Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

Servlets

Study

ARNIKA PATEL
CSE, PIT
Parul University

2.1 Basics of Web

A Web application is sometimes called a Web app, Thus, it is an application that is accessed using a Web browser over the network such as the Internet or an Intranet.

While many of the Web applications are written directly in PHP or Perl, Java remains a commonly used programming language for writing Web applications.

2.2 Servlet Lifecycle

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 work on the server side.
- Servlets are capable of handling complex requests obtained from the web server.
- Generate dynamic responses efficiently.

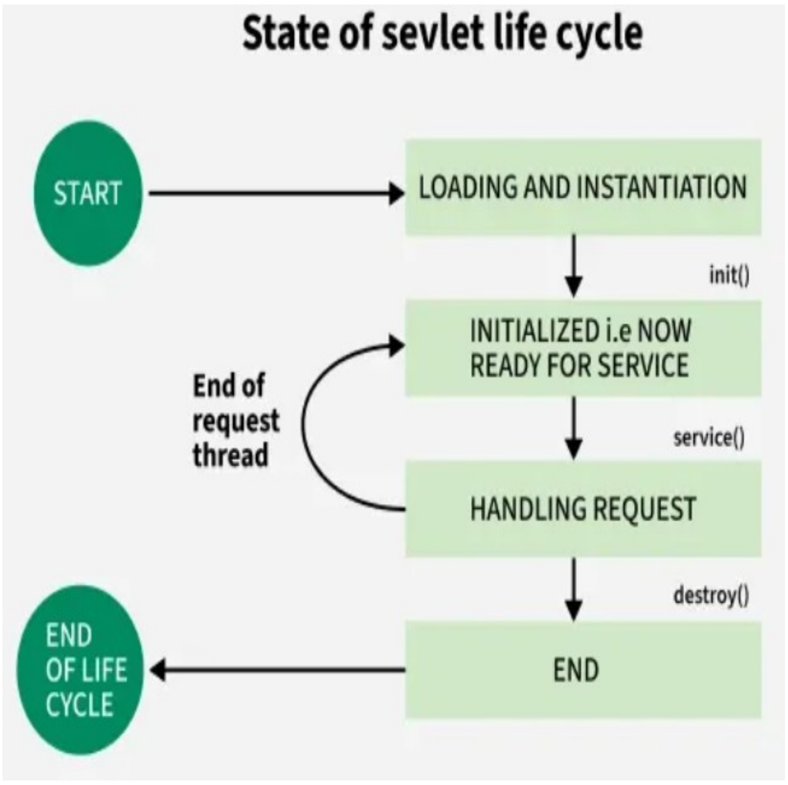

State of sevlet life cycle
START
LOADING AND INSTANTIATION
init()
INITIALIZED i.e NOW READY FOR SERVICE
End of request thread
service()
HANDLING REQUEST
destroy()
END
END OF LIFE CYCLE

Fig. 2.1 Servlet Lifecycle

The entire life cycle of a Servlet is managed by the Servlet container, which uses the **jakarta.servlet.Servlet** interface to understand the Servlet object and manage it. So, before creating a Servlet object, let's first understand the life cycle of the Servlet object, which actually understands how the Servlet container manages the Servlet object.

The Servlet life cycle mainly goes through four stages, which is explained below:

1. Loading a 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).

2. Initializing a Servlet

After the Servlet is instantiated, the Servlet container initializes it by calling the init(ServletConfig config) method. This method is called only once during the Servlet's life cycle.

3. Handling request

Once the Servlet is initialized, it is ready to handle client requests. The Servlet container performs the following steps for each request:

- **Create Request and Response Objects**
    - The container creates ServletRequest and ServletResponse objects.
    - For HTTP requests, it creates HttpServletRequest and HttpServletResponse objects.
- **Invoke the service() Method**
    - The container calls the service(ServletRequest req, ServletResponse res) method.
    - The service() method determines the type of HTTP request (GET, POST, PUT, DELETE, etc.) and delegates the request to the appropriate method (doGet(), doPost(), etc).

4. Destroying a Servlet

When the Servlet container decides to remove the Servlet, it follows these steps which are listed below

- **Allow Active Threads to Complete:** The container ensures that all threads executing the service() method complete their tasks.

- **Invoke the destroy() Method:** The container calls the destroy() method to allow the Servlet to release resources (e.g., closing database connections, freeing memory).
- **Release Servlet Instance:** After the destroy() method is executed, the Servlet container releases all references to the Servlet instance, making it eligible for garbage collection

Servlet Life Cycle Methods

There are three life cycle methods of a Servlet:

- init()
- service()
- destroy()

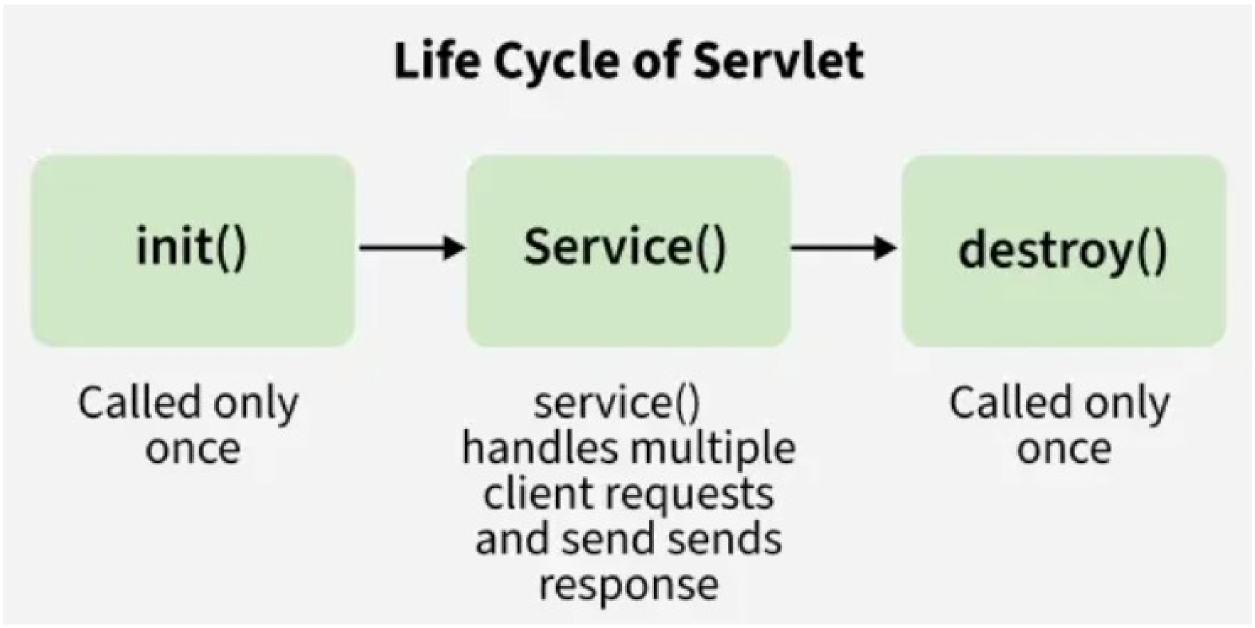


Fig. 2.2 Servlet Lifecycle Methods

Servlet life cycle can be defined as the stages through which the servlet passes from its creation to its destruction.

The servlet life cycle consists of these stages:

- Servlet is created
- Servlet is initialized
- Servlet is ready to service
- Servlet is servicing
- Servlet is not ready to service
- Servlet is destroyed

1. init() Method

This method is called by the Servlet container to indicate that this Servlet instance is instantiated successfully and is about to put into the service.

Illustration:

```
// init() method
public class MyServlet implements Servlet {
    public void init(ServletConfig config)
        throws ServletException
    {
        // initialization code
    }
        // rest of code
}
```

Important Key Points about init() Method

A Servlet life begins from this method. This method is called only once to load the servlet, Since it is called only once so the connected architecture code is written inside it because we only want once to get connected with the database

Approach 1: Using the Parameterized init(ServletConfig.con)

When the parameterized init(ServletConfig con) is called during the servlet life cycle, the following steps occur:

- First the overridden init(ServletConfig con) in our servlet class is executed.
- Then the init(ServletConfig con) method in the parent HttpServlet class is called using super.init(con).
- Finally the init() (non-parameterized version) in the parent HttpServlet class is invoked. However, this method has an empty body, making the call redundant.

```
public void init(ServletConfig con) throws ServletException
{
    super.init(con); // Calls HttpServlet's init(ServletConfig)
    // Custom database connection code here
}
```

Disadvantages:

- More Calls: Total 3 init() calls
    - Parameterized init(ServletConfig con) in our servlet class.
    - Parameterized init(ServletConfig con) in HttpServlet.
    - Non-parameterized init() in HttpServlet(empty body, hence useless)
- Performace Overhead: Extra calls increase stack usage and execution time.

Approach 2: Using the Non-Parameterized init()

- Instead of overriding the parameterized init(servletConfig con), override the non-parameterized init().
- During the servlet life cycle
    - The parameterized init(servletConfig con) in theHttpServlet class is called automatically.
    - This method then calls the non-parameterizedinit() method of our servlet directly.

```
public void init() throws ServletException
{
   // Custom database connection code here
}
```

Advantages:

- Fewer Calls: Total of 2 init() calls
    - Parameterized init(ServletConfig con) in HttpServlet.
    - Non-parameterized init() in our servlet class.

2. service() Method

This method is used to inform the Servlet about the client requests

- This method uses ServletRequest object to collect the data requested by the client
- This method uses ServletResponse object to generate the output content

Illustration:

```
// service() method
public class MyServlet implements Servlet {
   public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
   {
      // request handling code
   }
   // rest of code
```

}

Important Key Points about service() Method:

- This method provides the connection between client and server
- The web server calls the service() method to handle requests coming from the client(web browsers) and to send response back to the client
- This method determines the type of Http request (GET, POST, PUT, DELETE)
- This method also calls various other methods such as doGet(), doPost(), doPut(), doDelete()
- This method accepts two parameters.
    - req is the ServletRequest object which encapsulates the connection from client to server
    - res is the ServletResponse object which encapsulates the connection from server back to the client

3. destroy() Method

This method runs only once during the lifetime of a Servlet and signals the end of the Servlet instance

Syntax:

```
// destroy() method
public void destroy()
{
   // code
}
```

As soon as the destroy() method is activated, the Servlet container releases the Servlet instance

Important Key Points about destroy() Method:

- The destroy() method is called only once.
- It is called at the end of the life cycle of the servlet.
- This method performs various tasks such as closing connection with the database, releasing memory allocated to the servlet, releasing resources that are allocated to the servlet and other cleanup activities.
- After destroy() method the Servlet instance becomes eligible for garbage collection.

2.3 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,

- javax.servlet
- 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:

1. **GenericServlet:** To define a generic and protocol-independent servlet.
2. **ServletContextAttributeEvent:** To generate notifications about changes to the attributes of the servlet context of a web application.
3. **ServletContextEvent:** To generate notifications about changes to the servlet context of a web application.
4. **ServletInputStream:** This class provides an input stream to read binary data from a client request.
5. **ServletOutputStream:** This class provides an output stream for sending binary data to the client.
6. **ServletRequestAttributeEvent:** To generate notifications about changes to the attributes of the servlet request in an application.
7. **ServletRequestEvent:** To indicate lifecycle events for a ServletRequest.
8. **ServletRequestWrapper:** This class provides the implementation of the ServletRequest interface that can be subclassed by developers to adapt the request to a Servlet.
9. **ServletResponseWrapper:** This class provides the implementation of the ServletResponse interface that can be subclassed by developers to adapt the response from a Servlet.

Interfaces available in javax.servlet package:

1. **Filter:** To perform filtering tasks on either the request to a resource, or on the response from a resource, or both.
2. **FilterChain:** To provide a view into the invocation chain of a filtered request for a resource to the developer by the servlet container.
3. **FilterConfig:** To pass information to a filter during initialization used by a servlet container.

4. **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.
5. **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.
6. **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.
7. **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.
8. **ServletContextAttributeListener:** The classes that implement this interface receive notifications of changes to the attribute list on the servlet context of a web application.
9. **ServletRequest:** It defines an object that is created by servlet container to pass client request information to a servlet.
10. **ServletRequestAttributeListener:** To generate the notifications of request attribute changes while the request is within the scope of the web application in which the listener is registered.
11. **ServletRequestListener:** To generate the notifications of requests coming in and out of scope in a web component.
12. **ServletResponse:** It defines an object created by servlet container to assist a servlet in sending a response to the client.

Exceptions available in javax.servlet package:

1. **ServletException:** A general exception thrown by a servlet when it encounters difficulty.
2. **UnavailableException:** Thrown by a servlet or filter to indicate that it is permanently or temporarily unavailable.

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:

1. **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.
2. **HttpServlet:** Provides an abstract class that defines methods to create an HTTP suitable servlet for a web application.
3. **HttpServletRequestWrapper:** This class provides implementation of the HttpServletRequest interface that can be subclassed to adapt the request to a Servlet.
4. **HttpServletResponseWrapper:** This class provides implementation of the HttpServletResponse interface that can be subclassed to adapt the response from a Servlet.
5. **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.
6. **HttpSessionEvent:** To represent event notifications for changes to sessions within a web application.

Interfaces available in javax.servlet.http package:

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

GenericServlet Class

- Servlet API provide GenericServlet class in javax.servlet package.
- An abstract class that implements most of the servlet basic methods.

- Implements the Servlet, ServletConfig, and Serializable interfaces.
- Protocol-independent servlet.
- Makes writing servlets easier by providing simple versions of the lifecycle methods init() and destroy().
- To write a generic servlet, you need to extend ***javax.servlet.GenericServlet*** class and need to override the abstract service() method.

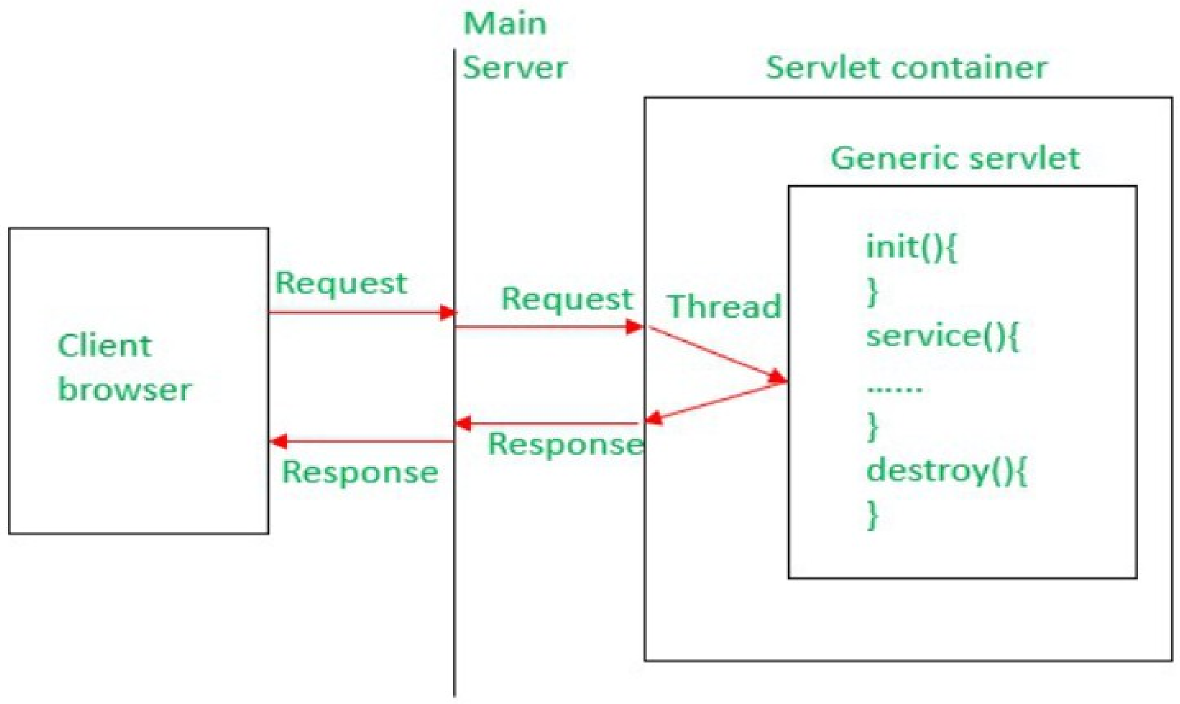


Fig. 2.3 Generic Servlet Class

HttpServlet Class

Servlet API provides HttpServlet class in javax.servlet.http package.

- An abstract class to be subclassed to create an HTTP-specific servlet that is suitable for a Web site/Web application.
- Extends Generic Servlet class and implements Serializable interface.
- HTTP Protocol-dependent servlet.

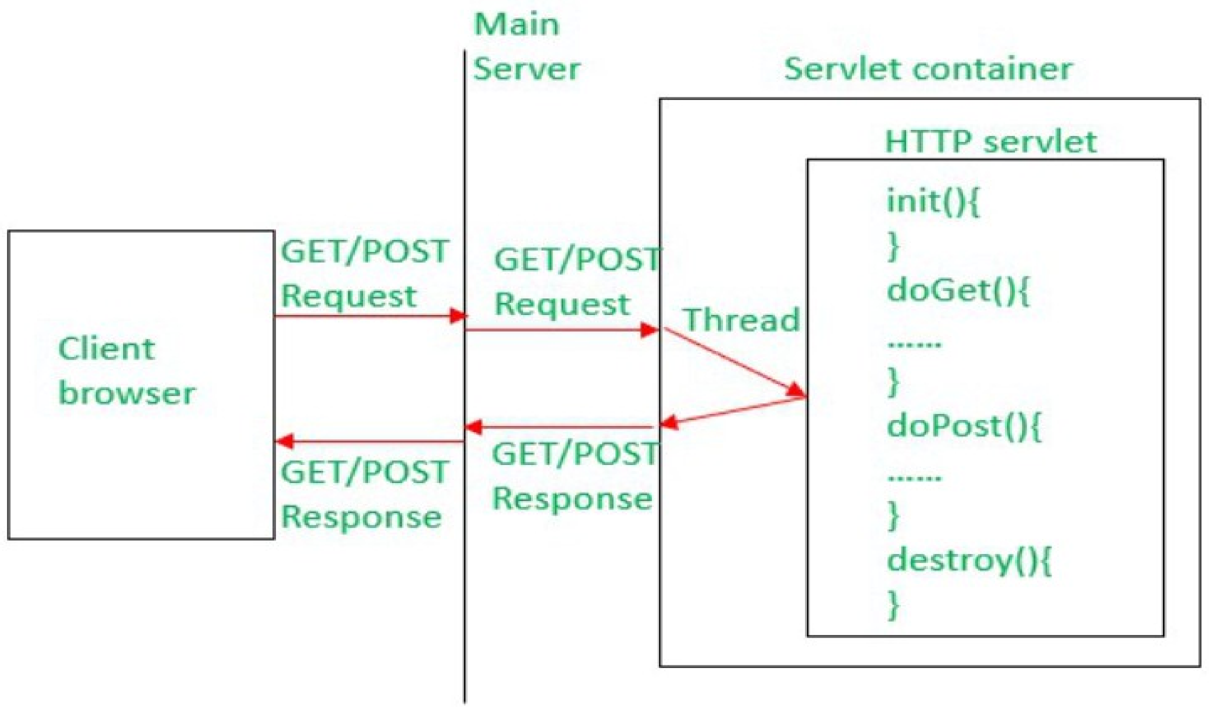


Fig. 2.4 HttpServlet Class

To write a Http servlet, you need to extend javax.servlet.http.HttpServlet class and must override at least one of the below methods,

- doGet() – to support HTTP GET requests by the servlet.
- doPost() – to support HTTP POST requests by the servlet.
- doPut() – to support HTTP PUT requests by the servlet.
- doDelete() – to support HTTP DELETE requests by the servlet.
- init() and destroy() – to manage resources held in the life of the servlet.
- getServletInfo() – To provide information about the servlet itself like the author of servlet or version of it etc.

Servlet API Package Hierarchy

Below is the package hierarchy of Generic Servlet and HTTP Servlet.

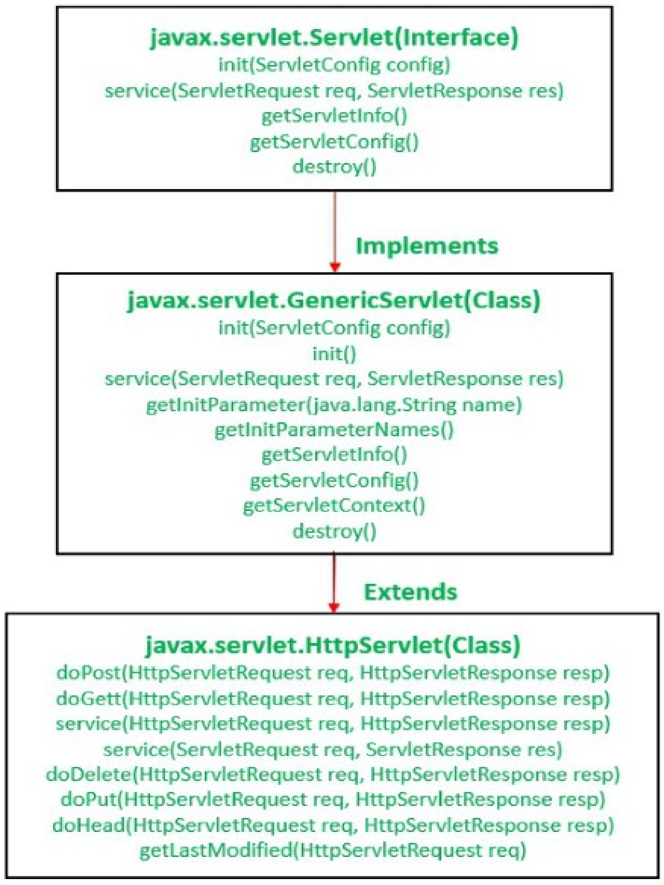

javax.servlet.Servlet(Interface)
init(ServletConfig config)
service(ServletRequest req, ServletResponse res)
getServletInfo()
getServletConfig()
destroy()
Implements
javax.servlet.GenericServlet(Class)
init(ServletConfig config)
init()
service(ServletRequest req, ServletResponse res)
getInitParameter(java.lang.String name)
getInitParameterNames()
getServletInfo()
getServletConfig()
getServletContext()
destroy()
Extends
javax.servlet.HttpServlet(Class)
doPost(HttpServletRequest req, HttpServletResponse resp)
doGett(HttpServletRequest req, HttpServletResponse resp)
service(HttpServletRequest req, HttpServletResponse resp)
service(ServletRequest req, ServletResponse res)
doDelete(HttpServletRequest req, HttpServletResponse resp)
doPut(HttpServletRequest req, HttpServletResponse resp)
doHead(HttpServletRequest req, HttpServletResponse resp)
getLastModified(HttpServletRequest req)

Fig. 2.5 Servlet API package Hierarchy

Servlet API provides all the required interfaces, classes, and methods to develop a web application. we need to add the servlet-api.jar file in our web application to use the servlet functionality. We can download this jar file from Maven Repository.

2.4 Servlet with Annotation

Whenever an HttpRequest comes from the client browser, the servlet container will map the request to the corresponding servlet based on the URL mappings provided in the deployment descriptor file – web.xml.

For example: Consider the below servlet mappings in the web.xml file.

```
<servlet>
<servlet-name>HelloServlet</servlet-name>
<servlet-class>HelloServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HelloServlet</servlet-name>
```

```
<url-pattern>/hello</url-pattern>
</servlet-mapping>
```

Here, whenever there is a "**/hello**" URL coming from the client request, we are mapping it to the "**HelloServlet**" class. Instead of providing these mappings in the **web.xml** file, we can simply provide an annotation in the Servlet as below,

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
	// Code to be executed...
}
```

Java Servlets provides a number of annotations to allow users to declare servlets, filters, listeners in the **javax.servlet.annotation** package. It also provides annotations to specify metadata for the components that are declared.

Annotation Type WebServlet:

public @interface WebServlet

- Available in javax.servlet.annotation package.
- Used to declare a Servlet.
- At the time of deployment, the servlet container will identify and process the annotation and makes the corresponding servlet available at the specified URL patterns.

Servlet with Annotation Examples

we will create a simple HTML page to map the Servlet with the URL.

Example 1: Servlet annotated with only URL pattern:

index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Home</title>
</head>
<body>
	<form action="hello" method="post">
		Welcome Page: <input type="submit" />
	</form>
</body>
</html>
```

HelloServlet.java

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
	ServletException, IOException
	{
		// set the response content type
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();
		// Print hello message to the client browser in
		// response object
		out.println( "<h3>Hello, Welcome to parul University!!</h3>");
		out.close();
	}
}
```

- When user clicks submit for the welcome page, container will process the "**@WebServlet("/hello")**" annotation and maps the "**HelloServlet**" class.
- As the method is mentioned as "**post**" in Html page, container will execute "**doPost()**" method in "**HelloServlet**" class.

2.5 Servlet Configuration

javax.servlet.ServletConfig is an interface as a part of servlet API. For every Servlet class in our application, the web container will create one ServletConfig object and the web container will pass this object as an argument to the **public void init(ServletConfig config)** method of our Servlet class object.

Some of the important points on ServletConfig are:

- ServletConfig is an object containing some initial parameters or configuration information created by the Servlet Container and passed to the servlet during initialization.

- ServletConfig is for a particular servlet, which means one should store servlet-specific information in web.xml and retrieve it using this object.

Methods in the ServletConfig Interface

1. public abstract java.lang.String **getServletName()**
2. public abstract javax.servlet.ServletContext **getServletContext()**
3. public abstract java.lang.String **getInitParameter(java.lang.String)**
4. public abstract java.util.Enumeration<java.lang.String> **getInitParameterNames()**

2.6 Servlet Context

ServletContext is the object created by Servlet Container to share initial parameters or configuration information to the whole application.

Example: Suppose, the name of one's job portal is "NewWebsite.tg".

Showing the website name at the top of webpages delivered by different servlets, one needs to store the website name in every servlet inviting redundancy.

Since the information shared by ServletContext can be accessed by every Servlet, it is better to go with ServletContext and retrieve the website name using getServletContext.getInitParameter("Name") whenever required.

Table 2.1 servlet config. Vs context

Servlet Configuration	**Servlet Context**
ServletConfig is servlet specific	ServletContext is for whole application
Parameters of servletConfig are present as name-value pair in <init-param> inside <servlet>	Parameters of servletContext are present as name-value pair in <context-param> which is outside of <servlet> and inside <web-app>
ServletConfig object is obtained by getServletConfig() method.	ServletContext object is obtained by getServletContext() method.
Each servlet has got its own ServletConfig object.	ServletContext object is only one and used by different servlets of the application.
Use ServletConfig when only one servlet needs information shared by it.	Use ServletContext when whole application needs information shared by it

2.7 Servlet Collaboration

- 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.

Using RequestDispatcher Interface

- The RequestDispatcher interface provides the option of dispatching the client's request to another web resource, which could be an HTML page, another servlet, JSP etc. It provides the following two methods:

- public void forward(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException:

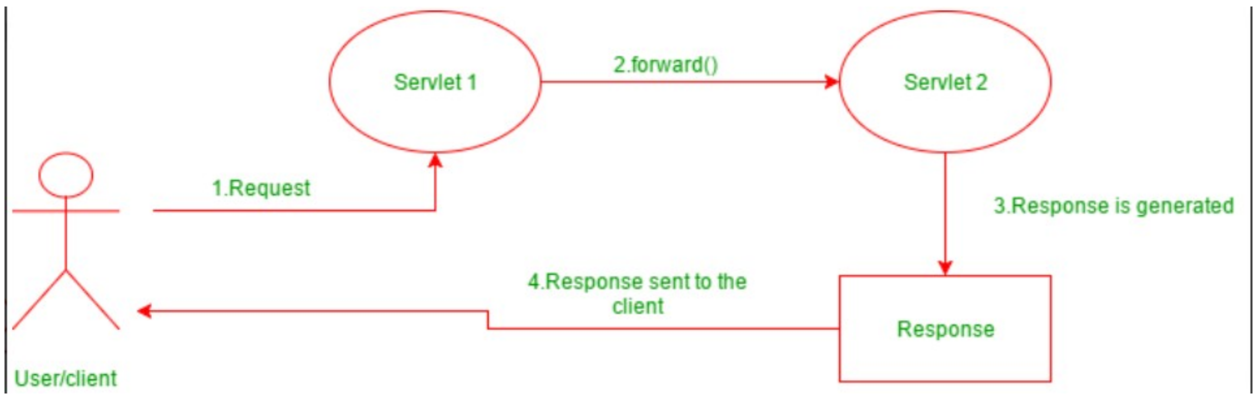


Fig. 2.6 Servlet Collaboration forward

public void include(ServletRequest request, ServletResponse response)throws ServletException, java.io.IOException:

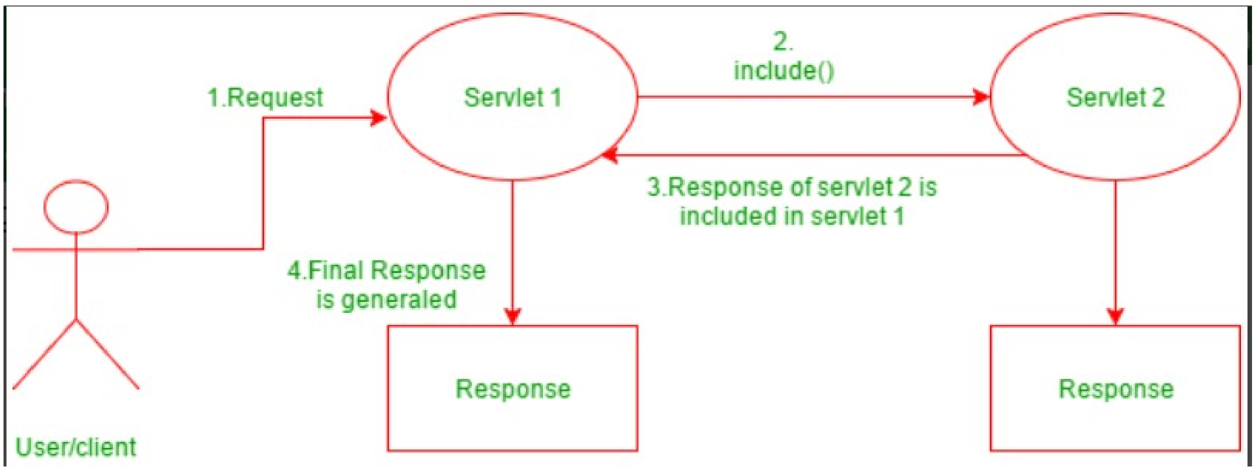

1.Request
Servlet 1
2. include()
Servlet 2
3.Response of servlet 2 is included in servlet 1
4.Final Response is generated
Response
Response
User/client

Fig. 2.7 Servlet Collaboration include

The include() method is used to include the contents of the calling resource into the called one.

When this method is called, the control still remains with the calling resource. It simply includes the processed output of the calling resource into the called one.

Using HttpServletResponse Interface

The HttpServletResponse interface is entrusted with managing Http responses. To achieve servlet collaboration, it uses the following method:

public void sendRedirect(String URL)throws IOException;

This method is used redirect response to another resource, which may be a servlet, jsp or an html file. The argument accepted by it, is a URL which can be both, absolute and relative. It works on the client side and uses the browser's URL bar to make a request.

Although the two methods appear to do the same thing, there are still differences between the two, which are as follows:

Table 2.2 forward. Vs sendRedirect

forward()	**sendRedirect()**
It works on the server side	It works on the client side
It sends the same request and response objects to another resource.	It always send a new request
It works only within the server.	It can be used within and outside the server.

2.8 Servlet Session Tracking

- 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
- HTTP is a "stateless" protocol, which means that each time a client requests a Web page, the client establishes a new connection with the Web server, and the server does not retain track of prior requests.
- The conversation of a user over a period of time is referred to as a **session**. In general, it refers to a certain period of time.
- The recording of the object in session is known as **tracking**.
- **Session tracking** is the process of remembering and documenting customer conversations over time. Session management is another name for it.
- The term "**stateful web application**" refers to a web application that is capable of remembering and recording client conversations over time.

Why is Session Tracking Required?

- Because the HTTP protocol is stateless, we require Session Tracking to make the client-server relationship stateful.

- Session tracking is important for tracking conversions in online shopping, mailing applications, and E-Commerce applications.
- The HTTP protocol is stateless, which implies that each request is treated as a new one. As you can see in the image below.

Deleting Session Data

- Remove a specific attribute You can delete the value associated with a specific key by calling the public void removeAttribute(String name) function.
- Delete your whole session. To delete an entire session, use the public void invalidate() function.
- Setting Session Timeout You may set the timeout for a session separately by calling the public void setMaxInactiveInterval(int interval) function.
- Log the user out On servers that support servlets 2.4, you may use the logout method to log the client out of the Web server and invalidate all of the users' sessions.
- web.xml Configuration If you're using Tomcat, you may set the session timeout in the web.xml file, in addition to the ways listed above

```
<session-config>
<session-timeout>20</session-timeout>
</session-config>
```

2.9 Session Tracking employs Four Different techniques

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

A. Cookies

Cookies are little pieces of data delivered by the web server in the response header and kept by the browser. Each web client can be assigned a unique session ID by a web server. Cookies are used to keep the session going. Cookies can be turned off by the client.

B. Hidden Form Field

The information is inserted into the web pages via the hidden form field, which is then transferred to the server. These fields are hidden from the user's view.

Illustration:

```
<input type = hidden'  name = 'session' value = '12345' >
```

C. URL Rewriting

With each request and return, append some more data via URL as request parameters. URL rewriting is a better technique to keep session management and browser operations in sync.

D. HttpSession

A user session is represented by the HttpSession object. A session is established between an HTTP client and an HTTP server using the HttpSession interface. A user session is a collection of data about a user that spans many HTTP requests.

Illustration:

```
HttpSession session = request.getSession( );
Session.setAttribute("username", "password");
```

The request must be made. Before sending any document content to the client, you must first call getSession(). The following is a list of the most significant methods provided by the HttpSession object:

Table 2.3 HttpSession methods

Method	Description
public Object getAttribute(String name)	This method returns the object in this session bound with the supplied name, or null if no object is bound with the name.
public Enumeration getAttributeNames()	This function returns an Enumeration of String objects with the names of all the items associated with this session.
public long getCreationTime()	This method returns the milliseconds since midnight January 1, 1970 GMT, when this session was created.
public String getId()	This function returns a string that contains the session's unique identification.
public long getLastAccessedTime()	This function returns the session's most recent accessible time in milliseconds since midnight on January 1, 1970 GMT.
public int getMaxInactiveInterval()	The maximum time interval (seconds) for which the servlet container will keep the session open between client requests is returned by this function.
public void invalidate()	This function unbinds any objects connected to this session and invalidates it.
public boolean isNew()	If the client is unaware of the session or decides not to join it, this function returns true.
public void removeAttribute(String name)	The object bound with the supplied name is removed from this session using this method.
public void setAttribute(String name, Object value)	This function uses the supplied name to tie an object to this session.
public void setMaxInactiveInterval(int interval)	This function defines the interval between client requests before the servlet container invalidates this session in seconds.

References:

1. **Book Reference**

   Jim Farley, William Crawford, David Flanagan. Java Enterprise in a Nutshell, O'Reilly
2. **Book Reference**

   Rocha, R., Purificação, J. (2018). Java EE 8 Design Patterns and Best Practices: Build Enterprise-ready Scalable Applications with Architectural Design Patterns. Germany: Packt Publishing..
3. **Website Reference**

   https://www.scribd.com/document/268349254/Java-8-Programming-Black-Book .
4. **Sources**

   https://developers.redhat.com/topics/enterprise-java
5. **Article**

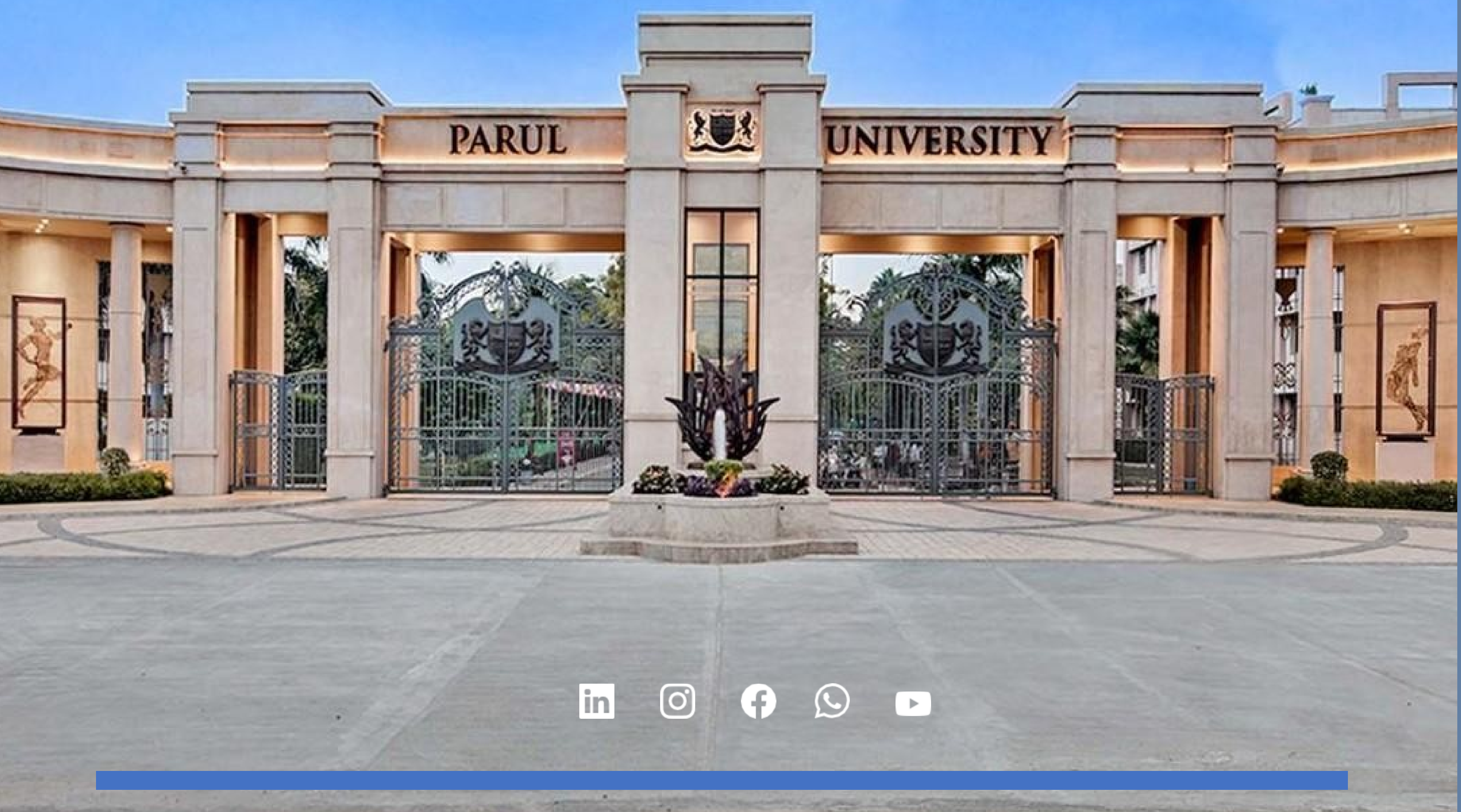
   https://www.researchgate.net/publication/276412369_Advanced_Java_Programming

Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and
Communication Technology

NAAC GRADE A++



https://paruluniversity.ac.in/