Java Servlets

A Comprehensive Guide with Theory, Code, and Diagrams

Teaching Report for Advanced Learning

October 30, 2025

Contents

1 Introduction to Java Servlets

1.1 What is a Servlet?

A **Servlet** is a Java programming language class that extends the capabilities of servers hosting applications accessed via a request-response programming model. Servlets are:

- Server-side components that handle HTTP requests
- Part of the Java Enterprise Edition (Jakarta EE) specification
- Platform-independent and run inside a servlet container
- The foundation of Java web applications

Key Characteristics

1. **Robust**: Use Java exception handling and memory management
2. **Portable**: Write once, run on any servlet container
3. **Efficient**: Multithreaded architecture for concurrent requests
4. **Secure**: Leverage Java security features

1.2 Evolution: javax vs jakarta

Important Warning

From Tomcat 10 onwards, the package namespace changed from `javax.servlet.*` to `jakarta.servlet.*`. Ensure your code matches your server version.

2 Servlet Architecture

2.1 Request Processing Flow

Figure 1 illustrates how a servlet container processes HTTP requests.

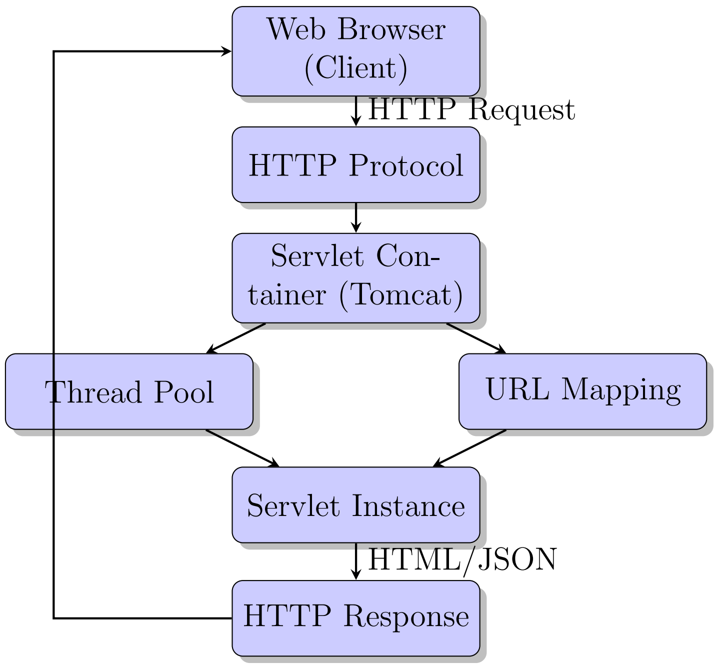

Figure 1: Servlet Request Processing Flow

2.2 Container Responsibilities

The servlet container (e.g., Apache Tomcat, Jetty, WildFly) manages:

- **Lifecycle Management**: Creating, initializing, and destroying servlets
- **Communication Support**: Handling network connections and protocol details
- **Multithreading**: Managing concurrent request processing
- **Security**: Implementing authentication and authorization
- **JSP Support**: Converting JSP pages to servlets

3 Servlet Lifecycle

3.1 The Three Phases

Every servlet goes through three distinct phases managed by the container:

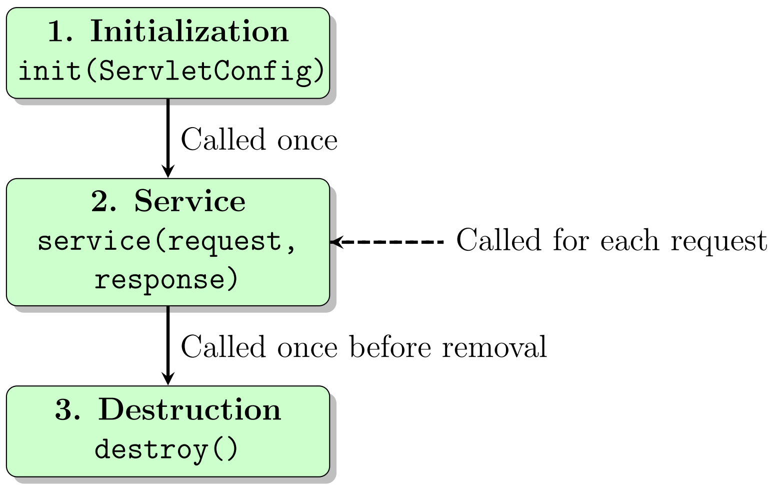

Figure 2: Servlet Lifecycle Phases

3.2 Lifecycle Methods Explained

1. **init(ServletConfig config)**:
   - Called once when servlet is first loaded
   - Used for one-time initialization (database connections, configuration)
   - Must complete before any requests are handled
2. **service(ServletRequest req, ServletResponse res)**:
   - Called for each client request
   - In `HttpServlet`, delegates to `doGet()`, `doPost()`, etc.
   - Must be thread-safe
3. **destroy()**:
   - Called once before servlet is unloaded
   - Used for cleanup (closing connections, releasing resources)
   - Container waits for all service methods to complete

4 Basic Servlet Implementation

4.1 HelloServlet Example

Let's create a complete servlet that demonstrates the basic structure:

```java
package com.example;

import jakarta.servlet.ServletConfig;
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

@WebServlet(name = "HelloServlet", urlPatterns = {"/hello"})
public class HelloServlet extends HttpServlet {

    private String initMessage;

    /**
     * Initialization method called once by the container
     */
    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        initMessage = "HelloServlet initialized at " + new Date();
        System.out.println(initMessage);
    }

    /**
     * Handles HTTP GET requests
     */
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        // Set response content type
        response.setContentType("text/html; charset=UTF-8");

        // Get writer to send response
        PrintWriter out = response.getWriter();

        // Extract query parameters
        String name = request.getParameter("name");
        if (name == null || name.isEmpty()) {
            name = "Guest";
        }

        // Generate HTML response
        out.println("<!DOCTYPE html>");
        out.println("<html><head>");
        out.println("<title>Hello Servlet</title>");
        out.println("<style>");
        out.println("body { font-family: Arial; margin: 40px; }");
        out.println("h1 { color: #2c3e50; }");
        out.println(".info { background: #ecf0f1; padding: 15px; }");
        out.println("</style>");
        out.println("</head><body>");
        out.println("<h1>Hello, " + name + "!</h1>");
        out.println("<div class='info'>");
        out.println("<p><strong>Request URI:</strong> "
                  + request.getRequestURI() + "</p>");
        out.println("<p><strong>Query String:</strong> "
                  + request.getQueryString() + "</p>");
        out.println("<p><strong>Method:</strong> "
                  + request.getMethod() + "</p>");
        out.println("<p><strong>Init Message:</strong> "
67                     + initMessage + "</p>");
68         out.println("</div>");
69         out.println("</body></html>");
70     }
71 
72     /**
73      * Cleanup method called once before servlet is destroyed
74      */
75     @Override
76     public void destroy() {
77         System.out.println("HelloServlet destroyed at " + new Date());
78         // Release resources here
79     }
80 }
```

Listing 1: HelloServlet.java - Basic Servlet Implementation

4.2 Code Walkthrough

Best Practice Tip

Line 14: The `@WebServlet` annotation maps URLs to this servlet. Alternatively, you can configure mappings in `web.xml`.
Line 17: Instance variables like `initMessage` should be immutable or thread-safe since the servlet handles concurrent requests.
Line 34: Always set the content type before writing response data to ensure proper character encoding.

5 Servlet Configuration

5.1 Using Annotations (Modern Approach)

Servlet 3.0+ supports annotations for configuration:

```
@WebServlet(
    name = "UserServlet",
    urlPatterns = {"/user", "/users/*"},
    initParams = {
        @WebInitParam(name = "dbUrl", value = "jdbc:mysql://localhost:3306/mydb"),
        @WebInitParam(name = "maxUsers", value = "100")
    },
    loadOnStartup = 1
)
public class UserServlet extends HttpServlet {

    private String dbUrl;

    @Override
    public void init() throws ServletException {
        // Access init parameters
        dbUrl = getServletConfig().getInitParameter("dbUrl");
    }
}
```

Listing 2: Annotation-Based Configuration

5.2 Using web.xml (Traditional Approach)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
         https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
         version="5.0">

    <servlet>
        <servlet-name>UserServlet</servlet-name>
        <servlet-class>com.example.UserServlet</servlet-class>
        <init-param>
            <param-name>dbUrl</param-name>
            <param-value>jdbc:mysql://localhost:3306/mydb</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>UserServlet</servlet-name>
        <url-pattern>/user</url-pattern>
    </servlet-mapping>

    <!-- Session timeout in minutes -->
    <session-config>
        <session-timeout>30</session-timeout>
    </session-config>

</web-app>
```

Listing 3: web.xml Configuration

5.3 Configuration Comparison

Aspect	**Annotations**	**web.xml**
Ease of use	Simple, colocated with code	Verbose, separate file
Override capability	Cannot override at deployment	Can override at deployment
Centralization	Scattered across classes	Centralized configuration
Java version	Requires Servlet 3.0+	Works with all versions
Preferred for	Development, simple apps	Production, complex deployments

Table 1: Annotations vs web.xml Comparison

6 HTTP Methods and Request Handling

6.1 HTTP Method Overview

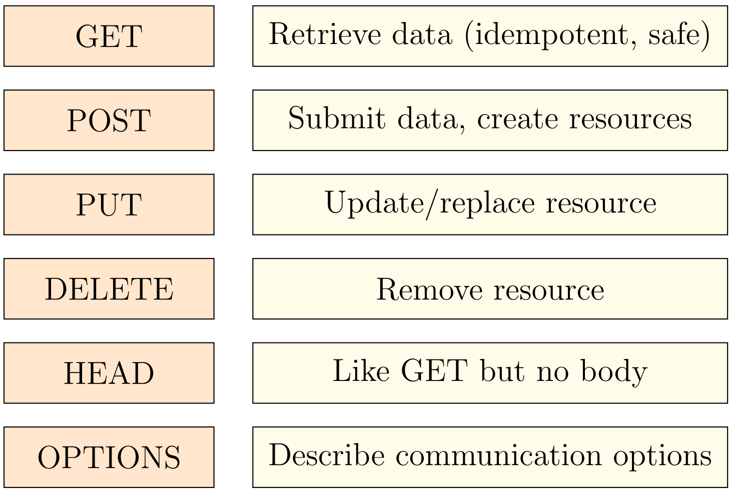
GET
Retrieve data (idempotent, safe)
POST
Submit data, create resources
PUT
Update/replace resource
DELETE
Remove resource
HEAD
Like GET but no body
OPTIONS
Describe communication options

Figure 3: HTTP Methods Supported by Servlets

6.2 Form Processing Example

```java
package com.example;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/login")
public class LoginServlet extends HttpServlet {

    /**
     * Display login form
     */
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<!DOCTYPE html>");
        out.println("<html><head><title>Login</title></head><body>");
        out.println("<h2>Login Form</h2>");
        out.println("<form method='post' action='login'>");
        out.println("Username: <input type='text' name='username'><br>"
    );
```

```java
        out.println("Password: <input type='password' name='password'><br>");
        out.println("<input type='submit' value='Login'>");
        out.println("</form>");
        out.println("</body></html>");
    }

    /**
     * Process login form submission
     */
    @Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {

        // Extract form parameters
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Validate credentials (simplified - use proper authentication!)
        if (isValidUser(username, password)) {
            // Create session and store user info
            HttpSession session = request.getSession();
            session.setAttribute("username", username);
            session.setAttribute("loginTime", System.currentTimeMillis());

            // Redirect to welcome page
            response.sendRedirect("welcome");
        } else {
            // Show error
            response.setContentType("text/html; charset=UTF-8");
            PrintWriter out = response.getWriter();
            out.println("<html><body>");
            out.println("<h3 style='color:red;'>Invalid credentials!</h3>");
            out.println("<a href='login'>Try again</a>");
            out.println("</body></html>");
        }
    }

    private boolean isValidUser(String username, String password) {
        // TODO: Check against database
        return "admin".equals(username) && "pass123".equals(password);
    }
}
```

Listing 4: LoginServlet.java - Form Processing

6.3 RESTful API Example

```java
@WebServlet("/api/products/*")
public class ProductServlet extends HttpServlet {

    // GET: Retrieve products
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        String pathInfo = request.getPathInfo();
        response.setContentType("application/json; charset=UTF-8");
        PrintWriter out = response.getWriter();

        if (pathInfo == null || pathInfo.equals("/")) {
            // List all products
            out.print("{\"products\": [{\"id\":1, \"name\":\"Laptop
   \"}]}");
        } else {
            // Get specific product
            String id = pathInfo.substring(1);
            out.print("{\"id\":" + id + ", \"name\":\"Laptop\"}");
        }
    }

    // POST: Create product
    @Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {

        // Read JSON from request body
        StringBuilder json = new StringBuilder();
        String line;
        try (BufferedReader reader = request.getReader()) {
            while ((line = reader.readLine()) != null) {
                json.append(line);
            }
        }

        // Process and create product (parse JSON, save to DB)

        response.setStatus(HttpServletResponse.SC_CREATED);
        response.setContentType("application/json");
        response.getWriter().print("{\"message\":\"Created\"}");
    }

    // PUT: Update product
    @Override
    protected void doPut(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        String id = request.getPathInfo().substring(1);
        // Update logic here

        response.setContentType("application/json");
        response.getWriter().print("{\"message\":\"Updated\"}");
    }

    // DELETE: Remove product
    @Override
    protected void doDelete(HttpServletRequest request,
                            HttpServletResponse response)
            throws ServletException, IOException {

        String id = request.getPathInfo().substring(1);
        // Delete logic here

        response.setStatus(HttpServletResponse.SC_NO_CONTENT);
    }
}
```

Listing 5: ProductServlet.java - RESTful CRUD Operations

7 Session Management

7.1 Understanding Sessions

Sessions maintain state across multiple HTTP requests from the same client. The container assigns a unique session ID (JSESSIONID) tracked via cookies or URL rewriting.

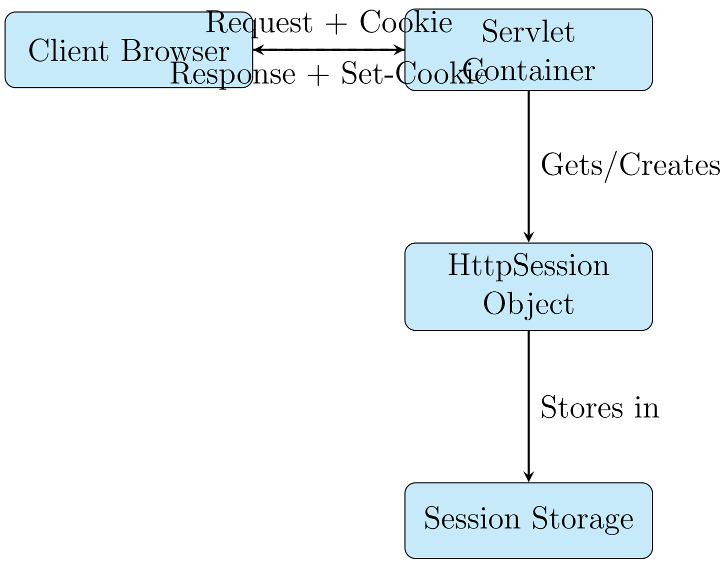

Figure 4: Session Management Flow

7.2 Session Example: Shopping Cart

```java
@WebServlet("/cart")
public class ShoppingCartServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        HttpSession session = request.getSession();

        @SuppressWarnings("unchecked")
        List<String> cart = (List<String>) session.getAttribute("cart")
    ;

```

```java
        if (cart == null) {
            cart = new ArrayList<>();
            session.setAttribute("cart", cart);
        }

        // Display cart
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Your Shopping Cart</h2>");
        out.println("<p>Session ID: " + session.getId() + "</p>");
        out.println("<p>Created: "
                    + new Date(session.getCreationTime()) + "</p>");
        out.println("<ul>");
        for (String item : cart) {
            out.println("<li>" + item + "</li>");
        }
        out.println("</ul>");
        out.println("<form method='post'>");
        out.println("Add Item: <input name='item'>");
        out.println("<input type='submit' value='Add'>");
        out.println("</form>");
        out.println("</body></html>");
    }

    @Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {

        HttpSession session = request.getSession();
        String item = request.getParameter("item");

        if (item != null && !item.isEmpty()) {
            @SuppressWarnings("unchecked")
            List<String> cart = (List<String>) session.getAttribute("cart");
            if (cart == null) {
                cart = new ArrayList<>();
                session.setAttribute("cart", cart);
            }
            cart.add(item);
        }

        response.sendRedirect("cart");
    }
}
```

Listing 6: ShoppingCartServlet.java

7.3 Session Configuration

```xml
<session-config>
    <!-- Timeout in minutes -->
    <session-timeout>30</session-timeout>

    <!-- Cookie configuration -->
    <cookie-config>
        <http-only>true</http-only>
        <secure>true</secure>
        <max-age>3600</max-age>
    </cookie-config>

    <!-- Tracking mode -->
    <tracking-mode>COOKIE</tracking-mode>
</session-config>
```

Listing 7: Session timeout in web.xml

Important Warning

Security Considerations:

- Always use `HttpOnly` flag to prevent XSS attacks
- Use `Secure` flag for HTTPS-only transmission
- Call `session.invalidate()` on logout
- Regenerate session ID after authentication
- Don't store sensitive data directly in sessions

8 Filters

8.1 Filter Architecture

Filters intercept requests and responses, allowing preprocessing and postprocessing.

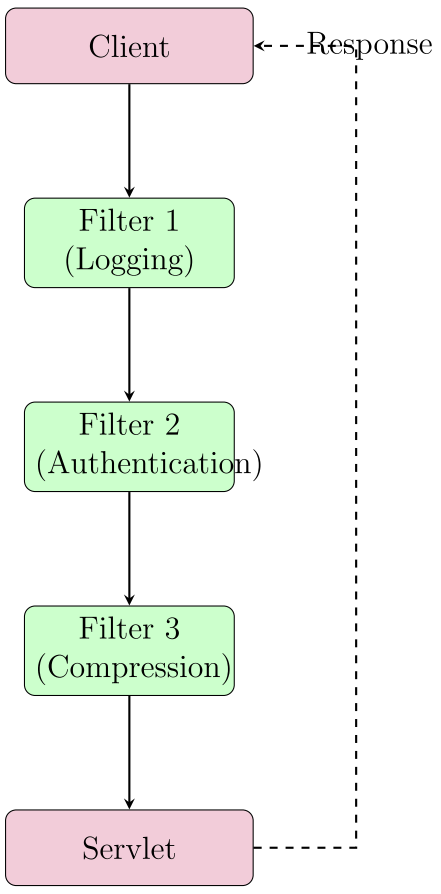

Figure 5: Filter Chain Processing

8.2 Authentication Filter Example

```java
package com.example.filters;

import jakarta.servlet.*;
import jakarta.servlet.annotation.WebFilter;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import java.io.IOException;

@WebFilter(filterName = "AuthFilter", urlPatterns = {"/protected/*"})
public class AuthenticationFilter implements Filter {

    @Override
    public void init(FilterConfig filterConfig) throws ServletException
    {
        System.out.println("AuthenticationFilter initialized");
    }

    @Override
    public void doFilter(ServletRequest request, ServletResponse
   response,
                         FilterChain chain)
            throws IOException, ServletException {

```

```java
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse) response;

        // Check if user is logged in
        HttpSession session = httpRequest.getSession(false);
        boolean loggedIn = (session != null &&
                           session.getAttribute("username") != null);

        String loginURI = httpRequest.getContextPath() + "/login";
        boolean loginRequest = httpRequest.getRequestURI().equals(loginURI);

        if (loggedIn || loginRequest) {
            // Continue to requested resource
            chain.doFilter(request, response);
        } else {
            // Redirect to login page
            httpResponse.sendRedirect(loginURI);
        }
    }

    @Override
    public void destroy() {
        System.out.println("AuthenticationFilter destroyed");
    }
}
```

Listing 8: AuthenticationFilter.java

8.3 Logging Filter Example

```java
@WebFilter("/*")
public class LoggingFilter implements Filter {

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
                        FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;

        // Log request details
        String uri = httpRequest.getRequestURI();
        String method = httpRequest.getMethod();
        String remoteAddr = httpRequest.getRemoteAddr();
        long startTime = System.currentTimeMillis();

        System.out.println(">>> Incoming: " + method + " " + uri
                          + " from " + remoteAddr);

        try {
            // Continue with request processing
            chain.doFilter(request, response);
        } finally {
            // Log response time
            long duration = System.currentTimeMillis() - startTime;
            System.out.println("<<< Completed: " + uri
                                  + " in " + duration + "ms");
        }
    }
}
```

Listing 9: LoggingFilter.java

9 Listeners

9.1 Listener Types

Listeners respond to lifecycle events in web applications:

Listener Interface	**Monitors**
`ServletContextListener`	Application startup/shutdown
`ServletContextAttributeListener`	ServletContext attribute changes
`HttpSessionListener`	Session creation/destruction
`HttpSessionAttributeListener`	Session attribute changes
`ServletRequestListener`	Request creation/destruction
`ServletRequestAttributeListener`	Request attribute changes

Table 2: Servlet Listener Types

9.2 Application Context Listener Example

```
package com.example.listeners;

import jakarta.servlet.ServletContext;
import jakarta.servlet.ServletContextEvent;
import jakarta.servlet.ServletContextListener;
import jakarta.servlet.annotation.WebListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

@WebListener
public class AppContextListener implements ServletContextListener {

    /**
     * Called when application starts
     */
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext ctx = sce.getServletContext();

        // Load configuration
        String dbUrl = ctx.getInitParameter("dbUrl");
        String dbUser = ctx.getInitParameter("dbUser");
        String dbPassword = ctx.getInitParameter("dbPassword");

        // Initialize database connection pool
```

```java
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            Connection conn = DriverManager.getConnection(dbUrl,
                                                          dbUser,
                                                          dbPassword);

            // Store connection in application scope
            ctx.setAttribute("dbConnection", conn);

            System.out.println("Database connection initialized");
            System.out.println("Application started successfully");

        } catch (ClassNotFoundException | SQLException e) {
            e.printStackTrace();
            throw new RuntimeException("Failed to initialize database",
    e);
        }

        // Initialize other application-wide resources
        ctx.setAttribute("appStartTime", System.currentTimeMillis());
        ctx.setAttribute("requestCount", 0);
    }

    /**
     * Called when application shuts down
     */
    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        ServletContext ctx = sce.getServletContext();

        // Close database connection
        Connection conn = (Connection) ctx.getAttribute("dbConnection")
;
        if (conn != null) {
            try {
                conn.close();
                System.out.println("Database connection closed");
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        System.out.println("Application shutdown complete");
    }
}
```

Listing 10: AppContextListener.java - Application Initialization

9.3 Session Listener Example

```java
@WebListener
public class SessionCounterListener implements HttpSessionListener {

    private static int activeSessions = 0;

    @Override
    public void sessionCreated(HttpSessionEvent se) {
        synchronized(SessionCounterListener.class) {
            activeSessions++;
        }
        System.out.println("Session created. Active sessions: "
                           + activeSessions);
        System.out.println("Session ID: " + se.getSession().getId());
    }

    @Override
    public void sessionDestroyed(HttpSessionEvent se) {
        synchronized(SessionCounterListener.class) {
            activeSessions--;
        }
        System.out.println("Session destroyed. Active sessions: "
                           + activeSessions);
    }

    public static int getActiveSessions() {
        return activeSessions;
    }
}
```

Listing 11: SessionCounterListener.java

10 File Upload

10.1 Multipart Form Handling

Servlet 3.0+ provides built-in support for file uploads via the `@MultipartConfig` annotation.

```java
package com.example;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.MultipartConfig;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.Part;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;

@WebServlet("/upload")
@MultipartConfig(
    fileSizeThreshold = 1024 * 1024,      // 1 MB
    maxFileSize = 1024 * 1024 * 10,       // 10 MB
    maxRequestSize = 1024 * 1024 * 50     // 50 MB
)
public class FileUploadServlet extends HttpServlet {

    private static final String UPLOAD_DIR = "uploads";

    /**
     * Display upload form
     */
    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<!DOCTYPE html>");
        out.println("<html><head><title>File Upload</title></head>");
        out.println("<body>");
        out.println("<h2>Upload Files</h2>");
        out.println("<form method='post' enctype='multipart/form-data'>");
        out.println("Select file: <input type='file' name='file'><br><br>");
        out.println("Description: <input type='text' name='description'><br><br>");
        out.println("<input type='submit' value='Upload'>");
        out.println("</form>");
        out.println("</body></html>");
    }

    /**
     * Handle file upload
     */
    @Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {

        // Get the upload directory path
        String applicationPath = request.getServletContext()
                                        .getRealPath("");
        String uploadPath = applicationPath + File.separator + UPLOAD_DIR;

        // Create directory if it doesn't exist
        File uploadDir = new File(uploadPath);
        if (!uploadDir.exists()) {
            uploadDir.mkdir();
        }

        // Process each uploaded file
        for (Part part : request.getParts()) {
            String fileName = getFileName(part);

            if (fileName != null && !fileName.isEmpty()) {
                // Sanitize filename to prevent directory traversal
                fileName = Paths.get(fileName).getFileName().toString();

                // Save file
                Path filePath = Paths.get(uploadPath, fileName);

                try (InputStream input = part.getInputStream()) {
                    Files.copy(input, filePath,
                              StandardCopyOption.REPLACE_EXISTING);
                }

                System.out.println("File uploaded: " + fileName);
            }
        }

        // Get description
        String description = request.getParameter("description");

        // Send response
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h3>Upload Successful!</h3>");
        out.println("<p>Description: " + description + "</p>");
        out.println("<a href='upload'>Upload another file</a>");
        out.println("</body></html>");
    }

    /**
     * Extract filename from content-disposition header
     */
    private String getFileName(Part part) {
        String contentDisposition = part.getHeader("content-disposition
   ");

        if (contentDisposition != null) {
            for (String token : contentDisposition.split(";")) {
                if (token.trim().startsWith("filename")) {
                    return token.substring(token.indexOf('=') + 1)
                               .trim()
                               .replace("\"", "");
                }
            }
        }
        return null;
    }
}
```

Listing 12: FileUploadServlet.java

Important Warning

File Upload Security Best Practices:

- Validate file types (check content, not just extension)
- Limit file size (`maxFileSize` parameter)
- Sanitize filenames to prevent path traversal attacks
- Store files outside web root if possible
- Scan uploaded files for malware
- Use unique filenames to prevent overwrites

11 Concurrency and Threading

11.1 Thread Safety Concerns

Critical Concept: Single Instance Multiple Threads

The servlet container creates **one instance** of each servlet and uses **multiple threads** to handle concurrent requests. This means:

- Instance variables are shared across all threads
- Local variables are thread-safe (stored on stack)
- Synchronization needed for shared mutable state

11.2 Thread Safety Examples

11.2.1 Unsafe Code (DO NOT USE)

```java
@WebServlet("/unsafe")
public class UnsafeServlet extends HttpServlet {

    // DANGER: Shared mutable state!
    private int requestCount = 0;
    private String lastUser;

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        // Race condition: multiple threads can modify simultaneously
        requestCount++;
        lastUser = request.getParameter("user");

        // This will produce incorrect results under concurrent load
        response.getWriter().println("Request #" + requestCount
                                    + " by " + lastUser);
    }
}
```

Listing 13: UnsafeServlet.java - Thread Safety Violation

11.2.2 Safe Code with Atomic Variables

```java
import java.util.concurrent.atomic.AtomicInteger;

@WebServlet("/safe")
public class ThreadSafeServlet extends HttpServlet {

    // Thread-safe counter
    private final AtomicInteger requestCount = new AtomicInteger(0);

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        // Atomically increment
        int count = requestCount.incrementAndGet();

        // Local variable - thread-safe
        String user = request.getParameter("user");

        response.getWriter().println("Request #" + count
                                    + " by " + user);
    }
}
```

Listing 14: ThreadSafeServlet.java - Correct Implementation

11.3 Threading Best Practices

Best Practice Tip

Guidelines for Thread-Safe Servlets:

1. **Prefer local variables** over instance variables
2. **Use immutable objects** when possible
3. **Use** `java.util.concurrent` classes for counters/collections
4. **Synchronize carefully** - minimize critical sections
5. **Avoid storing request data** in instance variables
6. **Never use SingleThreadModel** - it's deprecated and inefficient

12 Security Fundamentals

12.1 Common Security Threats

Threat	Description	Prevention
SQL Injection	Malicious SQL in input	Use PreparedStatement
XSS (Cross-Site Scripting)	Injecting scripts	Escape output, CSP headers
CSRF (Cross-Site Request Forgery)	Unauthorized commands	Use CSRF tokens
Session Hijacking	Stealing session IDs	HTTPS, HttpOnly cookies
Path Traversal	Accessing files outside root	Validate/sanitize paths

Table 3: Common Web Application Security Threats

12.2 Secure Configuration Example

```java
@WebServlet("/secure")
public class SecureServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        // 1. Security Headers
        response.setHeader("X-Content-Type-Options", "nosniff");
        response.setHeader("X-Frame-Options", "DENY");
        response.setHeader("X-XSS-Protection", "1; mode=block");
        response.setHeader("Content-Security-Policy",
                           "default-src 'self'");

        // 2. Secure Cookie
        Cookie cookie = new Cookie("sessionToken", generateToken());
        cookie.setHttpOnly(true);     // Prevent JavaScript access
        cookie.setSecure(true);       // HTTPS only
        cookie.setPath("/");
        cookie.setMaxAge(3600);
        response.addCookie(cookie);

        // 3. Input Validation
        String userId = request.getParameter("id");
        if (!isValidUserId(userId)) {
            response.sendError(HttpServletResponse.SC_BAD_REQUEST,
                             "Invalid user ID");
            return;
        }

        // 4. Output Escaping (prevent XSS)
        String userInput = request.getParameter("comment");
        String safeOutput = escapeHtml(userInput);

        response.setContentType("text/html; charset=UTF-8");
        response.getWriter().println("<p>" + safeOutput + "</p>");
    }

    private boolean isValidUserId(String id) {
        return id != null && id.matches("\\d+");
    }

    private String escapeHtml(String input) {
        if (input == null) return "";
        return input.replace("&", "&amp;")
                    .replace("<", "&lt;")
                    .replace(">", "&gt;")
                    .replace("\"", "&quot;")
                    .replace("'", "&#x27;");
    }

    private String generateToken() {
        // Use secure random token generation
        return java.util.UUID.randomUUID().toString();
    }
}
```

Listing 15: SecureServlet.java - Security Best Practices

12.3 HTTPS Configuration

```xml
<security-constraint>
    <web-resource-collection>
        <web-resource-name>Entire Application</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <user-data-constraint>
        <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
</security-constraint>
```

Listing 16: web.xml - Force HTTPS

13 Deployment and Packaging

13.1 WAR File Structure

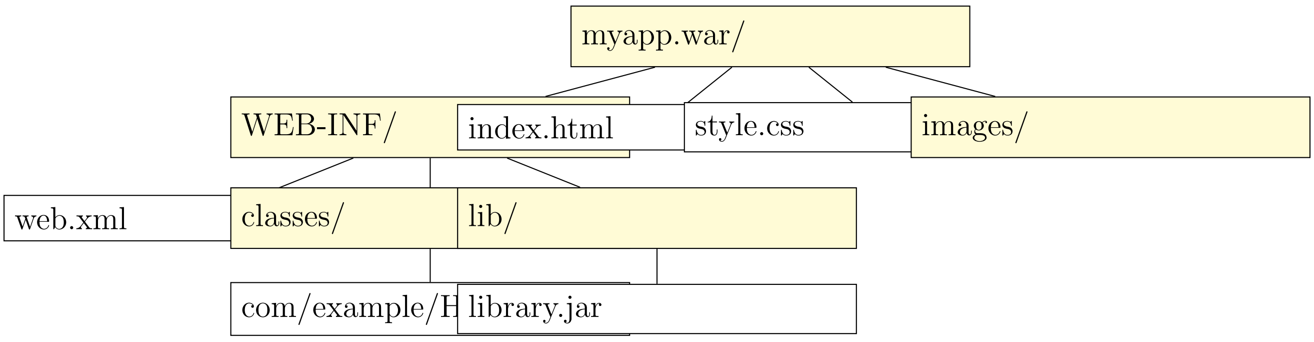

Figure 6: WAR File Directory Structure

13.2 Maven Project Setup

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>servlet-demo</artifactId>
    <version>1.0</version>
    <packaging>war</packaging>

    <properties>
        <maven.compiler.source>11</maven.compiler.source>
        <maven.compiler.target>11</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.
   sourceEncoding>
    </properties>

    <dependencies>
        <!-- Jakarta Servlet API -->
        <dependency>
            <groupId>jakarta.servlet</groupId>
            <artifactId>jakarta.servlet-api</artifactId>
            <version>5.0.0</version>
            <scope>provided</scope>
        </dependency>

        <!-- JSTL (optional) -->
        <dependency>
            <groupId>jakarta.servlet.jsp.jstl</groupId>
            <artifactId>jakarta.servlet.jsp.jstl-api</artifactId>
            <version>2.0.0</version>
        </dependency>
    </dependencies>

    <build>
        <finalName>myapp</finalName>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-war-plugin</artifactId>
                <version>3.3.2</version>
            </plugin>
        </plugins>
    </build>

</project>
```

Listing 17: pom.xml - Maven Configuration

13.3 Deployment Steps

1. **Build the WAR file:**

```
mvn clean package
```

2. **Copy to Tomcat:**

```
cp target/myapp.war $TOMCAT_HOME/webapps/
```

3. **Start Tomcat:**

```
$TOMCAT_HOME/bin/startup.sh     # Linux/Mac
$TOMCAT_HOME/bin/startup.bat    # Windows
```

4. **Access the application:**

```
http://localhost:8080/myapp/
```

14 Debugging and Best Practices

14.1 Logging Configuration

```java
import java.util.logging.Logger;
import java.util.logging.Level;

@WebServlet("/log")
public class LoggingServlet extends HttpServlet {

    private static final Logger LOGGER =
        Logger.getLogger(LoggingServlet.class.getName());

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        LOGGER.info("Processing GET request from "
                   + request.getRemoteAddr());

        try {
            // Business logic
            String result = processRequest(request);
            LOGGER.fine("Request processed successfully");

            response.getWriter().println(result);

        } catch (Exception e) {
            LOGGER.log(Level.SEVERE, "Error processing request", e);
            response.sendError(HttpServletResponse.
    SC_INTERNAL_SERVER_ERROR);
        }
    }

    private String processRequest(HttpServletRequest request) {
        return "Success";
    }
}
```

Listing 18: Using java.util.logging

14.2 Best Practices Checklist

Best Practice Tip

Servlet Development Best Practices:

1. **Design:**
   - Keep servlets thin - delegate business logic to services
   - Use MVC pattern (Model-View-Controller)
   - Follow RESTful principles for APIs
2. **Performance:**
   - Reuse objects when possible
   - Close resources in finally blocks or use try-with-resources
   - Use connection pooling for databases
   - Cache static content
3. **Security:**
   - Validate all input
   - Escape all output
   - Use HTTPS in production
   - Implement proper authentication/authorization
4. **Maintainability:**
   - Use meaningful names
   - Add javadoc comments
   - Follow Java naming conventions
   - Write unit tests

15 Sample Exam Questions

15.1 Short Answer Questions

1. **Q: Explain the servlet lifecycle and name the methods involved.**

   A: The servlet lifecycle consists of three phases:

  - `init(ServletConfig)`: Called once when servlet is loaded
  - `service(ServletRequest, ServletResponse)`: Called for each request
  - `destroy()`: Called once before servlet is unloaded

2. **Q: What is the difference between `doGet()` and `doPost()`?**

   A:

   - `doGet()`: Handles HTTP GET requests - should be idempotent and safe (no state changes), parameters in URL
   - `doPost()`: Handles HTTP POST requests - used for data submission and state changes, parameters in request body

3. **Q: Why should you avoid using instance variables in servlets?**

   A: Servlets are multi-threaded. The container creates one servlet instance and uses multiple threads to handle requests. Instance variables are shared across all threads, leading to race conditions. Use local variables or thread-safe data structures instead.

4. **Q: How does session tracking work in servlets?**

   A: Sessions are tracked using a unique session ID (JSESSIONID) which is:

   - Stored in a cookie (default)
   - Appended to URLs (URL rewriting)
   - The container maps this ID to a HttpSession object

5. **Q: What is a servlet filter and when would you use one?**

   A: A filter intercepts requests/responses before they reach the servlet. Use cases:

   - Authentication/Authorization
   - Logging
   - Data compression
   - Character encoding
   - CORS headers

15.2 Practical Coding Questions

15.2.1 Question 1: Visit Counter

Task: Write a servlet that counts and displays the number of times a user has visited the page using sessions.

```java
@WebServlet("/visitcounter")
public class VisitCounterServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        HttpSession session = request.getSession();

        // Get current visit count from session
        Integer visits = (Integer) session.getAttribute("visitCount");

        if (visits == null) {
            visits = 0;
        }

        // Increment and store
        visits++;
        session.setAttribute("visitCount", visits);

        // Display result
        response.setContentType("text/html; charset=UTF-8");
        PrintWriter out = response.getWriter();
        out.println("<html><body>");
        out.println("<h2>Visit Counter</h2>");
        out.println("<p>You have visited this page " + visits
                    + " time(s).</p>");
        out.println("<p>Session ID: " + session.getId() + "</p>");
        out.println("<a href='visitcounter'>Refresh</a>");
        out.println("</body></html>");
    }
}
```

Listing 19: Solution: VisitCounterServlet.java

15.2.2 Question 2: Authentication Filter

Task: Create a filter that blocks requests to `/admin/*` unless the user has a valid authentication token in the header.

```java
@WebFilter("/admin/*")
public class AdminAuthFilter implements Filter {

    private static final String AUTH_HEADER = "X-Auth-Token";
    private static final String VALID_TOKEN = "secret123";

    @Override
    public void doFilter(ServletRequest request,
                         ServletResponse response,
                         FilterChain chain)
            throws IOException, ServletException {

        HttpServletRequest httpRequest = (HttpServletRequest) request;
        HttpServletResponse httpResponse = (HttpServletResponse)
   response;

        String token = httpRequest.getHeader(AUTH_HEADER);

        if (VALID_TOKEN.equals(token)) {
            // Valid token - continue
            chain.doFilter(request, response);
        } else {
            // Invalid/missing token - block
            httpResponse.sendError(
                HttpServletResponse.SC_UNAUTHORIZED,
                "Missing or invalid authentication token"
            );
        }
    }
}
```

Listing 20: Solution: AdminAuthFilter.java

15.2.3 Question 3: Request Parameter Validator

Task: Write a servlet that validates email and age parameters, returning appropriate error messages.

```java
@WebServlet("/validate")
public class ValidatorServlet extends HttpServlet {

    @Override
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {

        String email = request.getParameter("email");
        String ageStr = request.getParameter("age");

        List<String> errors = new ArrayList<>();

        // Validate email
        if (email == null || !isValidEmail(email)) {
            errors.add("Invalid email address");
        }

        // Validate age
        Integer age = null;
        try {
            age = Integer.parseInt(ageStr);
            if (age < 18 || age > 120) {
                errors.add("Age must be between 18 and 120");
            }
        } catch (NumberFormatException e) {
            errors.add("Age must be a valid number");
        }

        // Send response
        response.setContentType("application/json; charset=UTF-8");
        PrintWriter out = response.getWriter();

        if (errors.isEmpty()) {
            out.print("{\"status\":\"success\", \"message\":\"Valid\"}"
    );
        } else {
            out.print("{\"status\":\"error\", \"errors\":");
            out.print(errors.toString().replace("'", "\""));
            out.print("}");
        }
    }

    private boolean isValidEmail(String email) {
        String regex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}
    $";
```

```java
        return email.matches(regex);
    }
}
```

Listing 21: Solution: ValidatorServlet.java

16 Advanced Topics

16.1 Asynchronous Processing

Servlet 3.0+ supports asynchronous request processing for long-running operations:

```java
@WebServlet(urlPatterns = "/async", asyncSupported = true)
public class AsyncServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        // Start async context
        final AsyncContext asyncContext = request.startAsync();
        asyncContext.setTimeout(30000); // 30 seconds

        // Process in separate thread
        asyncContext.start(() -> {
            try {
                // Simulate long-running operation
                Thread.sleep(5000);

                PrintWriter out = asyncContext.getResponse().getWriter
    ();
                out.write("Async processing complete!");

                // Complete the async context
                asyncContext.complete();

            } catch (Exception e) {
                e.printStackTrace();
            }
        });
    }
}
```

Listing 22: AsyncServlet.java

16.2 ServletContext vs Session vs Request Scope

Scope	**Lifetime**	**Use Case**	**Thread-Safe?**
ServletContext	Application lifetime	Global config, shared resources	Need sync
HttpSession	User session	User-specific data	Single user
Request	Single request	Request-specific data	Yes (thread-local)

Table 4: Comparison of Servlet Scopes

17 Conclusion

This comprehensive guide has covered the essential aspects of Java Servlets, from basic concepts to advanced features. Servlets remain the foundation of Java web development, providing a robust, scalable platform for building web applications.

Key takeaways:

- Servlets provide a powerful, thread-safe framework for handling HTTP requests
- Proper lifecycle management ensures efficient resource utilization
- Filters and listeners extend functionality without modifying servlets
- Security must be considered at every layer
- Modern servlets support asynchronous processing and RESTful APIs

Continue learning by exploring related technologies like JSP, JSF, Spring MVC, and microservices frameworks that build upon servlet technology.