

Spring Web – Servlets, Tomcat, and Embedded Servers (Beginner-Friendly Note)

This document explains **everything about Spring Web**, including: - What is a **Servlet** - How Servlet works internally - Setting up **Tomcat server** - What is **embedded Tomcat** in Spring Boot - How Spring Web handles requests - Lifecycle diagrams + simple examples



1. What is a Web Application?

A **web application** runs on a server and responds to HTTP requests (GET, POST, PUT, DELETE).

Example:

```
https://example.com/login
```

Your browser sends a request → server processes → sends a response.

To receive that request, Java uses **Servlets**.



2. What is a Servlet?

A **Servlet** is a Java class that handles HTTP requests.

A servlet: - Runs inside a web server (Tomcat / Jetty / Undertow) - Processes HTTP requests - Returns HTTP responses

Example of a Very Simple Servlet:

```
public class HelloServlet extends HttpServlet {  
    @Override  
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)  
        throws IOException {  
        resp.getWriter().write("Hello from servlet!");  
    }  
}
```



3. How Does a Servlet Work? (The Lifecycle)

A servlet lifecycle is:

Loading → Initialization → Service → Destroy

Breakdown

1. **Loading** – Servlet class is loaded
2. **Initialization** – `init()` is called (one-time)
3. **Service** – Each request calls `service()` → `doGet()`, `doPost()`, etc.
4. **Destroy** – `destroy()` when server stops



4. What is a Servlet Container?

A **Servlet Container** is a program that:

- Creates servlet objects
- Manages servlet lifecycle
- Maps URL to servlets
- Handles networking (HTTP requests/responses)

Most popular Servlet container: Apache Tomcat

Others: - Jetty - Undertow - Resin



5. What is Tomcat?

Tomcat is: ✓ A servlet container ✓ A web server ✓ An HTTP server

Tomcat knows how to:

- Accept HTTP requests
- Convert them to ServletRequest
- Call your servlet
- Return a ServletResponse as HTTP



6. How to Set Up Tomcat Server (Spring Core, Old Style)

To run servlets manually, we need:

1. Create a **Dynamic Web Project**
2. Add a servlet class
3. Add `web.xml`
4. Deploy on Tomcat

► `web.xml` Mapping

```
<servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>com.example.HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>Hello</servlet-name>
```

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

When user opens:

```
localhost:8080/hello
```

Tomcat finds the servlet mapped to `/hello` and runs it.



7. What is Spring Web MVC?

Spring Web MVC builds on top of Servlets.

Spring uses a special servlet:

DispatcherServlet (Very Important)

This is the **front controller**.

```
Client → DispatcherServlet → Controller → Service → Repository → DB
```

You do **not** write servlets manually; Spring handles them.



8. Setting Up Spring MVC (Without Spring Boot)

web.xml

```
<servlet>
    <servlet-name>spring</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</
    servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>spring</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

This single servlet handles all requests.



9. What is Embedded Tomcat (Spring Boot)?

Spring Boot changed everything.

You no longer need: ~~X~~ external Tomcat ~~X~~ web.xml ~~X~~ WAR files

Spring Boot includes Tomcat inside the JAR file.

This is called **Embedded Tomcat**.

Run a Spring Boot app → Tomcat starts automatically.

```
@SpringBootApplication  
public class App {  
    public static void main(String[] args) {  
        SpringApplication.run(App.class, args);  
    }  
}
```

Output:

```
Tomcat started on port 8080
```

Your server is ready.



10. Why Embedded Tomcat Is Better

Feature	External Tomcat	Embedded Tomcat
Deployment	WAR file	JAR file (just run)
Start command	Start server manually	<code>java -jar app.jar</code>
Complexity	High	Very Low
Microservices	Not suitable	Perfect

Modern applications = **always embedded Tomcat**.



11. How Spring MVC Works Internally

- 1 User sends HTTP request
- 2 Embedded Tomcat receives it

- 3** Tomcat gives it to DispatcherServlet
- 4** DispatcherServlet finds handler (Controller)
- 5** Controller processes
- 6** DispatcherServlet returns response
- 7** Tomcat sends response back to user



12. Simple Controller Example (Spring Boot)

```
@RestController  
public class HomeController {  
  
    @GetMapping("/hello")  
    public String hello() {  
        return "Hello from Spring Web!";  
    }  
}
```

Run app → open:

```
http://localhost:8080/hello
```

Spring Web handles everything.



13. Summary

Servlet

Java class that handles HTTP requests.

Servlet Container

Runs servlets (Tomcat).

Tomcat

Server + servlet container that understands HTTP.

Embedded Tomcat

Tomcat inside your Spring Boot app.

Spring MVC

Uses DispatchServlet to manage all HTTP requests.

If you want, I can add: - Servlet lifecycle diagram - Flowchart: Browser → Tomcat → Spring - A real project example using Spring MVC (without Boot) - JSP rendering example - WAR deployment walkthrough

Just tell me!