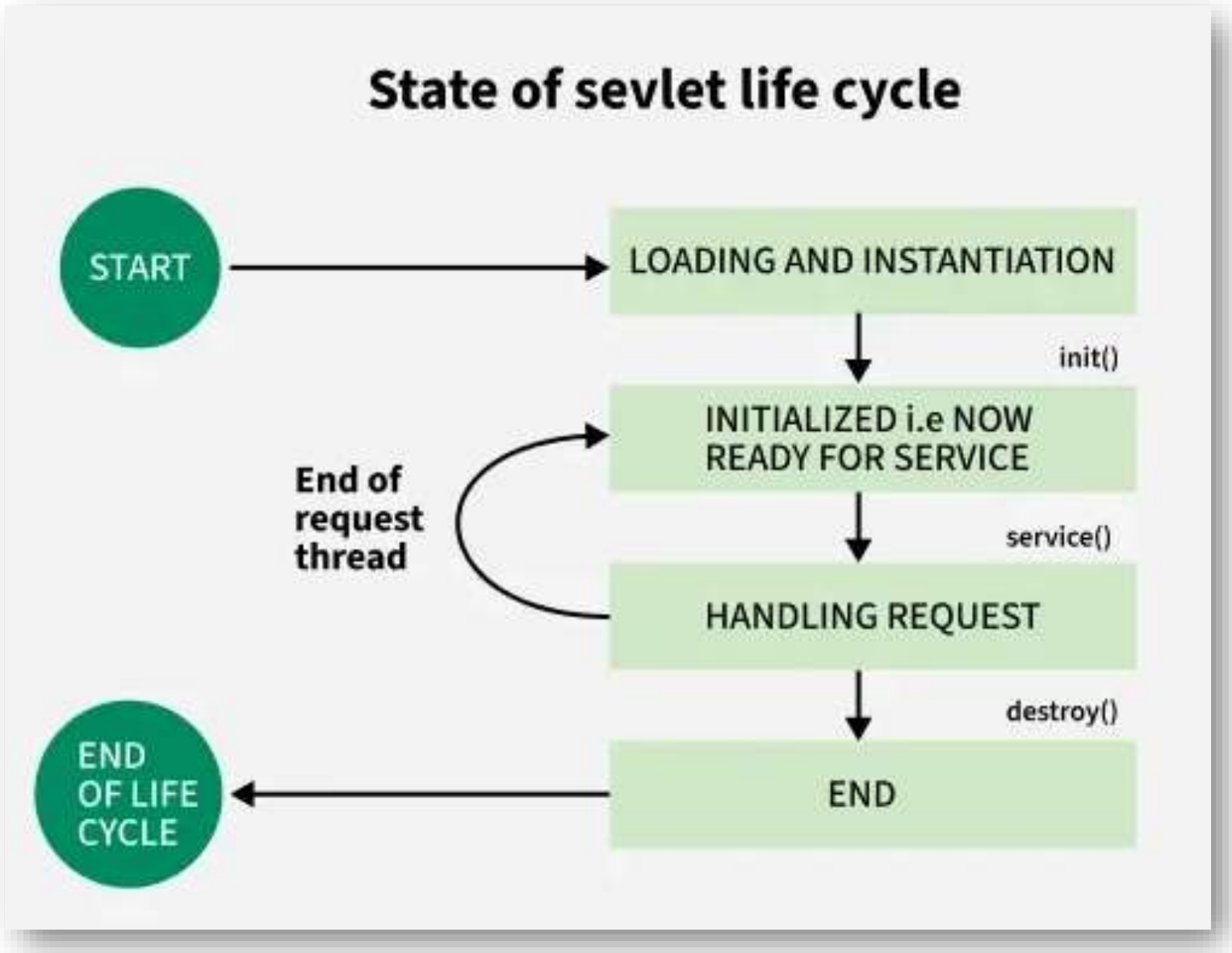
Difference between HttpServlet and GenericServlet

- Both these Classes are Abstract Classes.
- GenericServlet is a super class of HttpServlet class.
- The main difference is that, HttpServlet is a protocol dependent whereas GenericServlet is protocol independent. So GenericServlet can handle all types of protocols, but HttpServlet handle only HTTP specific protocols.
- GenericServlet belongs to javax.servlet package.
- HttpServlet belongs to javax.servlet.http package
- GenericServlet is an abstract class which extends Object and implements Servlet, ServletConfig and java.io.Serializable interfaces.
- HttpServlet is an abstract class which extends GenericServlet and implements java.io.Serializable interface.
- GenericServlet supports only service() method does not contain doGet() and doPost() methods.
- HttpServlet support also doGet(), doPost(), doHead() methods (HTTP 1.0) plus doPut(), doOptions(), doDelete(), doTrace() methods (HTTP 1.1).

Servlet Life Cycle :

- A servlet life cycle can be defined as the entire process from its creation till the destruction.
- The servlet is initialized by calling the init() method.
- The servlet calls service() method to process a client's request.
- The servlet is terminated by calling the destroy() method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.
- **Servlet life cycle contains five steps:**

1. Loading of Servlet
2. Creating instance of Servlet
3. Invoke init() once
4. Invoke service() repeatedly for each client request
5. Invoke destroy()


State of sevlet life cycle
START
LOADING AND INSTANTIATION
init()
INITIALIZED i.e NOW READY FOR SERVICE
End of request thread
service()
HANDLING REQUEST
destroy()
END
END OF LIFE CYCLE

- **Step 1: Loading of Servlet**

  The first stage of the Servlet lifecycle involves loading and initializing the Servlet. The Servlet container performs the following operations:

  Loading: The Servlet container loads the Servlet class into memory.

  Instantiation: The container creates an instance of the Servlet using the no-argument constructor.

  The Servlet container can load the Servlet at one of the following times:

  During the initialization of the web application (if the Servlet is configured with a zero or positive integer value in the deployment descriptor).

  When the Servlet is first requested by a client (if lazy loading is enabled).

- **Step 2: Creating instance of Servlet**

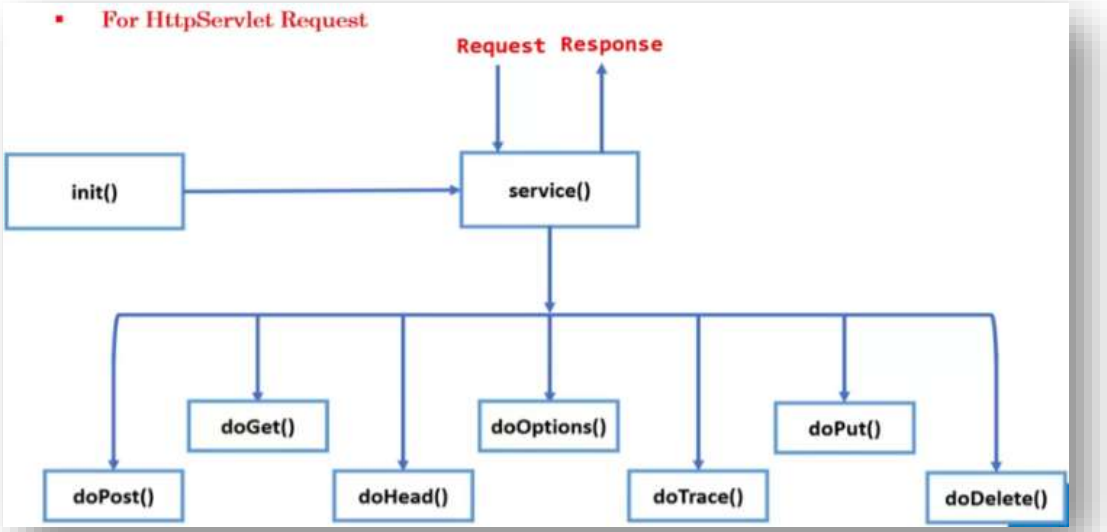
  Once all the Servlet classes loaded, the servlet container creates instances of each servlet class. Servlet container creates only once instance per servlet class and all the requests to the servlet are executed on the same servlet instance.

- **Step 3: Invoke init() method**
- The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.
- The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

  Example of ini():

```
public void init() throws ServletException
 {
      // Initialization code...
}
```

- **Step 4: Invoke service() method**
- Each time the web server receives a request for serv it spawns a new thread that calls service() method.
- If the servlet is GenericServlet then the request is served by the service() method itself,if the servlet is HttpServlet then service() method receives the request and dispatches it to the correct handler method based on the type of request.
- The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.



- **Example of service() :**

  public void **service**(ServletRequest request, ServletResponse response) throws ServletException, IOException

  {

  //statements

  }
- **Step 5: Invoke destroy()**

  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.

  After the destroy() method is called, the servlet object is marked for garbage collection.
- **Example of destroy() :**

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

Java PrintWriter class :

- Java PrintWriter class is the implementation of Writer class. It is used to print the formatted representation of objects to the text-output stream.
- The servlet binded with the url-pattern is called.
- The method being called depends on the kind of request (doGet, doPost, doPut).
- The method normally receives the request and response objects, we then call the .getWriter() method for the response obj that gets us the stream on which we can write our output.
- response.getWriter() returns a PrintWriter object that can send character text to the client.
- **Example :**

  PrintWrite pw= response.getWriter();

  pw.print(“hello”);

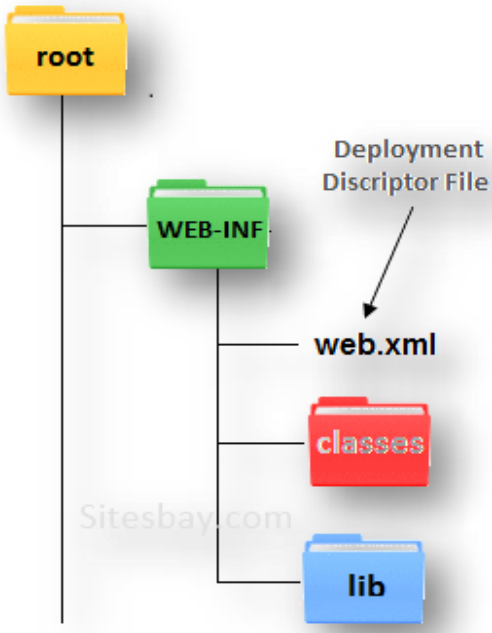
Servlet Configuration with Deployment Descriptor (WEB.XML file) :

- Java web applications use a deployment descriptor file to determine how URLs map to servlets, which URLs require authentication, and other information. This file is named web.xml, and resides in the app's WAR under the **WEB-INF**/ directory. web.xml is part of the servlet standard for web applications.
- A web application's deployment descriptor describes the classes, resources and configuration of the application and how the web server uses them to serve web requests. When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code that ought to handle the request.
- The deployment descriptor is a file named web.xml. It resides in the app's WAR under the WEB-INF/ directory. The file is an XML file whose root element is <web-app>.

- Servlets and URL paths
- web.xml defines mappings between URL paths and the servlets that handle requests with those paths. The web server uses this configuration to identify the servlet to handle a given request and call the class method that corresponds to the request method. For example: the doGet() method for HTTP GET requests.
- To map a URL to a servlet, you declare the servlet with the <servlet> element, then define a mapping from a URL path to a servlet declaration with the <servlet-mapping> element.
- The <servlet> element declares the servlet, including a name used to refer to the servlet by other elements in the file, the class to use for the servlet, and initialization parameters. You can declare multiple servlets using the same class with different initialization parameters. The name for each servlet must be unique across the deployment descriptor.

Web.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
   http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
   <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>
```



Handling Servlet Requests and Responses :

1. ServletRequest Interface
2. Servlet Response Interface

1. Handling ServletRequest Interface:

- An object of ServletRequest is used to provide the client request information to a Servlet such as content type, content length, parameter names and values, header information's, attributes etc.
- Methods of ServletRequest interface:

  1.getParameter():is used to obtain the value of a parameter by name.

- **getParameter(String name):**
  Returns the value of a request parameter as a String.
- **getParameterNames():**
  Returns an enumeration of all request parameter names.
- **getParameterValues():**
  Returns the array of a request parameter value as a String.
- **getAttribute(String name):** Returns the value of an attribute.
- **setAttribute(String name, Object value):** Sets the value of an attribute.
- **getSession():**
  Returns the current session associated with this request.
- **getHeader(String name):**
  Returns the value of a specified request header.
- **getHeaderNames():**
  Returns an enumeration of all request header names.
- **getRequestURI():**
  Returns the part of this request's URL from the protocol name up to the query string.
- **getInputStream():**
  Returns the body of the request as a ServletInputStream.

- **Example 1 getParameter()**

```
<form action="Servlet_file" method="get">
Enter your name<input type="text" name="name"><br>
<input type="submit" value="login">
</form>
```

- **Handling Servlet Requests and Responses (Servlet_file.java file)**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class req_demo extends HttpServlet
{
   public void doGet(HttpServletRequest req,HttpServletResponse
   res)throws IOException,ServletException
   {
         res.setContentType("text/html");
         PrintWriter out=res.getWriter();
         String name=req.getParameter("name");//will return value
         out.println("Welcome"+name);
   }
}
```

- **Example 2 getParameterValues() index.html file**

```
<form action=" req_demo" method="post">
Habits :
<input type="checkbox" name="habits" value="Reading">
Reading
<input type="checkbox" name="habits" value="Movies">Movies
<input type="checkbox" name="habits" value="Writing">Writing
<input type="checkbox" name="habits" value="Singing">Singing
<input type="submit" value="Submit"> </form>
```

Servlet_file.java

```
res.setContentType("text/html");
PrintWriter pw=res.getWriter();
String[] values=req.getParameterValues("habits");
pw.println("Selected Values...");
for(int i=0;i<values.length;i++)
{
   pw.println("<li>"+values[i]+"</li>");
}
```

Example 3 getParameterNames()

```
<form action="get_pname" method="post">
Name:<input type="text" name="name">
Country:<input type="text" name="country">
<input type="submit" value="Submit">
</form>
```

Servlet_file.java

```
PrintWriter pw=res.getWriter();
res.setContentType("text/html");
Enumeration en=req.getParameterNames();
while(en.hasMoreElements())
{
   Object objOri=en.nextElement();
   String param=(String)objOri;
   String value=req.getParameter(param);
   pw.println("Parameter Name is '"+param+"'
   and Parameter Value is '"+value+"'");
}
```

- **2. Handling Servlet Response Interface:**

  1. The servlet container is connected to the web server that receives Http Requests from client on a certain port.

  2. When client sends a request to web server, the servlet container creates HttpServletRequest and HttpServletResponse objects and passes them as an argument to the servlet service() method.

  3. The response object allows you to format and send the response back to the client.

  4. First we will see the commonly used methods in the ServletReponse interface and then we will see an example.

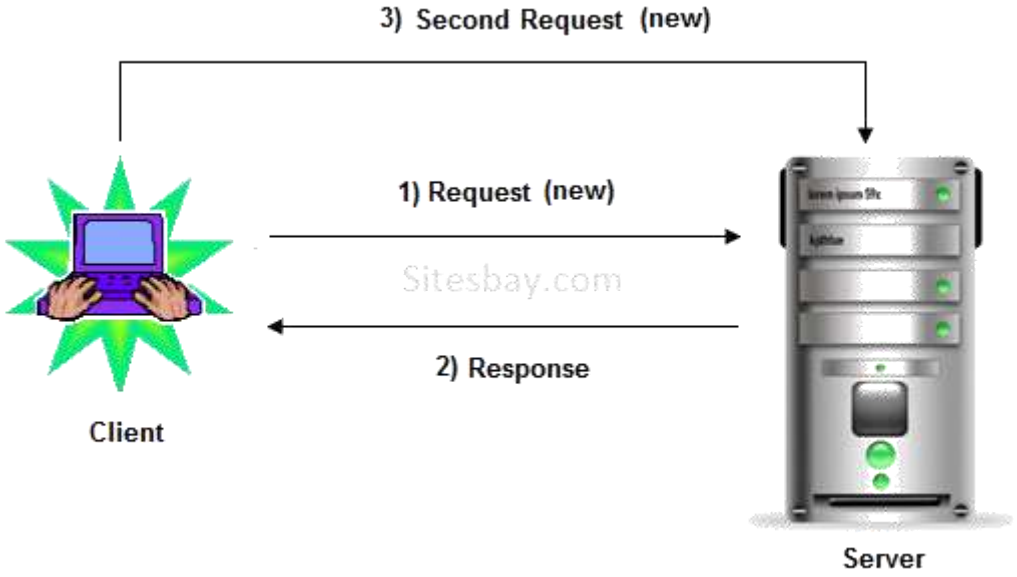
- **Methods of Servlet Response Interface:**

  1) String getCharacterEncoding(): It returns the name of the MIME charset used in body of the response sent to the client.
- 2) String getContentType(): It returns the response content type. e.g. text, html etc.
- 3) ServletOutputStream getOutputStream(): Returns a ServletOutputStream suitable for writing binary data in the response.
- 4) java.io.PrintWriter getWriter(): Returns the PrintWriter object.
- 5) void setCharacterEncoding(java.lang.String charset): Set the MIME charset (character encoding) of the response.
- 6) void setContentLength(int len): It sets the length of the response body.
- 7) void setContentType(java.lang.String type): Sets the type of the response data.
- 8) void setBufferSize(int size): Sets the buffer size.
- 9) int getBufferSize(): Returns the buffer size.
- 10) void flushBuffer(): Forces any content in the buffer to be written to the client.
- 11) boolean isCommitted(): Returns a boolean indicating if the response has been committed.
- 12) void reset(): Clears the data of the buffer along with the headers and status code.

Session Tracking Approaches :

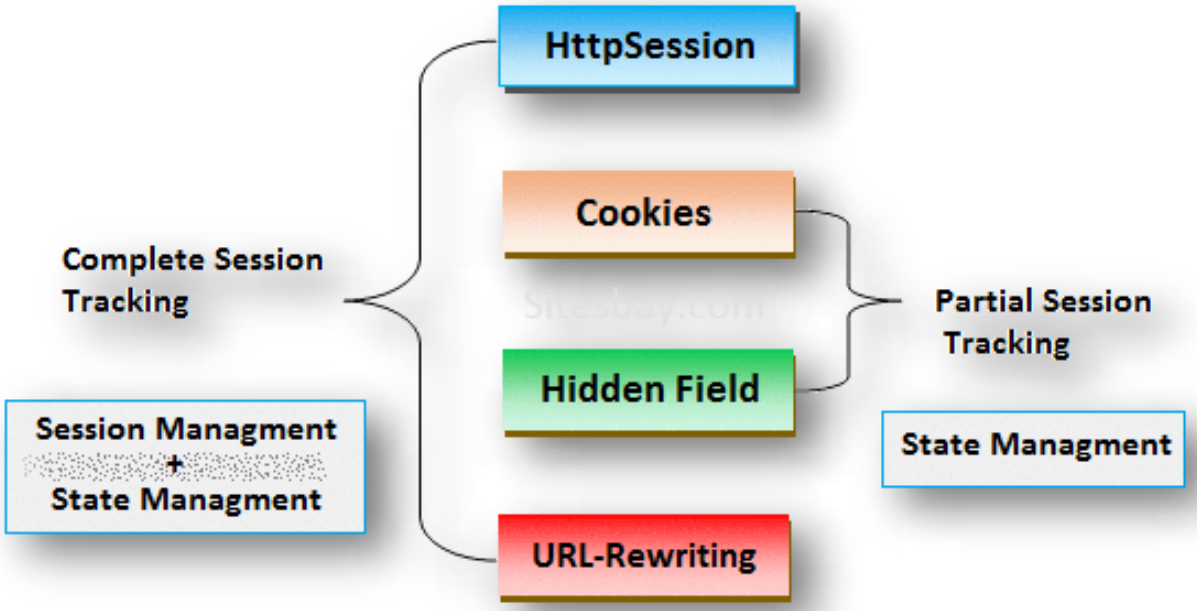
- We all know that HTTP protocol is a stateless protocol which means no user information is pertained and server considersevery request as a new request.
- Session is the conversion of user within span of time. In general meaning particular interval of time.
- Trackingis the recording of the thing under session.
- Session Trackingis remembering and recording of client conversion in span of time.
- It is also called as session management.
- If web application is capable of remembering and recording of client conversion in span of time then that web application is called as stateful web application.

- **Why need Session Tracking ?**
- Http protocol is stateless, to make stateful between client and server we need Session Tracking.
- Session Tracking is useful for online shopping, mailing application, E-Commerce application to track the conversion.
- Http protocol is stateless, that means each request is considered as the new request.



- **Why Http is design as stateless protocol ?**
- If Http is stateful protocol for multiple requests given by client to web application single connection will be used between browser and web server across the multiple requests.
- This **may make clients to engage connection with web server for long time event though the connection are ideal.**
- Due to this the web server reach to maximum connections even though most of its connection are idle.
- To overcome this problem **Http is given as stateless.**

- **Session Tracking Techniques :**
- Servlet technology allows four technique to track conversion, they are;
- Cookies
- URL Rewriting
- Hidden Form Field
- HttpSession



Cookies :

- 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.
- In order to use cookies in java, use a Cookie class that is present in **javax.servlet.http** package.
- To make a cookie, create an object of Cookie class and pass a name and its value.
- To add cookie in response, use addCookie(Cookie) method of **HttpServletResponse** interface.
- To fetch the cookie, getCookies() method of Request Interface is used.
- **Types of Cookie:**
- There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

- **1.Non-persistent cookie:**
  It is valid for single session only.
  It is removed each time when user closes the browser.
- **2.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.

- **Methods in Cookies**

1. clone(): Overrides the standard java.lang.Object.clone method to return a copy of this Cookie.
2. getComment(): Returns the comment describing the purpose of this cookie, or null if the cookie has no comment.
3. getDomain(): Gets the domain name of this Cookie.
4. getMaxAge(): Gets the maximum age in seconds of this Cookie.
5. getName(): Returns the name of the cookie.
6. getPath(): Returns the path on the server to which the browser returns this cookie.
7. getSecure(): Returns true if the browser is sending cookies only over a secure protocol, or false if the browser can send cookies using any protocol.
8. getValue(): Gets the current value of this Cookie.
9. getVersion(): Returns the version of the protocol this cookie complies with.
10. setValue(String newValue): Assigns a new value to this Cookie.
11. setVersion(int v): Sets the version of the cookie protocol that this Cookie complies with.

- **Example :**

```
<form action="NewServlet">
      Cookie name : <input type="text" name="txt_ck">
      <input type="submit" value="create cookie">
</form>
```

- **NewServlet.java (import package for cookie : import javax.servlet.http.Cookie;)**

```
protected void processRequest(HttpServletRequest request, HttpServletResponse
    response)          throws ServletException, IOException {
       response.setContentType("text/html;charset=UTF-8");
       try (PrintWriter out = response.getWriter()) {
            String a=request.getParameter("txt_ck");
            Cookie ck_obj=new Cookie("name",a); //create cookie
            ck.setMaxAge(20); // set expiry time
            response.addCookie(ck_obj); //add cookie
            Cookie ck_print[]=request.getCookies(); //retrieve all cookie
            out.println("Hello "+ ck_print[0].getValue()); //print value of cookie
          out.println("<a href='Next_Servlet1'>click here</a>");
       }
   }
```

- **Next_Servlet1.java :**

```
protected void processRequest(HttpServletRequest request,
   HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        Cookie all_ck[]=request.getCookies();
        if (all_ck != null) {
            out.println("name of cookie :"+all_ck[0].getName());
            out.println("value of cookie :"+all_ck[0].getValue());
        }
        else
            out.println("cookie not found");
    }
}
```

2.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.
- **Syntax:**

  <input type="hidden" name="uname" value="admin">
- **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.

- **Example of hidden form field :**
- **File.html**

```
<form action="hidden_servlet">
<input type="hidden" value="tybca" name="txt_hidden">
<input type="submit" value="create cookie">
</form>
```

- **hidden_servlet.java**

```
protected void processRequest(HttpServletRequest request,
   HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        out.println(request.getParameter("txt_hidden"));
    }
```

3.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 get Parameter() method to obtain a parameter value.
- out.println("click here to open new page");

- **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:**
- Generate more network traffic.
- **It will work only with links.**
- It can send Only textual information.
- Less secure because query string in session id displace on address bar.

- **servlet_file.java**

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response)  throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            String n="tybca";
            out.println("<a href='url_second_servlet?p1="+n+"'>click here to
open new page</a>");
        }
}
```

- **url_second_servlet.java**

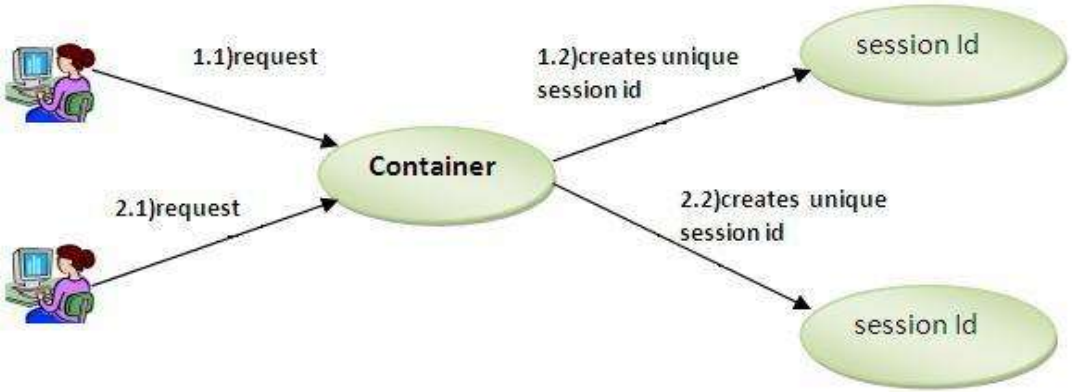
```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)   throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
         out.println(request.getParameter("p1"));
      }
    }
```

4.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:

  1.binds the objects

  2.view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.
- **methods of HttpSession interface:**
- getId():Returns a string containing the unique identifier value.
- getSession():Returns the current session associated with this request, or if the request does not have a session, creates one.



1. **String getId():** Returns a unique identifier string assigned to the session.
2. **long getCreationTime():** Returns the time, in milliseconds since midnight January 1, 1970 GMT, when the session was created.
3. **long getLastAccessedTime():** Returns the last time, in milliseconds since midnight January 1, 1970 GMT, the client sent a request associated with this session.
4. **void invalidate():** Invalidates the session and unbinds any objects bound to it, effectively ending the session.
5. **void setAttribute(String name, Object value):** Binds an object to the session, associating it with the specified name. If an attribute with the same name already exists, it is replaced.
6. **Object getAttribute(String name):** Retrieves the value of the session attribute with the specified name. Returns null if no such attribute exists.
7. **Enumeration<String> getAttributeNames():** Returns an Enumeration of String objects, each representing the name of an attribute bound to the session.
8. **void removeAttribute(String name):** Removes the attribute with the specified name from the session.
9. **void setMaxInactiveInterval(int interval):** Sets the maximum time, in seconds, that the session should remain active without client requests before being invalidated by the container.
10. **int getMaxInactiveInterval():** Returns the maximum inactive interval for the session, in seconds.
11. **boolean isNew():** Returns true if the server created the session and it has not yet been accessed by the client (i.e., this is the first request associated with this session).

Method	Description
public HttpSession getSession()	Gets the HttpSession object. If the request doesn't have a session associated with it, a new session is created
public HttpSession getSession(boolean create)	Gets the session associated with the request. If not already present, then a new one is created based on the value of the boolean argument passed into it
public String getId()	Returns the unique session id
public long getCreationTime()	It returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
public long getLastAccessedTime()	It returns the time when this session was last accessed, measured in milliseconds since midnight January 1, 1970 GMT.
public long getLastAccessedTime)	It returns the time when this session was last accessed, measured in milliseconds since midnight January 1, 1970 GMT.
public void invalidate()	Invalidates the session

Example for creating a session :

- In this example create 4 files :

1. One html file for design
2. Create one servlet file for create session if username is ‘admin’.
3. Create second servlet file for print value of session ,session is created else redirect to the design file.
4. Create logout servlet file for unbind object of session and redirect with the design file.

1. session_design.html:

```
<form action="session_servlet_first">
   User name : <input type="text" name="txt_usr">
   <input type="submit" value="login">
</form>
```

2. session_servlet_first.java

Add package for use HttpSession import javax.servlet.http.HttpSession;

```
protected void processRequest(HttpServletRequest request, HttpServletResponse
    response)
        throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
         out.println("create session page");

         if(request.getParameter("txt_usr").equals("admin"))
         {
            out.println("match");

            HttpSession session=request.getSession(); // Get or create session
            session.setAttribute("s_nm",request.getParameter("txt_usr")); // Store user data

            out.print("<a href='session_servlet_second'>click here to second session
   page</a>");
         }
         else
            out.println("unmatch");
      }
   }
```

3. session_servlet_second.java

Add package for use HttpSession import
javax.servlet.http.HttpSession;

```
protected void processRequest(HttpServletRequest request,
    HttpServletResponse response)
        throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
         HttpSession session=request.getSession();
         if(session==null)
         {
            response.sendRedirect("session_design.html");
         }
         else
         {
            out.println("session is : "+session.getAttribute("s_nm"));
            out.print("<a href='session_servlet_logout'>click here to logout
   </a>");
         }
      }
   }
```

4. session_servlet_logout.java

Add package for use HttpSession import javax.servlet.http.HttpSession;

```
protected void processRequest(HttpServletRequest request,
   HttpServletResponse response)
         throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
         HttpSession session=request.getSession();
         if(session!=null)
         {
            session.invalidate();
            response.sendRedirect("session_design.html");
         }
      }
   }
```

- **Display Custom error page :**
- **File.html**

```
<form action="NewServlet123"> Cookie name : <input type="text"
   name="txt_ck"> <input type="submit" value="create cookie"> </form>
```

- **Web.xml**

```
<error-page>
      <error-code>404</error-code>
      <location>/error_404</location>
   </error-page>
```

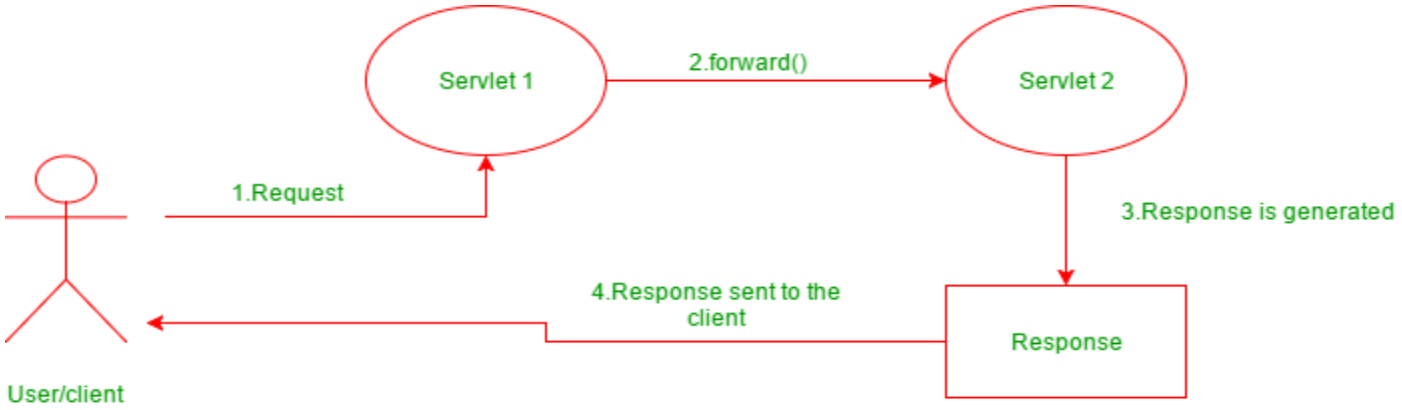
- **Error_404.java(servlet file)**

```
protected void processRequest(HttpServletRequest request,
   HttpServletResponse response) throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      try (PrintWriter out = response.getWriter()) {
                  out.println("<h1>Sorry</h1>");
}
   }
```

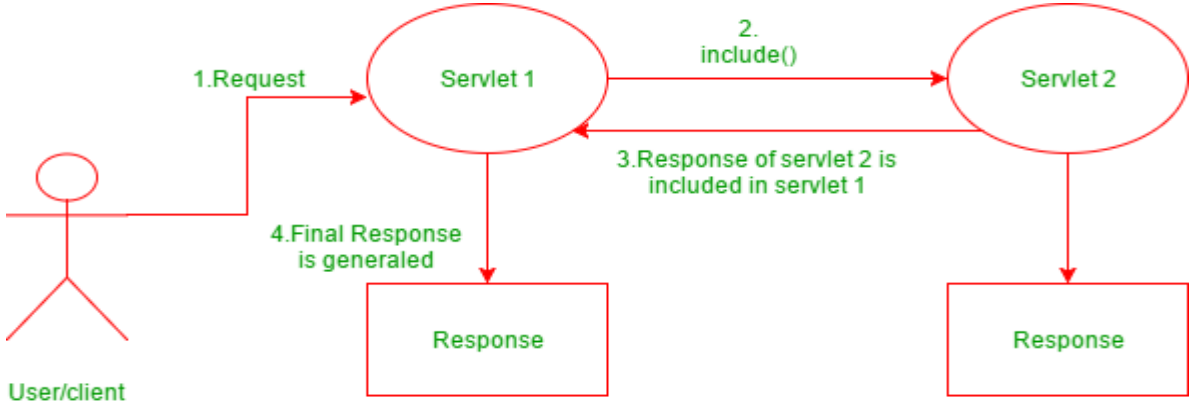
Servlet Collaboration:

- Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).
- The exchange of information among servlets of a particular Java web application is known as Servlet Collaboration.
- This enables passing/sharing information from one servlet to the other through method invocations.
- The servlet api provides two methodnamely:
- **1.getNamedDispatcher():**
- This method takes a string argument indicating the name of a servlet known to the ServletContext.
- If a servlet is known to the ServletContext by the given name.
- **2.getRequestDispatcher():**
- 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.

- The **RequestDispatcher interface** in Java is a helpful tool for the working of the servlet. It lets one servlet send a resource to another. This could be a servlet, a JSP, or HTML. You can do this in two main ways:
- **Forward:** When you make a request, it goes to another resource, the server. But's the thing—the original servlet's response won't be shown.



- **Include:** This way lets you take content from another resource and add it to the response of the current servlet.



File : index.java

```
<html>
<head>
<body>
  <form action="servlet1" method="post">
  Name:<input type="text"
  name="userName"/><br/>
  Password:<input type="password"
  name="userPass"/><br/>
  <input type="submit" value="login"/>
</form>
</body>
</html>
```

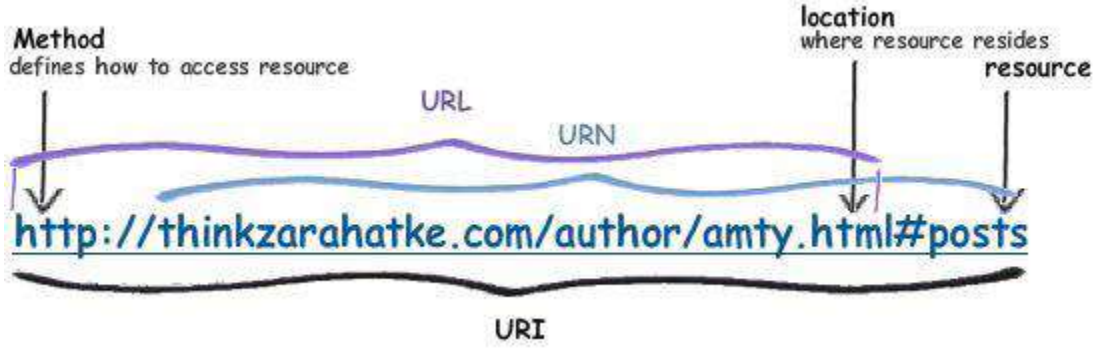
File : Servlet1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Login extends HttpServlet {
   public void doPost(HttpServletRequest req, HttpServletResponse res)  throws
    ServletException, IOException    {
      res.setContentType("text/html");
      PrintWriter out = response.getWriter();
    String n = request.getParameter("userName");
    String p = request.getParameter("userPass");
   if(p.equals("Thanos"){
         RequestDispatcher rd = request.getRequestDispatcher("servlet2");
    rd.forward(request, response);
   }
   else{
         out.print("Password mismatch");
         RequestDispatcher rd = request.getRequestDispatcher("index.html");
      rd.include(request, response);
    }
   }
}
```

File : Servlet2.java

```
// Called servlet in case password matches
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Welcome extends HttpServlet {

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

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      // fetches username
      String n = request.getParameter("userName");

      // prints the message
      out.print("Welcome " + n);
   }
}
```

- CGI : Common Gateway Interface
- URI (uniform resource identifier) identifies a resource (text document, image file, etc)
- URL (uniform resource locator) is a subset of the URIs that include a network location
- URN (uniform resource name) is a subset of URIs that include a name within a given space, but no location



Questions for Exam :

1. What is Naming Service?
2. Explain Servlet Life Cycle?
3. Write Servlet codde to start session when page is loaded with welcome guest message on screen.
4. Explain Request Dispatcher Interface.
5. Explain Servlet Collaboration.
6. Explain Single Thread Model.
7. Explain Cookie and Session.
8. Explain ServletAPI.
9. Name any three servlet API methods for session life time.
10. Explain Deployment Descriptor file.