Involves **two separate HTTP requests**:
1. The first request (POST/GET) goes to the original servlet.
2. The second is a new GET request made by the client to the redirected URL. Java Servlets are Java programs run on a Java-enabled web server or application server. They handle the request obtained from the web server, process the request, produce the response, and then send a response back to the web server.

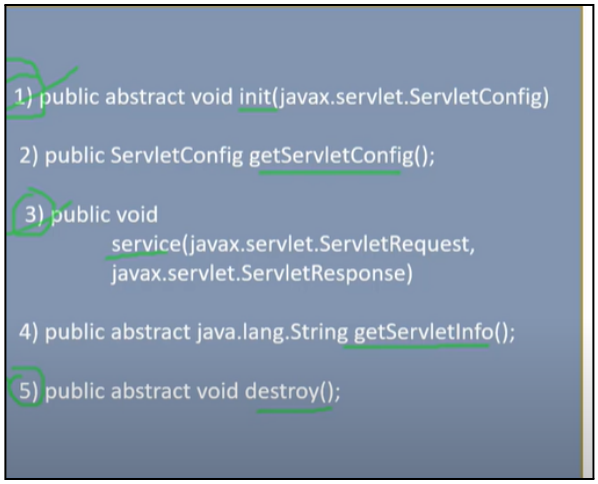
Properties of Java Servlet
The properties of Servlets are as follows:

Servlets work on the server side.
Servlets are capable of handling complex requests obtained from the web server.

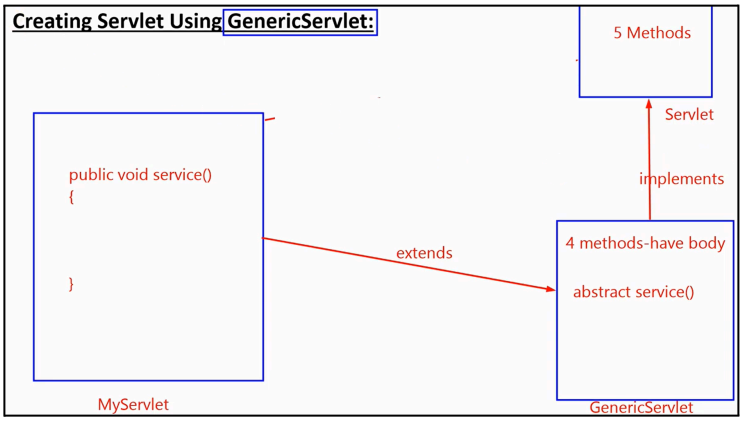
For more: - https://www.geeksforgeeks.org/introduction-java-servlets/

For the javax servlet, we will be implementing it from the **Servlet** interface.

We have 5 methods



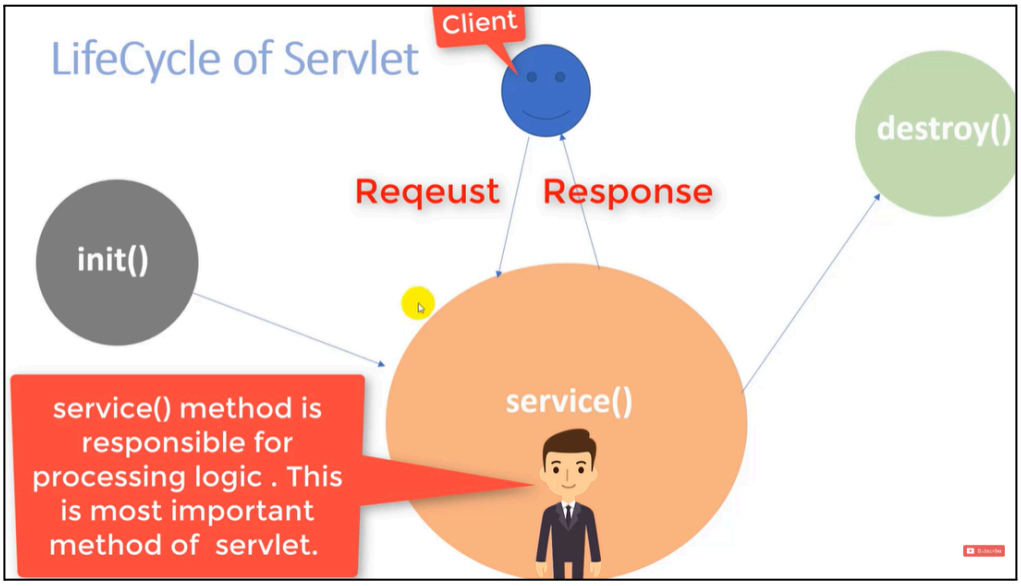
init() , service() , destroy() are the lifecycle methods



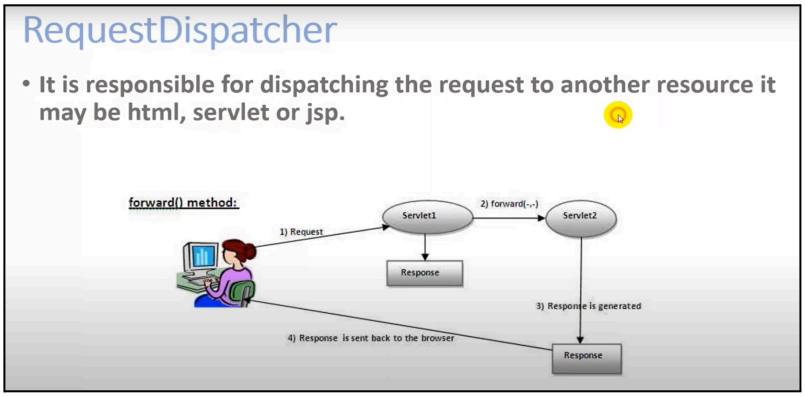
Generic servlet implements Servlet, it doesn't define service(), it's still an abstract class. We will have to provide an implementation for service() when extending GenericServlet.

LifeCycle of Servlet

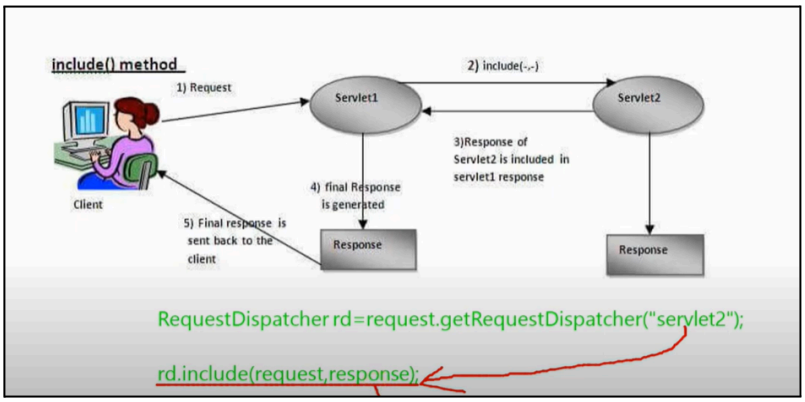
When the first request comes, the server creates a servlet object.
The server calls init() for servlet initialisation.
service() is called to process the logic.
To release resources destroy() is called.



RequestDispatcher is used to forward the request



It also has another important function, include() method

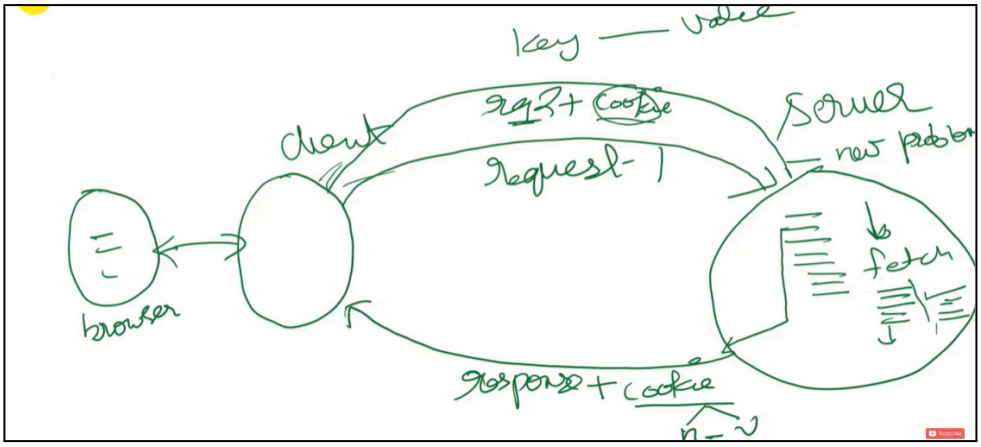


Session tracking is a way to maintain a user's state(data), also known as state management.
HTTP is a stateless protocol.
Stateless protocol means each request is treated as a new request.
Techniques used in session tracking:

1. Cookies
2. Url rewriting
3. Hidden form field
4. HttpSession

Example :- If you pass name in an input field from s1 to s2 , then try req.getParameter(name) in s2 , it returns null

COOKIES:
Cookies are textual info stored in key-value pairs format to the client's browser during multiple requests.

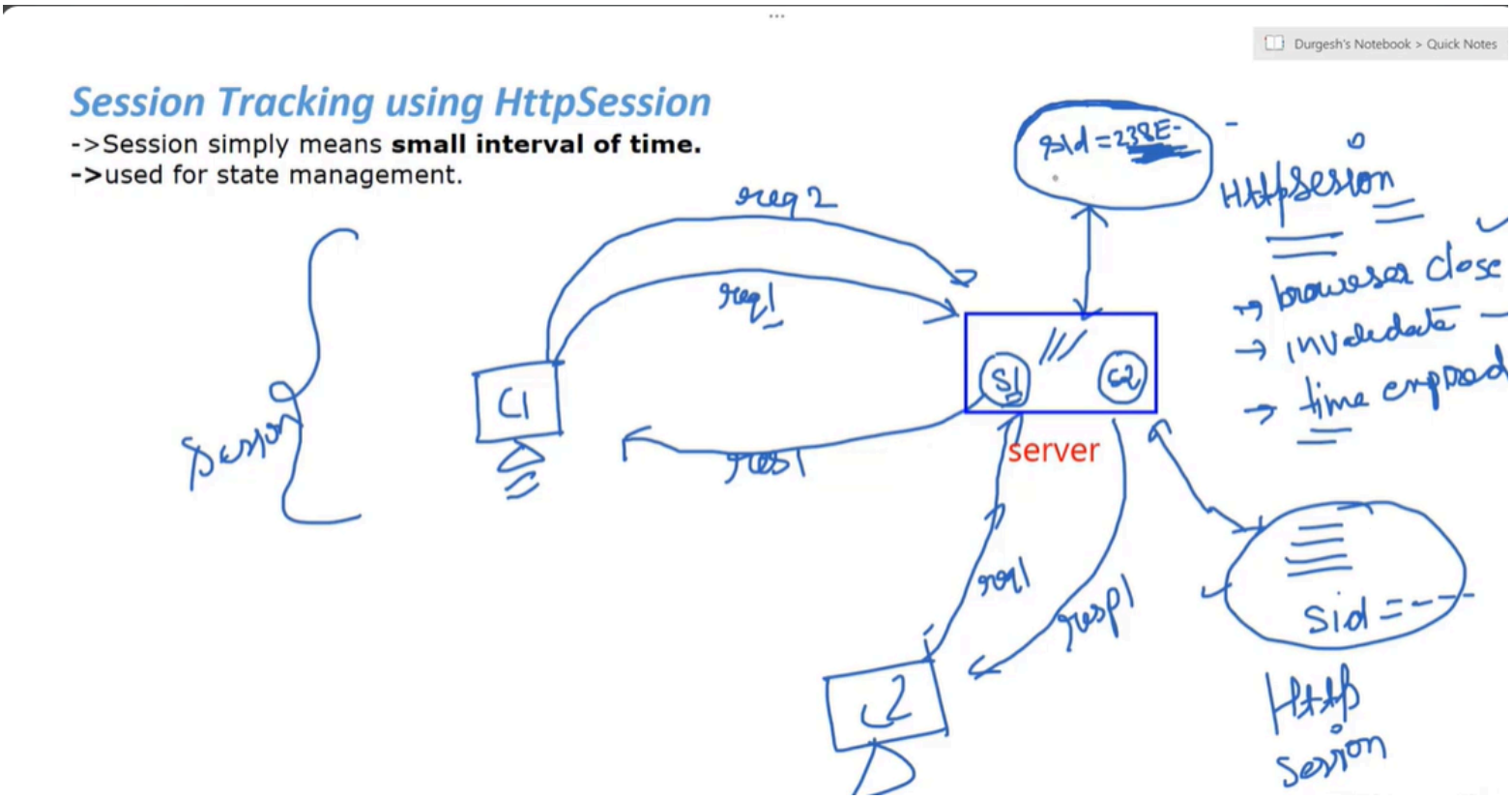


URL Rewriting:
Url rewriting is appending or modifying any URL while loading a page.

www.iweewnewe.com/t.jsp/?name=John?userNo=4
This happens from the client's side
You can make a request.getParameter in the servlet class

Hidden form:
Simply submitting stuff as a hidden field, and retrieving it in the next servlet

HttpSession:



HttpSession is an interface.
For every new client, the server creates a new session. The client can keep querying the session as long as it is active.
The session is active until

1. The browser is not closed
2. The session is not invalidated (session.invalidate()).
3. Session timeout hasn't occurred.

From the session, you can use setAttribute(String key, Object o), and then do getAttribute(key)
We have other methods getId(), removeAttribute(key), invalidate().
We have this request: getSession(). Then, we can pass a boolean true/false. True forces a new session to be created.
Refer to HttpSession documentation, there is a lot:)

Why do we need session tracking when we already have request.setAttribute() ?

Using `request.setAttribute` works only within the scope of a single HTTP request. If you set an attribute on the request, it is available only for that single request-response cycle. This means that any data saved in `request` attributes is lost after the response is sent to the client. If you need to retain data across multiple requests from the same client, you need session tracking mechanisms. Here's why each option is relevant:

1. **Cookies**: Cookies are stored on the client side and allow the server to identify returning users, enabling it to retrieve user-specific information across requests. They are useful for simple session tracking but have security and storage limitations.
2. **URL Rewriting**: URL rewriting appends session data to the URL. It works even if cookies are disabled but can make URLs messy and limit the data that can be stored.
3. **Hidden Form Fields**: Hidden form fields allow data to be passed between requests via forms. However, this only works with POST requests (or GET with query parameters) and is limited in scope to form-based navigation.
4. **HttpSession**: `HttpSession` is the most versatile server-side solution for session tracking. It stores data on the server and associates it with a unique session ID. This session ID is sent to the client via cookies or URL rewriting, allowing the server to manage user sessions securely and efficiently across multiple requests.

In summary, while `request.setAttribute` is suitable for single-request storage, session-tracking concepts are necessary for maintaining state across multiple requests from the same client.

Let's go through an example to illustrate why `request.setAttribute` won't work across multiple requests.

Scenario: Simple Login Example

Imagine we have a login page where a user enters their username. After logging in, we want to display a welcome message with the username on multiple pages.

Code Example (without Session Tracking)

Login Servlet: Sets the username using `request.setAttribute` and forwards to a welcome page.
java

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String username = request.getParameter("username");
    request.setAttribute("username", username);  // Set attribute only for this request
    RequestDispatcher dispatcher = request.getRequestDispatcher("welcome.jsp");
    dispatcher.forward(request, response);
}
```

Welcome Page (welcome.jsp): Displays the username.

```
<h1>Welcome, ${username}!</h1>
<a href="profile">Go to Profile Page</a>
```

Profile Servlet: The user clicks "Go to Profile Page," which makes a new request to the Profile Servlet. Here, we try to access the "username" attribute.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String username = (String) request.getAttribute("username");
    // Try to retrieve the username
    if (username == null) {
        username = "Guest";  // Since request.setAttribute is lost after the first
request
    }
    request.setAttribute("username", username);
    RequestDispatcher dispatcher = request.getRequestDispatcher("profile.jsp");
    dispatcher.forward(request, response);
}
```

Profile Page (profile.jsp): Displays the username.

```
<h1>Your Profile, ${username}</h1>
```

Result of Using `request.setAttribute`

When you first log in, the `username` is set using `request.setAttribute`, and it is available on the `welcome.jsp` page. However, when you navigate to the profile page, a new request is made to the server. The data from the previous request (e.g., `username`) is not carried over because `request.setAttribute` only holds data for a single request.

Profile page output: The profile page will not have access to the `username` set in the previous request. Instead, it will show `"Guest"` or no username since the attribute is not available beyond the initial request.

What is the difference between sendRedirect vs forward?

Feature	sendRedirect()	forward()
Number of Requests	Involves **two separate HTTP requests**: 1. The first request (POST/GET) goes to the original servlet. 2. The second is a new GET request made by the client to the redirected URL.	Involves **only one request**. The original request is simply forwarded to another resource internally.
Client vs. Server-Side Request	- Tells the client (browser) to make a new request to a different URL. - Client-side redirect; the browser's URL changes to the new location. - Sends a 302 HTTP status code and a new URL in the `Location` header to the client, prompting the browser to initiate a new GET request.	It happens entirely on the server side without involving the client. - The URL in the browser remains the same. - Control is transferred to another server resource (e.g., JSP, servlet) for further processing. No new HTTP request is made; the same request and response objects are passed to the new resource.
URL in Browser	The browser's URL **changes** to the new redirected location.	The browser's URL **remains the same** as the forward happens on the server, and the client is unaware of it.
Scope of Request and Response Objects	Since a new request is initiated, **new request and response objects** are created. - Any data (like attributes) set in the original request will not be available in the new request unless passed via query string or session.	- The **same request and response objects** are used. - Any data (attributes) set in the original request are available to the forwarded resource.
Performance Impact	Slightly slower due to an extra round-trip between the client and server (two requests).	More efficient as it happens entirely on the server side without an additional HTTP request.
Use cases	- Best used when redirecting to another domain, servlet, or resource, possibly outside the current application. - Useful when the client should be aware of the new URL (e.g., redirecting after form submission to prevent resubmission). - Common for the Post/Redirect/Get pattern.	- Used when passing control to another resource (e.g., JSP, servlet) without client knowledge. - Suitable for server-side processing within the same web application. - Often used for internal processing (e.g., forwarding from a servlet to a JSP to generate the final HTML response)

https://www.geeksforgeeks.org/introduction-to-jsp/

Finish JSP tut -> Hibernate vid 18 (project pure java) -> spring-data-jpa -> spring-core -> Todo part 1-> Todo part 2

For TechBlog project refer following links
https://getbootstrap.com/docs/4.0/
https://www.w3schools.com/icons/fontawesome_icons_webapp.asp
https://bennettfeely.com/clippy/
https://sweetalert.js.org/?utm_source=cdnjs&utm_medium=cdnjs_link&utm_campaign=cdnjs_library