

UNIT 4

Servlets and JSP

(14 Hours) — Advanced Java Programming (CAC354)

Important Notice — Please Read

These notes were prepared from the syllabus and general knowledge of Java. **They may contain errors** (typos, outdated API details, or mistakes in code and explanations), so please take care. Verify important points and test all code yourself, and cross-check with your textbook (Core Java Vol. II / Java: The Complete Reference), the official Java documentation, and your teacher's notes.

4.1 Background

Early web servers could only serve **static** HTML pages. **CGI (Common Gateway Interface)** allowed dynamic content, but started a new process for every request, which was slow and used a lot of memory. **Servlets** solve this: a servlet is a Java class that runs inside a **web container** (such as Apache Tomcat, Jetty) and handles requests using *threads* instead of processes.

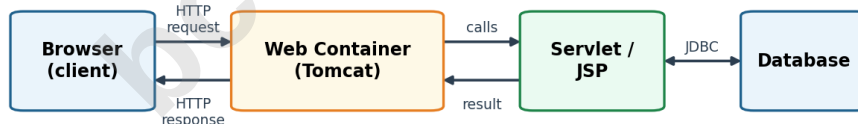


Figure 4.1 — Request flow in a Java web application

CGI	Servlet
New process per request	New thread per request; one servlet instance
Platform dependent (Perl, C)	Platform independent (Java)
Slow, high memory use	Fast and scalable
Less secure	Uses Java security and container services

- **Web server:** handles HTTP requests (e.g. Apache HTTP Server). **Web container / servlet container:** runs servlets and JSPs (Tomcat, Jetty, GlassFish).

- **HTTP** is stateless; common methods are GET (data in URL) and POST (data in body).
- **Deployment descriptor** `WEB-INF/web.xml` (or annotations such as `@WebServlet`) tells the container how to map URLs to servlets.

Note: Package names: Tomcat 9 and older use `javax.servlet.*`, while Tomcat 10 and newer (Jakarta EE 9+) use `jakarta.servlet.*`. The code here uses `javax.servlet`; replace `javax` with `jakarta` if your server requires it.

4.2 The Life Cycle of a Servlet

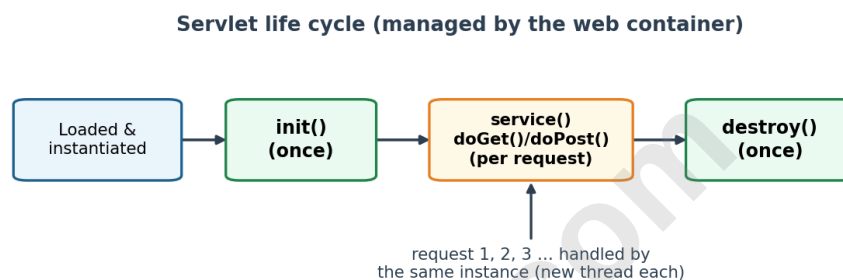


Figure 4.2 — Servlet life cycle

Phase	Method	Called	Purpose
Initialisation	<code>init (ServletConfig)</code>	Once, when first loaded (or at start-up)	Open DB connections, read config
Service	<code>service (req, res)</code>	For every request	Dispatches to <code>doGet()</code> , <code>doPost()</code> etc.
Destruction	<code>destroy ()</code>	Once, before removal	Release resources

- The container loads the class, creates **one instance** and calls `init ()`.
- Each request runs `service ()` in a separate thread of that same instance — so instance variables are **shared** and not thread-safe unless synchronised.
- When the container shuts down or unloads the servlet, `destroy ()` is called and the object is garbage-collected.

4.3 A Simple Servlet

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet("/hello")                                // URL mapping (Servlet 3.0+)
  
```

```

public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<html><body>");
        out.println("<h1>Hello from Servlet!</h1>");
        out.println("</body></html>");
    }
}

```

Equivalent mapping in **web.xml** (older style):

```

<web-app>
  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>HelloServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>

```

Steps to run

- Write the servlet and compile it with the servlet API JAR (`servlet-api.jar`) on the classpath.
- Create the web application structure: `myapp/WEB-INF/classes/` (compiled classes), `WEB-INF/web.xml`, and static pages at the root.
- Copy the folder (or a WAR file) to Tomcat's `webapps` directory and start Tomcat.
- Open `http://localhost:8080/myapp/hello` in a browser.

4.4 The Servlet API

The API is in two packages: `javax.servlet` (generic, protocol-independent) and `javax.servlet.http` (HTTP-specific).

4.4.1 The `javax.servlet` Package

Interface / Class	Description
<code>Servlet</code>	Core interface: <code>init()</code> , <code>service()</code> , <code>destroy()</code> , <code>getServletConfig()</code> , <code>getServletInfo()</code>
<code>ServletConfig</code>	Per-servlet configuration (init parameters, <code>ServletContext</code>)
<code>ServletContext</code>	Information about the web application; shared by all servlets (attributes, <code>getRealPath</code> , logging)

Interface / Class	Description
ServletRequest	Client request: parameters, attributes, input stream
ServletResponse	Response: content type, output writer / stream
RequestDispatcher	Forward to or include another resource (servlet/JSP)
GenericServlet	Abstract protocol-independent servlet class
ServletInputStream / ServletOutputStream	Binary I/O streams
Filter / FilterChain	Intercept requests/responses
ServletException / UnavailableException	Servlet-related exceptions

4.4.2 Reading Servlet Parameters

Data sent from an HTML form or query string is available as request **parameters**:

Method	Returns
getParameter(String name)	Value of a parameter (String) or null
getParameterValues(String name)	String[] — for multiple values (e.g. checkboxes)
getParameterNames()	Enumeration of all parameter names
getParameterMap()	Map of all parameters

HTML form (login.html):

```
<form action="login" method="post">
  Username: <input type="text" name="user"><br>
  Password: <input type="password" name="pass"><br>
  Hobbies: <input type="checkbox" name="hobby" value="music">Music
           <input type="checkbox" name="hobby" value="sports">Sports<br>
  <input type="submit" value="Login">
</form>

@WebServlet("/login")
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
        String user = req.getParameter("user");
        String pass = req.getParameter("pass");
        String[] hobbies = req.getParameterValues("hobby");    // may be null

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        if ("admin".equals(user) && "1234".equals(pass))
            out.println("<h2>Welcome, " + user + "</h2>");
        else
            out.println("<h2>Invalid login</h2>");
    }
}
```

```

    }
}

```

4.4.3 The javax.servlet.http Package

Interface / Class	Description
HttpServlet	Base class for HTTP servlets; override doGet, doPost, doPut, doDelete, doHead
HttpServletRequest	HTTP request: getMethod, getHeader, getCookies, getSession, getRequestURI, getQueryString
HttpServletResponse	HTTP response: setStatus, sendRedirect, sendError, addCookie, setHeader
HttpSession	Session tracking object
Cookie	Small piece of data stored in the browser
HttpSessionListener, HttpSessionAttributeListener	Session events

4.5 Handling HTTP Requests and Responses

	GET	POST
Data location	Appended to URL (query string)	Request body
Visible in URL	Yes	No
Size limit	Limited (URL length)	Practically unlimited
Bookmarkable	Yes	No
Use for	Retrieving data (idempotent)	Submitting forms, changing data
Servlet method	doGet()	doPost()

```

protected void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException {
    String method = req.getMethod(); // "GET"
    String agent = req.getHeader("User-Agent");
    String uri = req.getRequestURI();
    res.setContentType("text/html");
    res.setStatus(HttpServletResponse.SC_OK); // 200
    res.getWriter().println("Method: " + method + " , URI: " + uri);
}
// Redirect and error
res.sendRedirect("home.html"); // client makes a NEW request (302)
res.sendError(HttpServletResponse.SC_NOT_FOUND, "Page missing"); // 404
// Server-side forward (same request, URL unchanged)
RequestDispatcher rd = req.getRequestDispatcher("/result.jsp");
req.setAttribute("msg", "Hello");
rd.forward(req, res);

```

Status code	Meaning
200 OK	Success
302 Found	Redirect
400 Bad Request	Malformed request
401 / 403	Unauthorised / Forbidden
404 Not Found	Resource not found
500 Internal Server Error	Server-side failure

Note: `sendRedirect()` causes the browser to send a new request (URL changes); `forward()` happens inside the server (URL does not change, request attributes are preserved).

4.6 Using Cookies

A **cookie** is a small name–value pair that the server sends to the browser, which then returns it with later requests to the same site. Cookies are used for remembering users, preferences and session IDs. The class is `javax.servlet.http.Cookie`.

```
// create and send a cookie
Cookie c = new Cookie("username", "Ram");
c.setMaxAge(60 * 60 * 24);           // 1 day in seconds (default: until browser closes)
c.setPath("/");
res.addCookie(c);

// read cookies from request
Cookie[] cookies = req.getCookies();           // null if none
if (cookies != null)
    for (Cookie ck : cookies)
        if (ck.getName().equals("username"))
            out.println("Welcome back " + ck.getValue());

// delete a cookie
Cookie d = new Cookie("username", "");
d.setMaxAge(0); res.addCookie(d);
```

- Limits: about 4 KB each; only String values; the user can disable or delete cookies.
- Never store sensitive data (passwords) in cookies. Use `setHttpOnly(true)` and `setSecure(true)` for security.

4.7 Session Tracking

HTTP is stateless, so the server cannot remember a user between requests. **Session tracking** maintains state for a user across requests. Techniques:

Technique	Description	Remarks
Cookies	Store session data/ID in the browser	Simple; may be disabled
URL rewriting	Append session ID/data to URL: page?sid=123 (response.encodeURL())	Works without cookies; ugly URLs
Hidden form fields	<input type=hidden> carrying data	Only for form-based flows
HttpSession	Container-managed session object identified by JSESSIONID	Most widely used

The HttpSession API

Method	Purpose
req.getSession()	Gets the current session, creating one if none exists
req.getSession(false)	Gets existing session or null
setAttribute(name, obj) / getAttribute(name)	Store / retrieve data in the session
removeAttribute(name)	Remove a value
getId()	Unique session ID
setMaxInactiveInterval(sec)	Session timeout
invalidate()	Destroy the session (logout)
isNew(), getCreationTime(), getLastAccessedTime()	Session information

```

@WebServlet("/counter")
public class CounterServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException {
        HttpSession session = req.getSession();
        Integer count = (Integer) session.getAttribute("count");
        count = (count == null) ? 1 : count + 1;
        session.setAttribute("count", count);
        res.setContentType("text/html");
        res.getWriter().println("You have visited " + count + " time(s). Session: " + session.getId());
    }
}
// logout
req.getSession().invalidate();

```

Example 4.1 — Login with session

Create a login servlet that stores the user name in a session and a profile servlet that shows the name only for logged-in users.

Solution

On successful login call `session.setAttribute("user", name)`. In the profile servlet use `getSession(false)`; if null or attribute missing, redirect to the login page.

```
// LoginServlet.doPost
if ("admin".equals(user) && "1234".equals(pass)) {
    req.getSession().setAttribute("user", user);
    res.sendRedirect("profile");
} else res.sendRedirect("login.html");

// ProfileServlet.doGet
HttpSession s = req.getSession(false);
if (s == null || s.getAttribute("user") == null) { res.sendRedirect("login.html"); return; }
res.getWriter().println("Hello " + s.getAttribute("user"));
```

4.8 Introduction to JSP

JavaServer Pages (JSP) is a technology for creating dynamic web pages by embedding Java code in HTML. A JSP is **translated into a servlet** by the container the first time it is requested (*translation* → *compilation* → *loading* → *init* → *service* → *destroy*), so it has the same performance advantages as a servlet. JSP life-cycle methods are `jspInit()`, `_jspService()` and `jspDestroy()`.

JSP Elements

Element	Syntax	Purpose
Comment	<code><!-- comment --></code>	Not sent to client
Directive	<code><%@ page ... %></code>	Page settings, imports, include, taglib
Declaration	<code><%! int x = 0; %></code>	Declares fields/methods (class level)
Scriptlet	<code><% code %></code>	Java statements inside <code>_jspService()</code>
Expression	<code><%= expr %></code>	Evaluates and prints a value
Action	<code><jsp:include page="a.jsp"/></code>	Standard actions (include, forward, useBean...)
EL	<code>\${param.name}</code>	Expression Language for data access

Implicit Objects

Object	Type	Scope
request	<code>HttpServletRequest</code>	request
response	<code>HttpServletResponse</code>	page

Object	Type	Scope
out	JspWriter	page
session	HttpSession	session
application	ServletContext	application
config	ServletConfig	page
pageContext	PageContext	page
page	Object (this)	page
exception	Throwable (error pages only)	page

4.9 Using JSP

```
<%@ page language="java" contentType="text/html; charset=UTF-8" import="java.util.*" %>
<html>
<body>
    <%! int visits = 0; %>
    <% visits++; %>
    <h2>Hello, <%= request.getParameter("name") == null ? "Guest" : request.getParameter("name") %></h2>
    <p>Today is <%= new Date() %></p>
    <p>Page visits (since server start): <%= visits %></p>

    <% for (int i = 1; i <= 5; i++) { %>
        <p>Line <%= i %></p>
    <% } %>

    <jsp:include page="footer.jsp" />
</body>
</html>
```

Form handling in JSP

```
<%-- add.jsp --%>
<%
    int a = Integer.parseInt(request.getParameter("a"));
    int b = Integer.parseInt(request.getParameter("b"));
%>
Sum = <%= a + b %>
```

JSP with JavaBeans and JDBC

```
<jsp:useBean id="s" class="com.example.Student" scope="request" />
<jsp:setProperty name="s" property="*" />      <!-- fills bean from form fields -->
Name: <jsp:getProperty name="s" property="name" />

<%@ page import="java.sql.*" %>
<%
    try (Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/college","root","pwd");
        Statement st = con.createStatement();
        ResultSet rs = st.executeQuery("SELECT id,name FROM student")) {
        while (rs.next()) { %>
            <p><%= rs.getInt(1) %> - <%= rs.getString(2) %></p>
        <% }
    } catch (Exception e) { out.println(e); }
```

%>

Note: Scriptlets mix Java with HTML and are discouraged for large projects. Prefer JSTL tags and EL, and keep business logic in servlets/beans (MVC).

Directives: `page` (settings such as `import`, `errorPage`, `isErrorPage`, `session`), `include` (static include at translation time: `<%@ include file="header.jsp" %>`), and `taglib` (use tag libraries such as JSTL). **Standard actions:** `jsp:include` (dynamic include at request time), `jsp:forward`, `jsp:useBean`, `jsp:setProperty`, `jsp:getProperty`.

4.10 Comparing JSP with Servlet

Servlet	JSP
Java code with HTML inside <code>out.println()</code>	HTML with Java code embedded
Harder to design the presentation	Easy for web designers
Written and compiled by the developer	Translated to a servlet and compiled by the container
Must be mapped in <code>web.xml</code> / annotation	Mapped automatically by file name
Better for control logic, processing	Better for presentation (view)
Changes require recompiling and redeploying	Changes picked up automatically on next request
No implicit objects (obtain manually)	Implicit objects (request, session, out ...)

In the MVC (Model 2) architecture used in real applications: the **servlet is the controller**, **JavaBeans/DAO are the model** and **JSP is the view**. The servlet processes the request, puts results into request attributes and forwards to a JSP that displays them.

```
// Controller servlet
List<Student> list = dao.findAll();           // model
req.setAttribute("students", list);
req.getRequestDispatcher("list.jsp").forward(req, res);

<!-- list.jsp (view) using JSTL + EL -->
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<c:forEach var="s" items="${students}">
    <p>${s.id} - ${s.name}</p>
</c:forEach>
```

4.11 Java Web Frameworks

A **web framework** provides ready-made structure (MVC, request routing, validation, templating, security, database access) so developers write less boilerplate code.

Framework	Description
Spring MVC / Spring Boot	Most popular; dependency injection, REST, auto-configuration, embedded Tomcat
JSF (JavaServer Faces)	Component-based UI framework, part of Jakarta EE
Struts	Older action-based MVC framework (Struts 2 is current)
Hibernate / JPA	ORM (object–relational mapping) used together with web frameworks
Vaadin, Wicket, Play, Grails	Other UI/web frameworks
Jakarta Servlet / JAX-RS	Standards underlying many frameworks

- **Advantages:** faster development, clean separation of concerns, built-in security and validation, large community, easy testing.
- **Disadvantages:** learning curve, extra configuration, framework overhead.

4.12 Short Questions for Revision

- What is a servlet? Explain its life cycle with a diagram.
- Write a servlet to read form data and display it.
- Differentiate doGet() and doPost(); sendRedirect() and forward().
- What is session tracking? Explain different techniques with example.
- What are cookies? Write code to create and read a cookie.
- Explain JSP elements (directive, declaration, scriptlet, expression) with examples.
- Compare JSP and servlet. What are the implicit objects in JSP?

Key Takeaways — Unit 4

- ✓ Servlets are Java classes run by a web container; one instance handles many requests using threads (faster than CGI).
- ✓ Life cycle: `init()` once → `service()/doGet()/doPost()` per request → `destroy()` once.
- ✓ Read form data with `getParameter()` / `getParameterValues()`; respond using `HttpServletResponse` and `PrintWriter`.
- ✓ Cookies store data in the browser; `HttpSession` stores data on the server (identified by `JSESSIONID`); URL rewriting and hidden fields are alternatives.
- ✓ JSP is translated to a servlet; use directives, declarations, scriptlets, expressions, actions and implicit objects.
- ✓ Use servlets as controllers and JSPs as views (MVC); frameworks like Spring MVC, JSF and Struts build on this.

— *End of Unit 4* —

Reminder: these notes may contain errors — verify against your textbook and run the code yourself.