Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

JSP: Java Server Programming

Study

ARNIKA PATEL
CSE, PIT
Parul University

3.1 Introduction to JSP

- **JavaServer Pages (JSP)** is a server-side technology that creates dynamic web applications. It allows developers to embed Java code directly into HTML or XML pages, and it makes web development more efficient.
- JSP is an advanced version of Servlets. It provides enhanced capabilities for building scalable and platform-independent web pages.
- **Features of JSP**
    - It is platform-independent; we can write once, run anywhere.
    - It simplifies database interactions for dynamic content.
    - It contains predefined objects like request, response, session, and application, reducing development time.
    - It has built-in mechanisms for exception and error management.
    - It supports custom tags and tag libraries.
- **How is JSP More Advantageous than Servlets?**
    - JSP code is easier to manage than Servlets as it separates UI and business logic.
    - JSP minimizes the amount of code required for web applications.
    - Generate content dynamically in response to user interactions.
    - It provides access to the complete range of Java APIs for robust application development.
    - JSP is suitable for applications with growing user bases.
- **Why Use JSP?**
    - JSP is powerful because it allows us to:
    - Embed Java logic directly into HTML.
    - To create dynamic pages that respond to user actions.
    - To customize content for each user or session.

3.2 JSP Architecture

- JSP follows a three-layer architecture:
- Client Layer: The browser sends a request to the server.
- Web Server Layer: The server processes the request using a JSP engine.

- Database/Backend Layer: Interacts with the database and returns the response to the client.

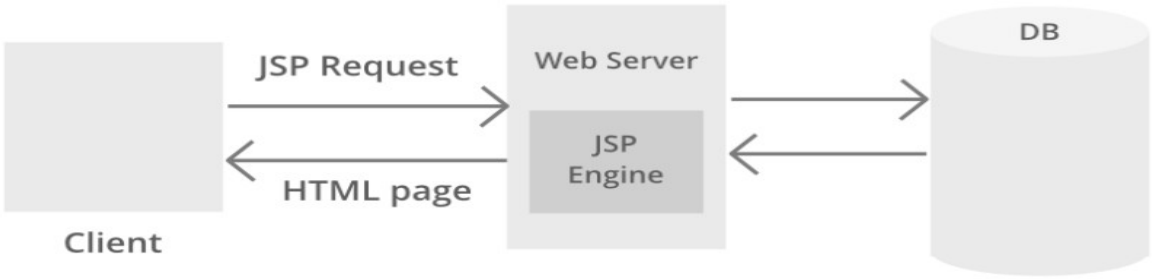

Fig. 3.1 JSP Architecture

- **JSP Processing** is illustrated and discussed in sequential steps prior to which a pictorial media is provided as a handful pick to understand the JSP processing better which is as follows:

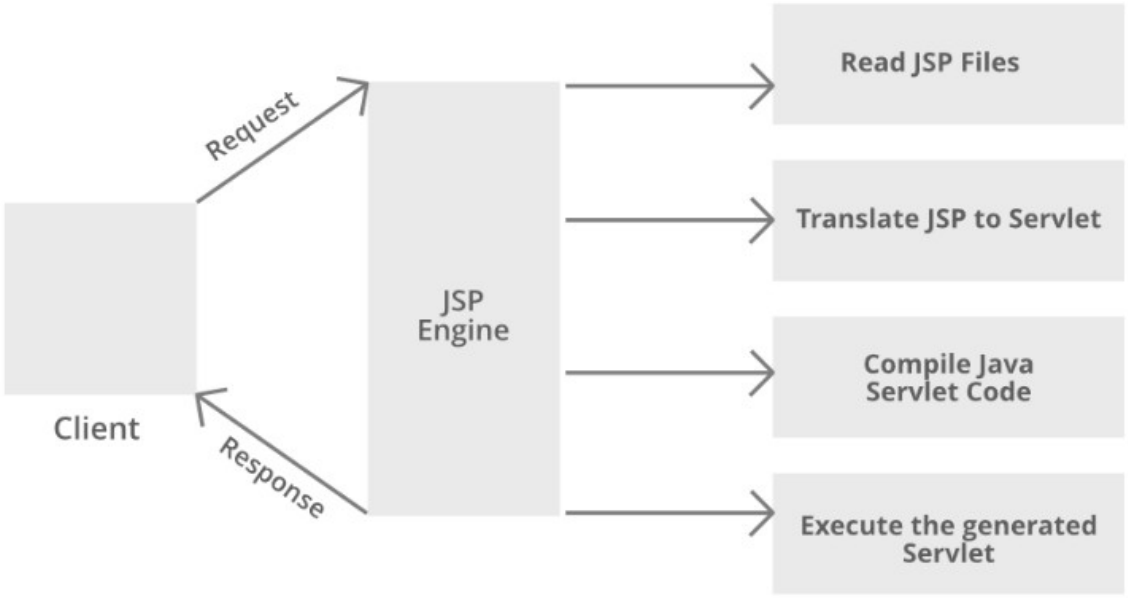

Fig. 3.2 JSP Processing

- **Step 1:** The client navigates to a file ending with the ***.jsp extension*** and the browser initiates an HTTP request to the webserver. **For example,** the user enters the login details and submits the button. The browser requests a status.jsp page from the webserver.
- **Step 2:** If the compiled version of JSP exists in the web server, it returns the file. Otherwise, the request is forwarded to the JSP Engine. This is done by recognizing the URL ending with **.jsp** extension.
- **Step 3:** The JSP Engine loads the JSP file and translates the JSP to Servlet(Java code). This is done by converting all the template text into println() statements and JSP elements to Java code. This process is called **translation.**
- **Step 4:** The JSP engine compiles the Servlet to an executable **.class** file. It is forwarded to the Servlet engine. This process is called **compilation** or **request processing phase.**

- **Step 5:** The **.class** file is executed by the Servlet engine which is a part of the Web Server. The output is an HTML file. The Servlet engine passes the output as an HTTP response to the webserver.
- **Step 6:** The web server forwards the HTML file to the client's browser.

3.3 JSP Lifecycle

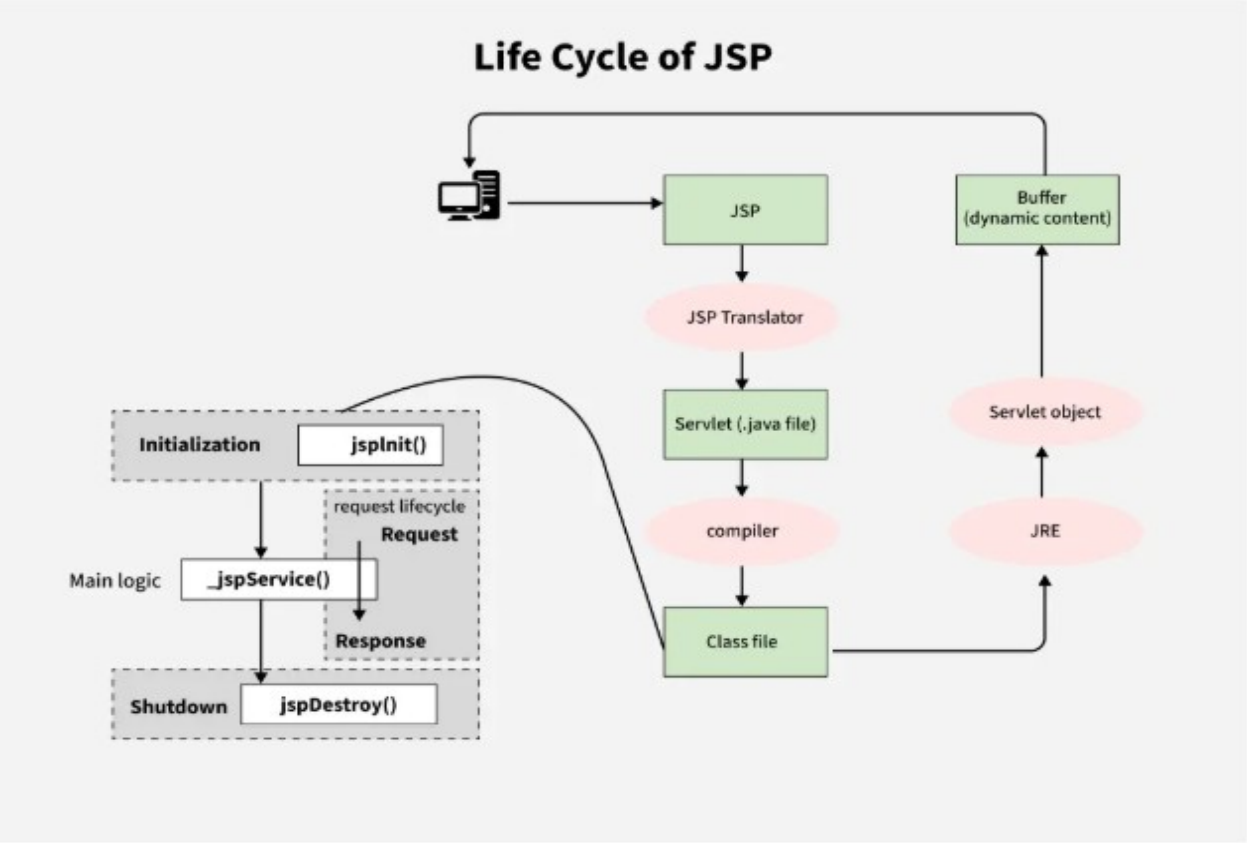

Fig. 3.2 JSP Lifecycle

- **Steps of JSP Life Cycle**

1. **Translation of JSP page to Servlet**
   - The JSP file is parsed and converted into a Java servlet source file (test.java).
   - This step checks for syntax correctness.

   public class TestServlet extends HttpServlet {

   // The generated servlet code

   }
2. **Compilation of JSP page(Compilation of JSP into test.java)**
   - The generated test.java file is compiled into a class file (test.class).

- This step converts the servlet code into bytecode.

```
public class TestServlet extends HttpServlet {
    // Generated servlet code here
}
```

3. **Classloading (test.java to test.class)**
   - The container dynamically loads the compiled class.

4. **Instantiation(Object of the generated Servlet is created)**
   - The container creates an instance of the generated servlet class. This instance handles multiple requests unless explicitly configured otherwise.

```
TestServlet servlet = new TestServlet();
```

5. **Initialization(jspInit() method is invoked by the container)**
   - This method is called only once when the JSP is first loaded. It is used for initializing resources like database connections or configurations.

```
public void jspInit() {
    // Initialization code, like setting up resources.
        System.out.println("JSP Initialized.");
 }
```

6. **Request processing(_jspService()is invoked by the container)**
   - This method is called for every request. It cannot be overridden because it is auto-generated and declared as final.
   - It receives HttpServletRequest and HttpServletResponse objects to handle the request.

```
public void _jspService(HttpServletRequest request, HttpServletResponse response)
{
// Code that handles the request, like generating HTML output.
response.getWriter().write("<html><body>Hello, World!</body></html>");
}
```

7. **JSP Cleanup (jspDestroy() method is invoked by the container)**
   - This method is called once before removing the JSP from service.It is used for releasing resources, such as closing database connections or cleaning up open files.

```
public void jspDestroy() {
// Clean up resources like closing database connections.
System.out.println("JSP Destroyed.");
```

}

This Lifecycle ensures that JSP pages are efficiently compiled, managed and cleaned up by the server container.

3.4 JSP Elements

In JSP elements can be divided into 4 different types.

- **Expression**
  - This tag is used to output any data on the generated page. These data are automatically converted to a string and printed on the output stream.
  - **Syntax:**
    <%= "Anything" %>
  - **Example:**
    <%="HelloWorld!" %>
- **Scriplets**
  - This allows inserting any amount of valid Java code. These codes are placed in the _jspService() method by the JSP engine.
  - **Syntax:**

```
<%
        // Java codes
%>
```

  - **Example:**

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

  - **Variables available to the JSP Scriptlets are:**
    - Request
    - Response
    - Session
    - Out
- **Directives**
- A JSP directive starts with <%@ characters. In the directives, we can import packages, define error-handling pages, or configure session information for the JSP page.
- **Syntax:**

```
<%@ directive attribute="value" %>
```

- **Types of Directives:**
  - page: It defines page settings.
  - include: It includes other files.
  - taglib: It declares a custom tag library.
- **Declarations**
- This is used for defining functions and variables to be used in the JSP.
- **Syntax:**

```
<%!
        //java codes
%>
```

References:

1. **Book Reference**

   Jim Farley, William Crawford, David Flanagan. Java Enterprise in a Nutshell, O'Reilly

2. **Book Reference**

   Rocha, R., Purificação, J. (2018). Java EE 8 Design Patterns and Best Practices: Build Enterprise-ready Scalable Applications with Architectural Design Patterns. Germany: Packt Publishing..

3. **Website Reference**

   https://www.scribd.com/document/268349254/Java-8-Programming-Black-Book .

4. **Sources**

   https://developers.redhat.com/topics/enterprise-java

5. **Article**

   https://www.researchgate.net/publication/276412369_Advanced_Java_Programming

Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

NAAC GRADE A++

https://paruluniversity.ac.in/