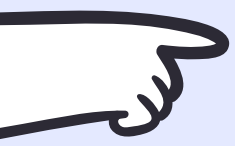


SERVLETS

Exploring the Foundations

Rohan Thapa

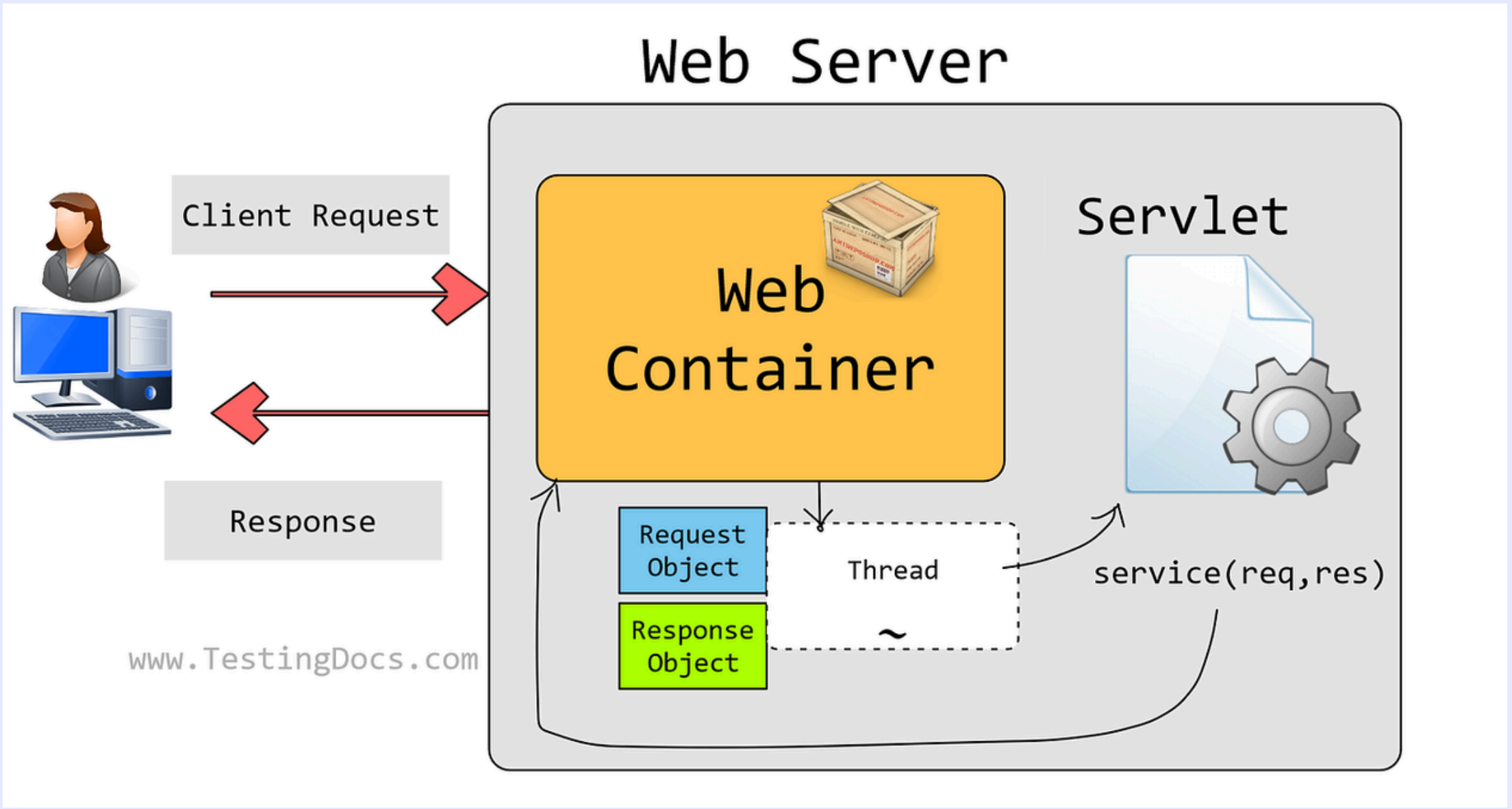
thaparohan2019@gmail.com

What is a Servlet?

- **Definition:**
  - A servlet is a small Java program that runs within a Web server.
  - A Servlet is a Java class that extends the **capabilities of servers** that host applications accessed by means of a **request-response** programming model. Servlets are typically used to process or store a Java object and manage HTTP requests and responses in a web application.

- **Part of Java EE:** Servlets are a part of the **Java Enterprise Edition (Java EE)** and are primarily used to create dynamic web content.

Diagram


Web Server
Client Request
Response
Web
Container
Servlet
Request
Object
Response
Object
Thread
service(req,res)
www.TestingDocs.com

Why use Servlets?

- **Dynamic Content Generation:** Servlets allow you to generate dynamic content (like **HTML pages, JSON data**) based on **user input** or **server-side logic**.
- **Scalability:** Servlets are capable of handling **multiple requests simultaneously**, making them highly scalable for web applications.
- **Security:** Servlets provide **built-in security** features like **session management**, **HTTPS** support, and **authentication**.
- **Portability:** Since Servlets are written in Java, they are portable across different platforms and servers that support the Servlet specification.

Servlet Lifecycle

Loading and Instantiation

- The Servlet class is loaded into memory by the web container (like **Tomcat**).
- The container creates an instance of the Servlet.

Initialization (init method)

- The container calls the init() method once when the Servlet is first loaded.
- This method is used to perform any required initialization, like resource allocation.

Servlet Lifecycle

Request Handling (service method)

- For each client request, the container spawns a new thread and calls the **service()** method.
- The **service()** method determines the request type **(GET, POST, etc.)** and dispatches it to the corresponding method **(doGet, doPost, etc.).**

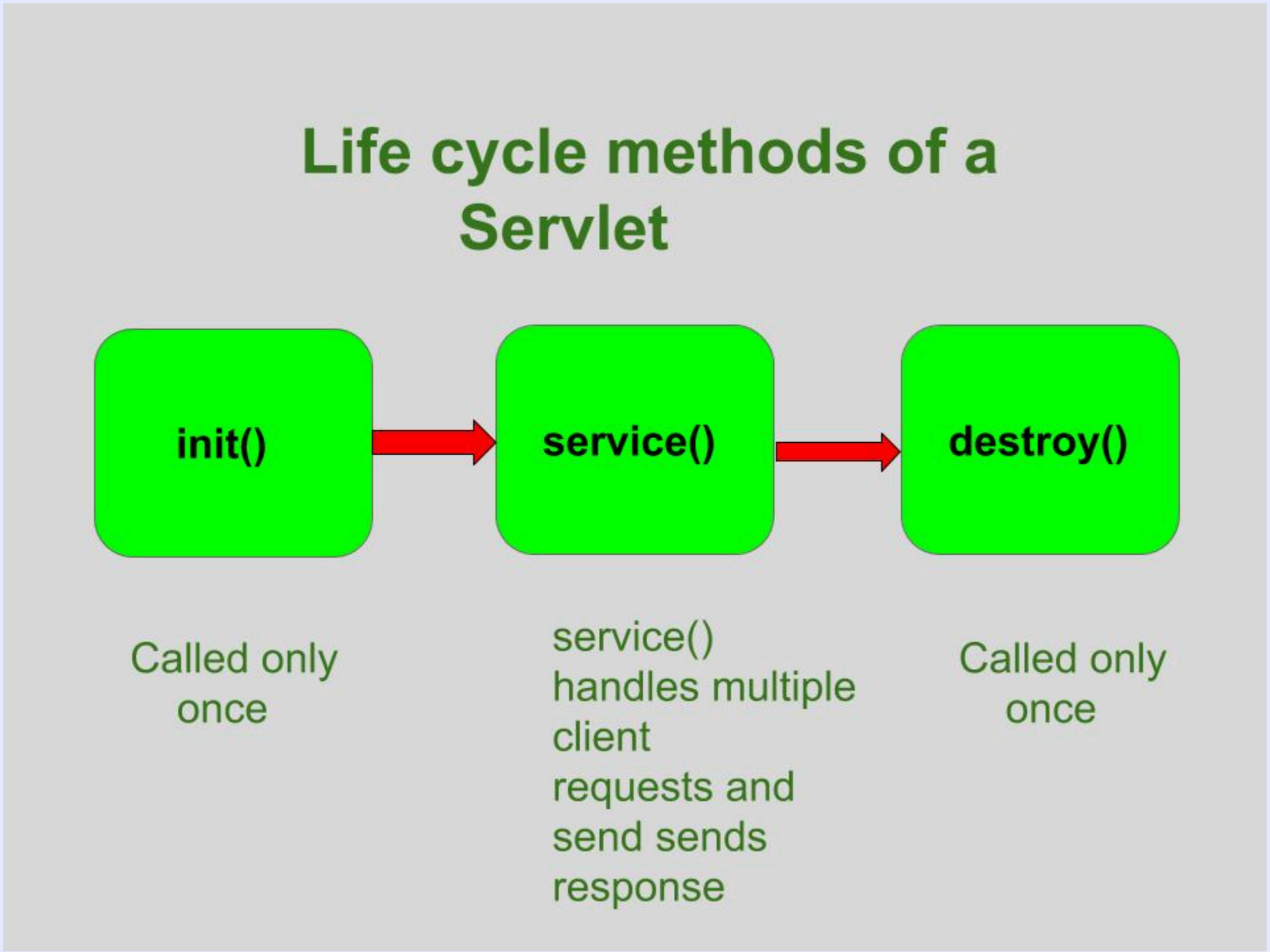
Destruction (destroy method)

- When the Servlet is no longer needed, the container calls the destroy() method.
- This method is used to release resources and perform cleanup tasks.

Garbage Collection

- After the destroy() method is called, the Servlet object becomes eligible for garbage collection.

Servlet Lifecycle



Creating a Servlet by Implementing the Servlet Interface

This is the most basic way to create a Servlet. You directly implement the **Servlet interface** and define all **five lifecycle methods**: **init, service, destroy, getServletConfig,** and **getServletInfo**.

```java
public class MyServlet implements Servlet {

    @Override
    public void init(ServletConfig config) throws ServletException {
        // Initialization code
    }

    @Override
    public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<h1>Hello, World from Servlet!</h1>");
    }

    @Override
    public void destroy() {
        // Cleanup code
    }

    @Override
    public ServletConfig getServletConfig() {
        return null;
    }

    @Override
    public String getServletInfo() {
        return "MyServlet";
    }
}
```

Creating a Servlet by Implementing the Servlet Interface

Web Deployment Descriptor **(web.xml)**:

```
<servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.example.MyServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>MyServlet</servlet-name>
    <url-pattern>/myservlet</url-pattern>
</servlet-mapping>
```

Creating a Servlet by Extending HttpServlet

Most commonly, **Servlets** are created by extending the **HttpServlet class**, which provides default **implementations** for **doGet**, **doPost**, and **other HTTP-specific methods**. You only need to override the methods that your application requires.

```java
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<h1>Hello, World from HttpServlet!</h1>");
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        // Handle POST request
    }
}
```

Creating a Generic Servlet by Extending GenericServlet

GenericServlet is an abstract class that **implements** the **Servlet interface** and **provides simple implementations** for all methods except **service().** You can extend GenericServlet and implement the **service()** method to create your own Servlet.

```java
public class MyGenericServlet extends GenericServlet {

    @Override
    public void service(ServletRequest req, ServletResponse res)
            throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<h1>Hello from GenericServlet!</h1>");
    }
}
```

Key Components and Concepts

HttpServletRequest and HttpServletResponse

- **HttpServletRequest:** Represents the client's request, encapsulating all the information sent by the client, such as **form data, headers, cookies,** etc.
- **HttpServletResponse:** Represents the response that the server sends back to the client, including the **status code, content type**, and the actual content **(like HTML, JSON).**

Key Components and Concepts

ServletContext and ServletConfig

- **ServletContext:** Represents the web application's environment. It allows Servlets to share information and communicate within the same application.
    - Example: Shared resources like a database connection pool can be stored in the ServletContext.
- **ServletConfig:** Provides initialization parameters for a specific Servlet. It is passed to the Servlet during initialization (init method).

Key Components and Concepts

Request Dispatcher

- **Forwarding Requests:** The **RequestDispatcher** interface allows a request to be forwarded to another resource (**Servlet, JSP, HTML file).**
- **Including Content**: You can also include the content of another resource in the response using the **RequestDispatcher.**

Key Components and Concepts

Session Management

- **HttpSession:** Java Servlets provide a way to track user sessions across multiple requests using the **HttpSession interface**.
    - **Example:** Storing user login information in the session to maintain the state across different pages.

Filters

- **Pre-processing** and **Post-processing**: Filters are used to intercept requests and responses to perform tasks like logging, authentication, data compression, etc.
    - Filters are configured in the web.xml file or using annotations and can be applied to specific URL patterns.

Advanced Servet Concepts

1. Asynchronous Servlets

- **Background Processing:** Servlets can handle long-running tasks without blocking the main thread using asynchronous processing **(@WebServlet(asyncSupported = true)).**
    - **Example:** Useful for applications that involve file uploads, long database operations, or external service calls.

2. WebSockets in Servlets

- **Real-time Communication:** Servlets can be used to handle WebSocket connections, allowing for full-duplex communication between the server and the client.

Advanced Servet Concepts

3. Annotations in Servlets

- **Simplified Configuration:** With the introduction of annotations **(@WebServlet, @WebFilter, etc.),** configuring Servlets has become much easier, eliminating the need for extensive **web.xml** configuration.

4. Security Features

- **Declarative Security:** Security constraints can be declared in web.xml, specifying roles and access controls for different URL patterns.
- **Programmatic Security:** Servlets can programmatically manage security aspects like user authentication and access control.

Servlets vs. JSP

- **Role:** Servlets primarily act as controllers or request processors, while JSPs are used for the view layer to generate dynamic content.

- **Complexity:** Writing complex HTML in Servlets can be cumbersome; JSPs simplify this by embedding Java code directly into the HTML.

- **Performance:** Since JSPs are eventually **compiled into Servlets**, there's no significant performance difference, but Servlets might be preferred for pure request handling.

Servlets in Modern Development

- **Foundational Technology:** Despite the rise of frameworks like **Spring Boot**, understanding Servlets is crucial as they form the foundation for many web technologies in the Java ecosystem.
- **Integration with Spring Boot:** In Spring Boot, while you might not write Servlets directly, the core concepts like **request handling, session management**, and **filters still** rely on Servlet principles.

Thank You

Rohan Thapa

thaparohan2019@gmail.com