JavaServer Pages (JSP): A Comprehensive Study

Web Development Technologies

October 30, 2025

Contents

1 Introduction to JSP

1.1 What is JSP?

JavaServer Pages (JSP) is a server-side technology used to create dynamic web content. It allows developers to embed Java code directly into HTML pages using special tags. JSP files are compiled into servlets by the JSP container before execution.

1.2 Key Features

- **Separation of Concerns:** JSP separates presentation logic from business logic
- **Platform Independent:** Runs on any platform that supports Java
- **Reusability:** Supports JavaBeans and custom tag libraries
- **Easy Maintenance:** HTML-like syntax makes it easier for web designers
- **Performance:** Compiled into servlets for efficient execution

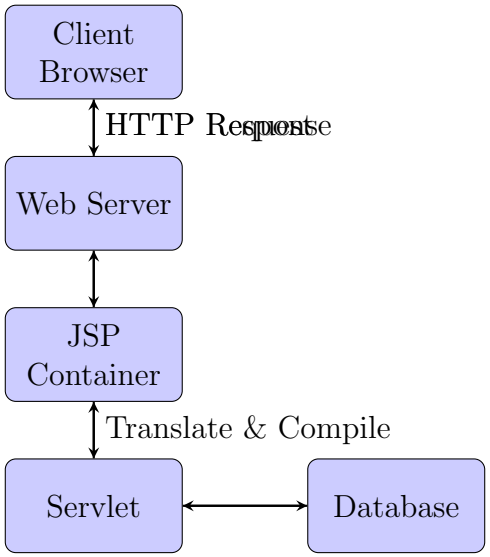
2 JSP Architecture

2.1 JSP Lifecycle

The JSP lifecycle consists of several phases:

1. **Translation:** JSP page is translated to Servlet code
2. **Compilation:** Servlet code is compiled to class file
3. **Class Loading:** Servlet class is loaded into container
4. **Instantiation:** Instance of servlet is created
5. **Initialization:** jspInit() method is called
6. **Request Processing:** _jspService() method handles requests
7. **Destruction:** jspDestroy() method is called before removal

2.2 JSP Architecture Diagram



3 JSP Syntax and Elements

3.1 JSP Scripting Elements

3.1.1 Scriptlet

Used to embed Java code directly in JSP:

```
<%
    String name = "John Doe";
    int age = 25;
    out.println("Name: " + name);
    out.println("Age: " + age);
%>
```

Listing 1: JSP Scriptlet Example

3.1.2 Expression

Used to output values directly:

```
<p>Current Date: <%= new java.util.Date() %></p>
<p>User Name: <%= request.getParameter("username") %></p>
```

Listing 2: JSP Expression Example

3.1.3 Declaration

Used to declare variables and methods:

```
<%!
    private int counter = 0;

    public String getMessage() {
```

```
        return "Welcome to JSP!";
    }
%>
```

Listing 3: JSP Declaration Example

3.2 JSP Directives

3.2.1 Page Directive

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" import="java.util.*,java.sql.*" %>
```

3.2.2 Include Directive

```
<%@ include file="header.jsp" %>
```

3.2.3 Taglib Directive

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

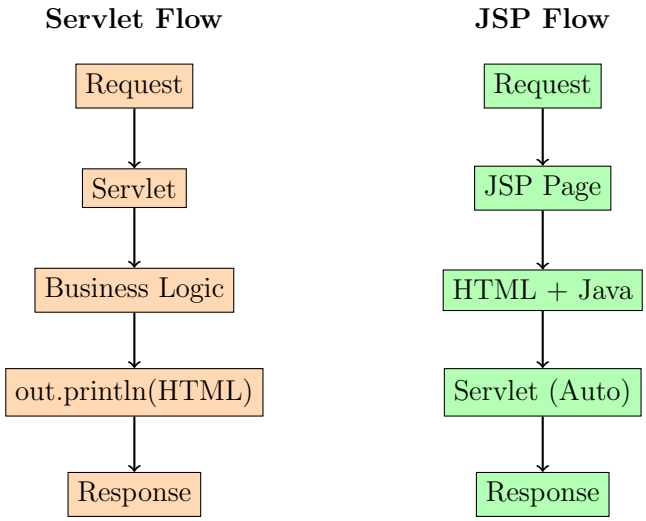
4 Servlet vs JSP: Detailed Comparison

4.1 Conceptual Differences

Aspect	Servlet	JSP
Definition	Java class that handles HTTP requests	HTML page with embedded Java code
Primary Use	Business logic implementation	Presentation layer
Extension	.java / .class	.jsp
Compilation	Manually compiled by developer	Automatically compiled by container
Modification	Requires recompilation	Automatic recompilation on change
HTML Generation	Difficult (using out.println())	Easy (direct HTML writing)
Java Code	Easy to write	Difficult to maintain in large amounts
MVC Role	Controller	View
Performance	Slightly faster (no translation phase)	Slightly slower (first request only)

Table 1: Servlet vs JSP Comparison

4.2 Architecture Comparison Diagram



5 Practical Examples

5.1 Example 1: User Registration Form

5.1.1 Using Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class RegisterServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String password = request.getParameter("password");

        // Business logic
        boolean isValid = validateUser(name, email, password);

        // Generate HTML response
        out.println("<html><head><title>Registration</title></
           head>");
        out.println("<body>");
        out.println("<h2>Registration Result</h2>");

        if(isValid) {
```

```
            out.println("<p style='color:green;'>Registration
               Successful!</p>");
            out.println("<p>Name: " + name + "</p>");
            out.println("<p>Email: " + email + "</p>");
        } else {
            out.println("<p style='color:red;'>Registration
               Failed!</p>");
        }

        out.println("</body></html>");
    }

    private boolean validateUser(String name, String email,
       String pwd) {
        return name != null && email.contains("@") && pwd.length
           () >= 6;
    }
}
```

Listing 4: RegisterServlet.java

5.1.2 Using JSP

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   %>
<!DOCTYPE html>
<html>
<head>
    <title>Registration</title>
</head>
<body>
    <h2>Registration Result</h2>

    <%
        String name = request.getParameter("name");
        String email = request.getParameter("email");
        String password = request.getParameter("password");

        // Business logic
        boolean isValid = name != null &&
                          email.contains("@") &&
                          password.length() >= 6;
    %>

    <% if(isValid) { %>
        <p style="color:green;">Registration Successful!</p>
        <p>Name: <%= name %></p>
        <p>Email: <%= email %></p>
    <% } else { %>
        <p style="color:red;">Registration Failed!</p>
        <p>Please check your input and try again.</p>
```

```
    <% } %>

</body>
</html>
```

Listing 5: register.jsp

5.2 Example 2: Product List Display

5.2.1 Using Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;

public class ProductListServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Get product data
        List<Product> products = getProducts();

        out.println("<html><head><title>Products</title>");
        out.println("<style>");
        out.println("table {border-collapse: collapse; width:
           100%;}");
        out.println("th, td {border: 1px solid black; padding: 8
           px;}");
        out.println("th {background-color: #4CAF50; color: white
           ;}");
        out.println("</style></head><body>");
        out.println("<h2>Product List</h2>");
        out.println("<table>");
        out.println("<tr><th>ID</th><th>Name</th><th>Price</th></
           tr>");

        for(Product p : products) {
            out.println("<tr>");
            out.println("<td>" + p.getId() + "</td>");
            out.println("<td>" + p.getName() + "</td>");
            out.println("<td>$" + p.getPrice() + "</td>");
            out.println("</tr>");
        }

        out.println("</table></body></html>");
    }
```

```
37
38      private List<Product> getProducts() {
39          List<Product> products = new ArrayList<>();
40          products.add(new Product(1, "Laptop", 999.99));
41          products.add(new Product(2, "Mouse", 29.99));
42          products.add(new Product(3, "Keyboard", 79.99));
43          return products;
44      }
45  }
```

Listing 6: ProductListServlet.java

5.2.2 Using JSP

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2           import="java.util.*, com.example.Product" %>
3  <!DOCTYPE html>
4  <html>
5  <head>
6      <title>Products</title>
7      <style>
8          table {
9              border-collapse: collapse;
10             width: 100%;
11         }
12         th, td {
13             border: 1px solid black;
14             padding: 8px;
15             text-align: left;
16         }
17         th {
18             background-color: #4CAF50;
19             color: white;
20         }
21     </style>
22 </head>
23 <body>
24     <h2>Product List</h2>
25
26     <%
27         // Get product data
28         List<Product> products = new ArrayList<>();
29         products.add(new Product(1, "Laptop", 999.99));
30         products.add(new Product(2, "Mouse", 29.99));
31         products.add(new Product(3, "Keyboard", 79.99));
32     %>
33
34     <table>
35         <tr>
36             <th>ID</th>
37             <th>Name</th>
```

```
            <th>Price</th>
        </tr>

        <% for(Product p : products) { %>
        <tr>
            <td><%= p.getId() %></td>
            <td><%= p.getName() %></td>
            <td>$<%= p.getPrice() %></td>
        </tr>
        <% } %>
    </table>

</body>
</html>
```

Listing 7: productList.jsp

6 JSP Implicit Objects

JSP provides nine implicit objects that are automatically available:

Object	Type	Description
request	HttpServletRequest	Represents client request
response	HttpServletResponse	Represents response to client
out	JspWriter	Used to write content to response
session	HttpSession	Represents user session
application	ServletContext	Shared across all users
config	ServletConfig	Configuration information
pageContext	PageContext	Provides access to all scopes
page	Object	Reference to current servlet
exception	Throwable	Available only in error pages

Table 2: JSP Implicit Objects

6.1 Implicit Objects Example

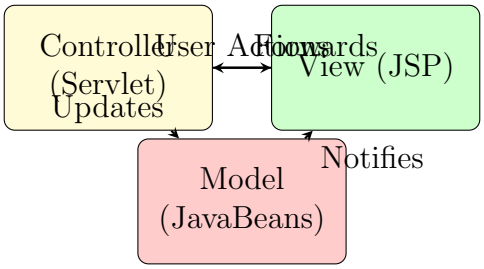
```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   %>
<!DOCTYPE html>
<html>
<head><title>Implicit Objects Demo</title></head>
<body>
    <h2>JSP Implicit Objects</h2>

    <!-- request object -->
    <p>Request Method: <%= request.getMethod() %></p>
    <p>Request URI: <%= request.getRequestURI() %></p>

```

```
    <!-- session object -->
    <% session.setAttribute("username", "JohnDoe"); %>
    <p>Session ID: <%= session.getId() %></p>
    <p>Username: <%= session.getAttribute("username") %></p>

    <!-- application object -->
    <% application.setAttribute("appName", "MyWebApp"); %>
    <p>Application Name: <%= application.getAttribute("appName")
       %></p>

    <!-- out object -->
    <% out.println("<p>Written using out object</p>"); %>

    <!-- config object -->
    <p>Servlet Name: <%= config.getServletName() %></p>

</body>
</html>
```

Listing 8: Implicit Objects Usage

7 MVC Pattern with Servlet and JSP

7.1 MVC Architecture Diagram



7.2 MVC Implementation Example

7.2.1 Model (JavaBean)

```
package com.example.model;

public class User {
    private int id;
    private String username;
    private String email;

    // Constructors
    public User() {}

    public User(int id, String username, String email) {
        this.id = id;
        this.username = username;
```

```
        this.email = email;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getUsername() { return username; }
    public void setUsername(String username) {
        this.username = username;
    }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

Listing 9: User.java (Model)

7.2.2 Controller (Servlet)

```
package com.example.controller;

import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
import com.example.model.User;

public class UserController extends HttpServlet {

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

        // Business logic
        User user = new User(1, "johndoe", "john@example.com");

        // Set model in request scope
        request.setAttribute("user", user);

        // Forward to view (JSP)
        RequestDispatcher dispatcher =
            request.getRequestDispatcher("userView.jsp");
        dispatcher.forward(request, response);
    }

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

        // Get form data
        String username = request.getParameter("username");
```

```
32          String email = request.getParameter("email");
33
34          // Create model
35          User user = new User();
36          user.setUsername(username);
37          user.setEmail(email);
38
39          // Save to database (example)
40          // userDAO.save(user);
41
42          // Redirect or forward
43          request.setAttribute("message", "User saved successfully!
               ");
44          request.setAttribute("user", user);
45
46          RequestDispatcher dispatcher =
47              request.getRequestDispatcher("userView.jsp");
48          dispatcher.forward(request, response);
49      }
50  }
```

Listing 10: UserController.java (Controller)

7.2.3 View (JSP)

```
1   <%@ page language="java" contentType="text/html; charset=UTF-8"
       %>
2   <!DOCTYPE html>
3   <html>
4   <head>
5       <title>User Information</title>
6       <style>
7           .container {
8               max-width: 600px;
9               margin: 50px auto;
10              padding: 20px;
11              border: 1px solid #ddd;
12              border-radius: 8px;
13          }
14          .success { color: green; font-weight: bold; }
15          .info { margin: 10px 0; padding: 10px; background: #
                f0f0f0; }
16      </style>
17  </head>
18  <body>
19      <div class="container">
20          <h2>User Details</h2>
21
22          <% String message = (String) request.getAttribute("
               message"); %>
23          <% if(message != null) { %>
```

```
24              <p class="success"><%= message %></p>
25          <% } %>
26
27          <jsp:useBean id="user" class="com.example.model.User"
28                       scope="request" />
29
30          <div class="info">
31              <p><strong>User ID:</strong>
32                 <jsp:getProperty name="user" property="id" /></p>
33              <p><strong>Username:</strong>
34                 <jsp:getProperty name="user" property="username"
                      /></p>
35              <p><strong>Email:</strong>
36                 <jsp:getProperty name="user" property="email" /></
                      p>
37          </div>
38
39          <h3>Update User</h3>
40          <form action="UserController" method="post">
41              <label>Username:</label>
42              <input type="text" name="username"
43                     value="<jsp:getProperty name='user' property='
                           username' />" />
44              <br/><br/>
45              <label>Email:</label>
46              <input type="email" name="email"
47                     value="<jsp:getProperty name='user' property='
                           email' />" />
48              <br/><br/>
49              <input type="submit" value="Update" />
50          </form>
51      </div>
52  </body>
53  </html>
```

Listing 11: userView.jsp (View)

8 Advantages and Disadvantages

8.1 Servlet Advantages

- Better for implementing complex business logic
- More control over request/response processing
- Easier debugging and testing
- Better performance (no translation overhead)
- Type-safe development

8.2 Servlet Disadvantages

- HTML generation is cumbersome
- Difficult for web designers to work with
- Any HTML change requires recompilation
- Not ideal for presentation layer

8.3 JSP Advantages

- Easy to create and maintain HTML content
- Separation of presentation from business logic
- Automatic compilation and deployment
- Web designers can work with familiar HTML
- Supports custom tags and JSTL

8.4 JSP Disadvantages

- Can become messy with too much Java code
- Difficult to test compared to Servlets
- First request is slower (compilation)
- Debugging can be challenging

9 Best Practices

9.1 When to Use Servlets

- Implementing application controllers
- Processing form submissions
- Performing authentication and authorization
- Handling business logic
- Database operations
- Session management

9.2 When to Use JSP

- Creating dynamic web pages
- Displaying data to users
- Generating reports
- Creating templates
- Presentation layer in MVC

9.3 General Best Practices

- Use MVC pattern: Servlet as Controller, JSP as View
- Minimize Java code in JSP pages
- Use JSTL and custom tags instead of scriptlets
- Keep business logic in JavaBeans
- Use request forwarding appropriately
- Handle exceptions properly
- Use session management wisely

10 Conclusion

Both Servlets and JSP are essential technologies in Java web development. Servlets excel at handling business logic and controller functionality, while JSP is ideal for creating dynamic, content-rich web pages. Understanding the strengths and appropriate use cases of each technology enables developers to build robust, maintainable web applications.

The key to successful web application development is using these technologies together in a complementary manner, typically following the MVC pattern where Servlets act as controllers and JSP pages serve as views, with JavaBeans representing the model layer.

11 References

1. Oracle Corporation. (2024). JavaServer Pages Technology.
2. Head First Servlets and JSP, 2nd Edition - O'Reilly Media
3. Professional Java for Web Applications - Wrox Press
4. Java EE 8 Documentation - Oracle