JAVA BACKEND DEVELOPMENT

FROM SERVLETS TO JSF

A Journey Through the Evolution of Java Web Technologies

Introduction

Welcome to this brief yet interesting primer on the development of Java backend development—a path from the basic components to more complex component-based systems. This tutorial investigates the evolution from Servlets to JavaServer Pages (JSP) and finally to JavaServer Faces (JSF, providing a thorough knowledge of how each technology has helped to create contemporary Java web applications.

The heart of Java web development, Servlets, start with Java programs running on a web server or application server to manage client requests and create dynamic answers. Managed by the servlet container—like Apache Tomcat—servlets follow a welldefined four-step cycle guaranteeing flawless request handling and response delivery.

We next look at JSP, a technology that extends servlets by enabling programmers to incorporate Java code directly within HTML using unique JSP tags. Prior to execution, JSP pages are turned into servlets that combine the power of Java with the simplicity of HTML. With features like Expression Language (EL), JSTL, and custom tags, JSP increases efficiency by clearly separating presentation from logic.

Finally, we come on JSF (JavaServer Faces), a strong, component-based web framework that streamlines the creation of interactive and reusable user interfaces. JSF is a preferred option for creating scalable enterprise-level applications because of its structured approach to web development using a sixphase lifecycle that handles the whole requestresponse process.

This handbook will help you learn about the lifecycles, application methods, and individual capabilities of these technologies—providing a strong basis for expert Java backend development.

Origins of Java Backend Development

Inspired by languages like C and C++, Java revolutionized the programming world when Sun Microsystems introduced it in 1995 with its platform-independent design. Originally, backend systems depended mostly on CGI scripts created in Perl or C, which were susceptible to performance bottlenecks caused by substantial process burden. With its object-oriented design and excellent performance, Java developed as a viable alternative. Java became a top backend language when Java Enterprise Edition (Java EE), presently known as Jakarta EE, debuted in 1999. This laid the groundwork for serverside developments including Servlets, JSP, and JSF.

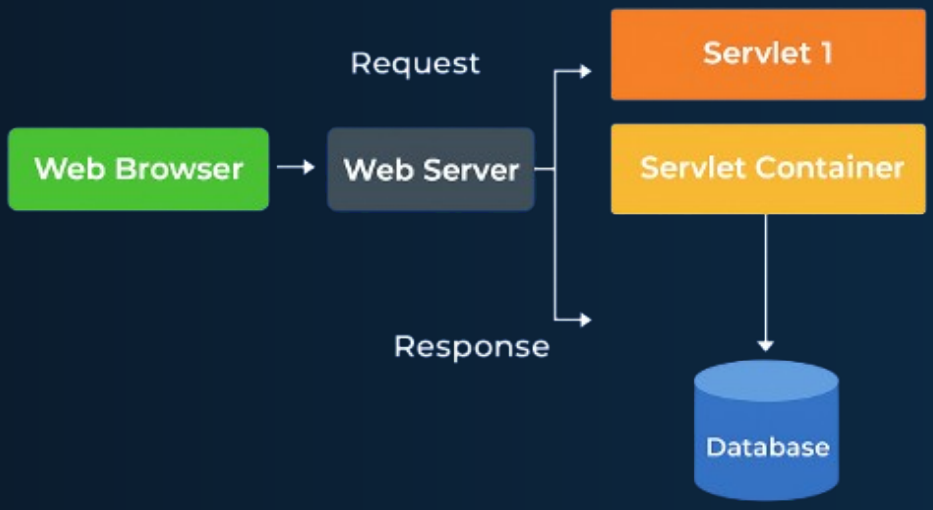
Servlets: The Foundation of Web Development

Introduced in 1997, servlets transformed dynamic web content creation by providing a more efficient alternative to CGI. Operating within the Java Virtual Machine (JVM) and utilizing multithreading, servlets got rid of the performance problems related to process-based systems. Expertly handling HTTP requests and responses, they swiftly evolved into the basis of Java web applications, paving the way for scalable, high-performance web solutions.

Servlets are widely used for tasks such as:

1. Processing or storing data submitted via HTML forms
2. Generating dynamic content, such as displaying results from a database query
3. Managing state information across sessions using cookies or HTTP sessions

Servlet Architecture



Servlet Lifecycle

The lifecycle of a Servlet includes four phases:

1. **Loading and Instantiation**: Firstly the `object` is created and then the servlet class is loaded and an instance is created.
2. **Initialization**: The `init()` method sets up resources.
3. **Request Handling**: The `service()` method processes requests, delegating to `doGet()` or `doPost()`.
4. **Destruction**: The `destroy()` method releases resources.

Ways to Implement Servlets

Servlets can be implemented in three ways:

- **Implementing `Servlet` Interface**: Provides full control over all lifecycle methods (init(), service(), destroy(), etc.). This is the most basic approach but requires more boiler-plate code.
- **Extending `GenericServlet`**: A protocol-independent abstract class that simplifies servlet creation by handling common functionality. Developers only need to override the service() method.
- **Extending `HttpServlet`**: The most commonly used method, designed specifically for handling HTTP requests. It provides convenient methods like doGet(), doPost(), etc., making it ideal for web-based applications.

Advantages of Servlets

1. **Platform Independent:** Runs on any JVM-supported platform.
2. **Scalable:** Efficiently handles multiple concurrent requests via servlet containers.
3. **Easy Integration:** Works seamlessly with JSP, JDBC, EJB, and other Java technologies.
4. **Robust & Secure:** Strong typing, exception handling, and memory management.
5. **Server-Side Processing:** Ideal for form handling, session management, and DB access.

Simplifying Web Development Through JSP

While Servlets offered robust backend capabilities, building user interfaces with them was often cumbersome due to embedded HTML within Java code. To address this, JavaServer Pages (JSP) were introduced in 1999, providing a more intuitive and design-friendly approach to web development. By blending HTML-like syntax with Java code, JSP made it easier to create dynamic web pages and effectively separated presentation from business logic. This shift not only improved developer productivity but also enabled better collaboration between designers and developers.

Introduction to JavaServer Pages (JSP)

JSP pages, with a `.jsp` extension, combine HTML, XML, or custom tags with Java code via scriptlets (`<% %>`), expressions (`<%= %>`), and directives (`<%@ %>`). JSPs compile into Servlets, ensuring compatibility with Java EE.

Advantages of JSP

- **Simplified UI Development**: HTML-like syntax suits front-end developers.
- **Reusability**: Custom tag libraries promote reusable components.
- **Integration**: Works seamlessly with Java EE components.
- **Maintainability**: Separates presentation and logic for cleaner code.

JSP Lifecycle

The JSP lifecycle includes:

1. **Translation**: The JSP file `(.jsp)` is parsed and translated into a servlet source file `(.java)`.
2. **Compilation**: The generated `(.java)` servlet is compiled into bytecode `(.class)`.
3. **Class Loading & Instantiation**: The servlet class is loaded into memory by the web container's classloader. And then the container creates an object.
4. **Initialization**: The `jspInit()` method is invoked once to handle initialization tasks.
5. **Request Processing**: `_jspService()` handles requests.
6. **Destruction**: `jspDestroy()` cleans up resources.

Ways to Add JSP

JSP can be integrated via:

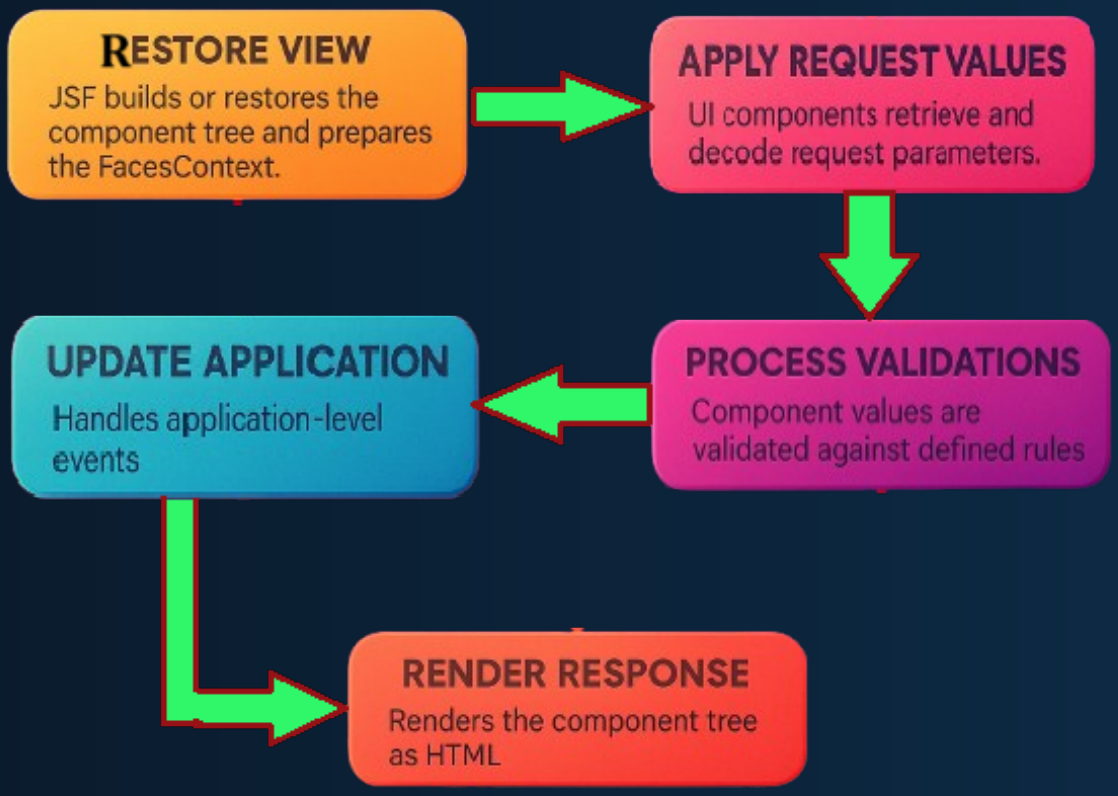
- **Direct JSP Pages**: Standalone `.jsp` files used independently to generate dynamic content.
- **JSP with Servlets**: Servlets handle the Business logic, while JSP is used to render the user interface (Presentation logic), promoting separation of concerns.
- **Custom Tag Libraries**: Define reusable,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

Introduction to JavaServer Faces (JSF)

JavaServer Faces (JSF) was Launched in 2004, for creation of dynamic and engaging user interfaces for Java web applications by using a componentbased MVC web framework. Based atop Servlets and JSP, JSF helps programmers to produce reusable UI components while also enabling eventdriven programming and a welldefined application lifetime.

Unlike traditional web development models, JSF promotes event-driven programming, allowing developers to handle user interactions with ease.

JavaServer Faces Lifecycle



The JSF lifecycle includes the following phases:

1. **Restore View**: Retrieves or creates the component tree.
2. **Apply Request Values**: Updates components with request data.
3. **Process Validations**: Validates user input.
4. **Update Model Values**: Updates backing beans.
5. **Invoke Application**: Executes logic and navigation.
6. **Render Response**: Generates output, often with JSP or Facelets.

Why Choose JavaServer Faces (JSF) for Web Development

JavaServer Faces (JSF) offers a robust platform for building dynamic and interactive web applications using Java. Its component-based architecture, built-in UI components, and event-driven model make it ideal for both simple websites and complex enterprise solutions.

- By understanding its core concepts and following best practices, you can start building feature-rich, maintainable, and scalable web applications with ease.
- JSF is built on top of the Servlet API. And it offers server-side validation, data conversion, defining page navigation, provides extensibility, support for internationalization, accessibility, etc.

Ways to Implement JSF

JSF applications can use:

- **JSP with JSF Tags**: Early JSF used JSP-based tags.
- **Facelets**: Preferred since JSF 2.0, using XHTML for templating.
- **Custom Components**: Reusable UI components for specific needs.

JSF Component Library Overview

JavaServer Faces (JSF) offers a rich set of UI components that form the foundation of its **component-based architecture**, supporting rendering, conversion, validation, and event handling. These components simplify the development of dynamic and interactive web interfaces by mapping directly to standard HTML elements.

Core JSF Components

Form & Input Handling

- `h:form` – Defines a form container for input components
- `h:inputText` – Text field for user input
- `h:inputSecret` – Password input field (masked characters)
- `h:commandButton` – Submits the form to trigger an action

Output & Display

- `h:outputText` – Displays text content
- `h:graphicImage` – Renders images
- `h:panelGrid` – Creates structured, grid-like layouts

Data & Selection Components

- `h:dataTable` – Displays and manages dynamic tabular data
- `h:selectOneMenu` – Dropdown for single selection
- `h:selectManyCheckbox` – Checkboxes for multiple selections

These components render as standard **HTML elements** (`<input>`, `<form>`, `<table>`, `<img>`, etc.), ensuring a **seamless and consistent integration** with the browser's rendering engine. With built-in support for event listeners, data conversion, and validation, JSF components enable developers to build modern, scalable, and interactive web applications efficiently.

Comparison of Servlets, JSP, and JSF

Feature	Servlet	JSP	JSF
Year	1997	1999	2004
Type	Java class (code-centric)	HTML-based with Java code	Component-based MVC framework
Primary Use	Processing requests and generating dynamic content	Simplifying UI creation with embedded Java	Building rich, reusable, event-driven user interfaces
Syntax Style	Pure Java code	HTML + Java (scriptlets, expressions)	XML-based component tags (e.g., `h:inputText`)
UI Development	Tedious and tightly coupled with logic	Easier, with separation of presentation	Simplified via reusable components and tag libraries
Model Architecture	No clear MVC (manual implementa-tion)	Follows MVC loosely	Strict MVC architecture
Reusability	Low – mostly custom code	Moderate – via custom tags	High – through UI components, templates, and facelets

Summary

- **Servlets**: Best for logic-heavy tasks but complex for UI.
- **JSP**: Simplifies dynamic content with HTML-like syntax.
- **JSF**: Component-based, ideal for complex UIs.

When to Use:

- Servlets for APIs or microservices.
- JSP for simple web pages or views.
- JSF for component-driven, large-scale applications.

Modern Java Backend Development

Although **Servlets**, **JSP**, and **JSF** remain essential for understanding legacy systems and the core fundamentals of Java web technologies, modern backend development has largely shifted toward frameworks like **Spring Boot**.

Spring Boot provides:

- Easy configuration and installation
- Embedded servers such as Tomcat
- Built-in support for microservices and RESTful APIs
- Seamless integration with modern Java EE components
- Improved maintainability and scalability

Mastering Spring Boot alongside Servlets, JSP, and JSF equips developers to handle both legacy systems and modern enterprise applications efficiently.

Thank You for Reaching the End!

Thank you for taking the time to explore this guide. Whether you're just beginning your journey into Java backend development or revisiting core technologies like **Servlets**, **JSP**, and **JSF**, I hope this resource has been insightful and added meaningful value to your learning experience.

Need Help or Stuck Somewhere?
If you have questions or face challenges while working with these technologies, feel free to reach out — I'm always happy to support fellow developers on their learning journey.

Explore the Source Code
The complete codebase, including practical implementations and examples for Servlets, JSP, and JSF, is available on my GitHub.

`GitHub Repository Link`

Let's Connect
I'd love to hear from you, collaborate, or just connect over tech. Feel free to explore more or reach out through the links below:

Keep Learning. Keep Building.
Thanks once again for reading — wishing you continued success and growth in your development journey!

— [Rishabh Singh]