JavaServer Pages (JSP)

Why JSP?

You've mastered Servlets — powerful, yes, but let's be honest... writing HTML using hundreds of out.println() statements is like trying to paint with a toothpick. Messy, slow, and frustrating.

Think of JSP as Servlets made human-friendly, letting you write HTML and embed Java code directly where needed. No more wrestling with endless print statements — just clean, readable pages.

- Cleaner separation of HTML and Java logic.
- Easier maintenance than writing Servlets for UI.
- Built-in features like tag libraries and Expression Language (EL).
- Ideal for MVC architecture as the view component.

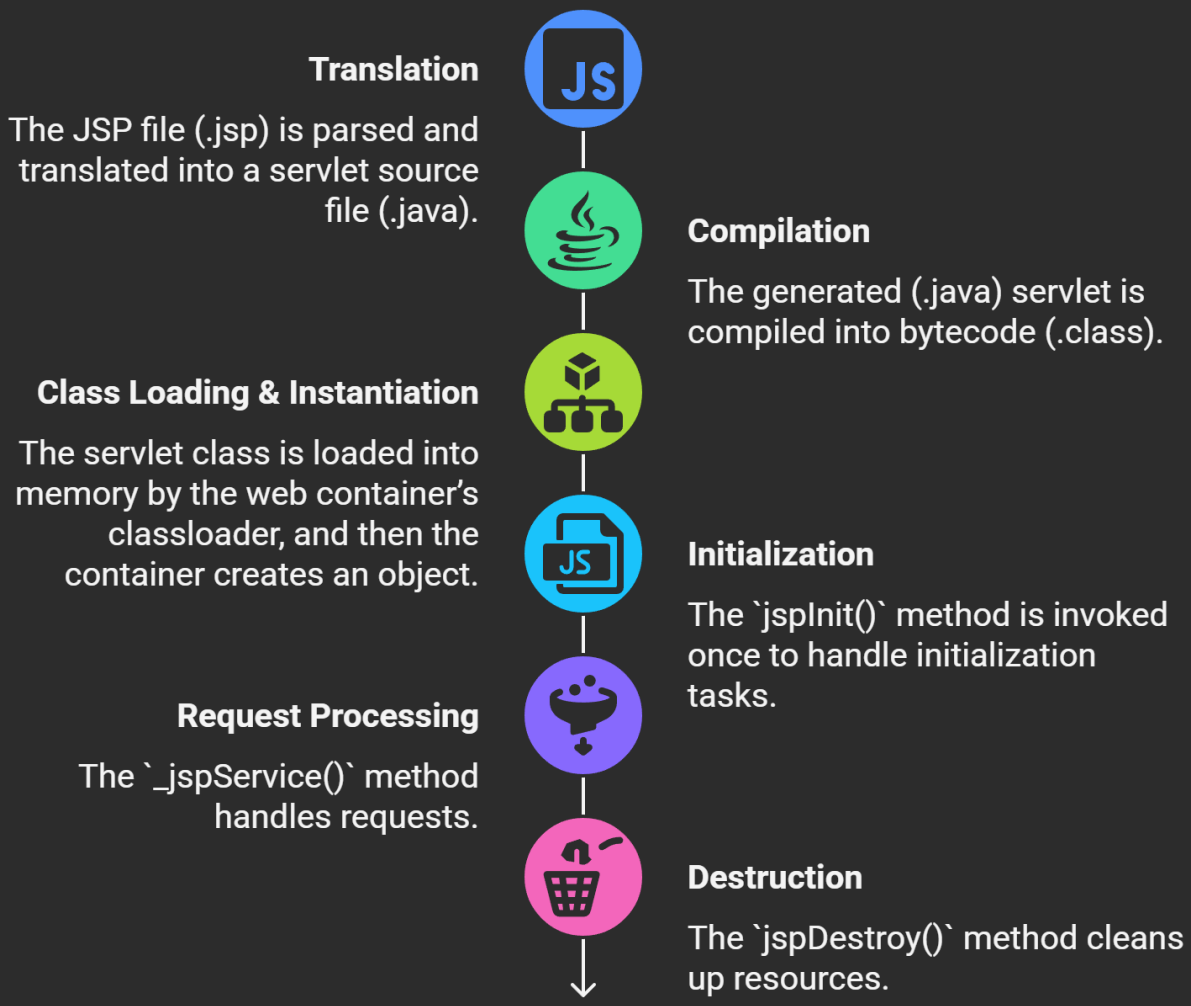
Introduction:

JSP technology is used to create web applications just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

JavaServer Pages (JSP) is a server-side technology that enables the creation of dynamic, platform-independent web applications. JSP

JSP Lifecycle: From Translation to Destruction



files are compiled into Servlets by the server and executed to generate dynamic content.

How JSP Works:

1. Client sends an HTTP request to the server for a JSP page.
2. The JSP engine translates the JSP into a Servlet.
3. The Servlet is compiled into bytecode and executed.
4. The output (HTML, JSON, etc.) is sent back to the client.

Ways to Add JSP:

JSP can be integrated via:

- **Direct JSP Pages**: Standalone .jsp files used independently to generate dynamic content.
- **JSP with Servlets**: Servlets handle the business logic, while JSP is used to render the user interface (presentation logic), promoting separation of concerns.
- **Custom Tag Libraries**: Define reusable, application-specific tags to simplify and modularize JSP code. Commonly used with JSTL or custom-defined tag libraries.

JSP Scripting Elements:

1. Declaration (<%! %>): Defines variables and methods at class level.
2. Scriptlet (<% %>): Java code embedded within HTML.
3. Expression (<%= %>): Outputs value directly to the response.

JSP Declaration

JSP declarations are used to **declare variables and methods** that are **used in the JSP page**. These are defined using <%! ... %> tags and are placed **outside the service method**.

Syntax:

```
<%! int count = 0; %>

<%! public int square(int x) { return x * x; } %>
```

- Code inside <%! %> becomes part of the **generated servlet class**, outside the _jspService() method.
- Can be used across the page.

JSP Scriptlet

Scriptlets allow writing **Java code inside JSP pages**, using <% ... %>.

Syntax:

```
<%
 String name = "Meghana";
 out.println("Welcome, " + name);
%>
```

- Code is inserted inside the _jspService() method of the generated servlet.
- Overused scriptlets can make code messy — modern practice is to minimize their use.

JSP Expressions

This scripting element is often wont to evaluate the only java expression and display that expression resultant value onto the client browser. The expression element contains a Java expression that returns a worth. This value is then written to the HTML page. The Expression tag can contain any expression that's valid and consistent with the Java Language Specification. This includes variables, method calls then return values, or any object that contains a toString() method. It evaluates the given expression and displays generated results onto browser windows. Anything that returns a result is called an expression.

Syntax:

<%=Java Expression%>

Example JSP Code:

```
<html>
<body>
  <h1>Hello from JSP</h1>
  <p>Today's date: <%= new java.util.Date() %></p>
</body>
</html>
```

JSP Directives

JSP directives **provide global information** about the JSP page. They control how the page is translated into a servlet.

In the web applications, JSP Directives can be used to define present JSP page characteristics, to include the target resource content into the present JSP page, and to make available user-defined tag library into the present JSP page. All the JSP directives are going to be resolved at the time of translating the JSP page to the servlet. The majority of JSP Directives will not give a direct effect to response generation.

Syntax:

<%@ directive attribute="value" %>

Common Directives:

1. **Page** – Defines page-level settings

<%@ page language="java" contentType="text/html" %>

2. **Include** – Includes another file during JSP translation

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

3. **Taglib** – Used to include custom tag libraries

<%@ taglib uri="..." prefix="..." %>

---

1. Page Directive

The **page directive** defines global settings for a JSP page, such as importing Java classes, setting the content type, configuring error handling, and more.

Syntax:

<%@ page attribute="value" %>

Common Attributes of the Page Directive

Attribute	Description
import	Imports Java classes (e.g., java.util.List).
contentType	Sets the content type (default: text/html).
isErrorPage	Marks the JSP as an error page (true or false).
errorPage	Specifies the error page to forward to when an exception occurs.
language	Defines the scripting language (default: java).
session	Enables or disables session tracking (true or false).

Examples:

a) Importing Java Classes

```
<%@ page import="java.util.Date, java.util.List" %>
Current Date: <%= new Date() %>
```

b) Setting Content Type

```
<%@ page contentType="text/xml" %>
Hello, XML!
```

c) Specifying an Error Page

```
<%@ page errorPage="error.jsp" %>
<%
   int result = 10 / 0; // This will cause an exception
%>
```

d) Creating an Error Page

```
<%@ page isErrorPage="true" %>
Error Occurred: <%= exception.getMessage() %>
```

---

2. Include Directive

The **include directive** inserts the content of another JSP file or static resource into the current JSP **at translation time**. Commonly used for headers, footers, or navigation menus.

Syntax:

```
<%@ include file="relative_path_to_file" %>
```

Example – Including a Header File:

header.jsp

```
<h1>My Website Header</h1>
```

index.jsp

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

Welcome to My Website!
```

---

3. Taglib Directive

The **taglib directive** imports custom tag libraries into a JSP page. Custom tags are reusable components that simplify complex logic.

Syntax:

```
<%@ taglib prefix="prefix" uri="URI_to_tag_library" %>
```

Example – Using JSTL (Java Standard Tag Library):

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

${item}
```

JSTL & Expression Language (EL):

JavaServer Pages Standard Tag Library (JSTL) and Expression Language (EL) make JSP cleaner by avoiding Java code in HTML.

Example:

```
<c:forEach var="item" items="${feedbackList}">
  <tr>
    <td>${item.name}</td>
    <td>${item.feedback}</td>
  </tr>
</c:forEach>
```

JSP Implicit Objects:

JSP provides several **implicit objects** that are automatically available in a JSP page, without needing to declare or create them. These objects simplify request processing, response generation, and application data handling.

1. out (JspWriter)

Used to send content to the response stream. Typically used for displaying output in the browser.

2. request (HttpServletRequest)

Used to retrieve client request data such as form parameters, headers, and request attributes.

3. response (HttpServletResponse)

Used to control the HTTP response, such as sending redirects or setting response headers.

4. config (ServletConfig)

Provides access to initialization parameters specific to the current JSP page.

5. application (ServletContext)

Represents the web application context. Shared across all JSPs and servlets in the application, used for global configuration and attributes.

6. session (HttpSession)

Stores and retrieves user-specific data across multiple requests within the same session.

7. pageContext (PageContext)

Provides a single API to manage attributes across different scopes: **page**, **request**, **session**, and **application**.

8. page (Object)

Refers to the current JSP page instance. Rarely used directly.

9. exception (Throwable)

Available only on error pages (isErrorPage="true"). Used to access and display exception details.

Advantages of JSP:

✅ Extension of Servlet (full access to Servlet features). And simplifies page creation.

✅ Cleaner separation: UI (HTML) & Logic (Java). Supports reusable components.

✅ No need to recompile after changes

✅ Use of EL, JSTL, and Custom Tags reduces boilerplate

✅ Works seamlessly with JavaBeans.

✅ Portable across different servers.

Best Practices:

- Use JSP for presentation, not business logic.
- Prefer JSTL and EL over scriptlets.
- Follow MVC architecture.
- Keep code modular and reusable.

Servlet vs. JSP Comparison

Characteristic	Servlet	JSP
Speed	Faster	Slower
Protocol Handling	All protocols	HTTP only
Service Method Override	Allowed	Not allowed
Session Management	Not enabled by default	Enabled by default
Logic Separation	Combined	Separated
Modification Update	Recompilation needed	Quick refresh
Implicit Objects	No built-in objects	Several built-in objects
JavaScript Support	No direct way	Supports embedding
Package Import	Top only	Anywhere

To deepen your understanding of Servlets, JSP, and JSF, check out this insightful blog post: Servelts, JSP, and JSF