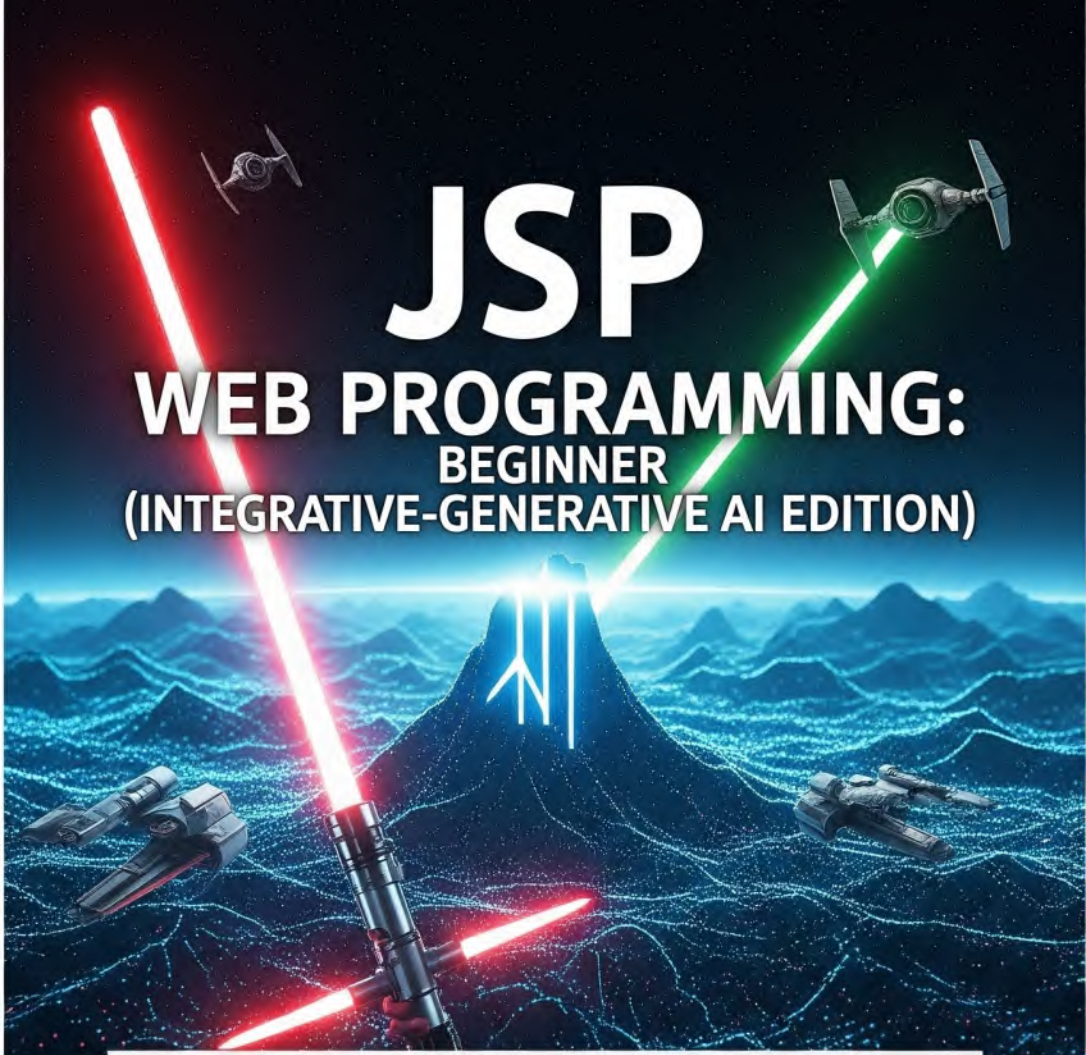

JSP

WEB PROGRAMMING:

BEGINNER (INTEGRATIVE-GENERATIVE AI EDITION)

Contents:

Author: Student Price Book Center

คำนำ

ในยุคที่การพัฒนาเว็บแอปพลิเคชันกลายเป็นทักษะพื้นฐานของนักพัฒนาเว็บไซต์ JSP (JavaServer Pages) ยังคงเป็นหนึ่งในเทคโนโลยีที่มีความสำคัญและทรงพลังในการสร้างเว็บแอปพลิเคชันแบบไดนามิก ด้วยจุดเด่นของภาษา Java ที่เสถียร ปลอดภัย และได้รับการสนับสนุนจากเครื่องมือพัฒนาหลากหลาย JSP จึงยังคงเป็นรากฐานสำคัญสำหรับการเรียนรู้ Full Stack Java Development

หนังสือ **"JSP Web Programming: Beginner"** เล่มนี้ ถูกออกแบบมาเพื่อเป็นคู่มือสำหรับผู้เริ่มต้นโดยเฉพาะ โดยครอบคลุมทุกหัวข้อสำคัญที่จำเป็นต่อการพัฒนา JSP ตั้งแต่ขั้นตอนติดตั้งสภาพแวดล้อม ไปจนถึงเทคนิคการจัดการข้อมูล การควบคุม Flow และการจัดการ Error อย่างมืออาชีพ โดยเนื้อหาได้เรียงลำดับอย่างเป็นระบบ พร้อมคำอธิบายเชิงลึก ตัวอย่างโค้ด และแนวทางปฏิบัติที่ดีที่สุด (Best Practices) เพื่อให้ผู้อ่านสามารถเรียนรู้และนำไปใช้ได้จริง

เริ่มต้นที่ **บทที่ 1** ผู้อ่านจะได้เรียนรู้การติดตั้ง Java Development Kit (JDK), Apache Tomcat และการใช้ IDE ยอดนิยม เช่น Eclipse, IntelliJ IDEA และ NetBeans พร้อมเทคนิคการเชื่อมต่อกับ JSP และการสร้างโปรเจกต์ JSP เบื้องต้นอย่างถูกต้อง เพื่อให้สามารถรันและทดสอบ JSP ได้บนเครื่องของตนเอง

ต่อมาใน **บทที่ 2** จะกล่าวถึงโครงสร้างของไฟล์ JSP โดยละเอียด ตั้งแต่การใช้ Directives (page, include, taglib), การฝัง Java ลงใน HTML ด้วย Scriptlets และ Expressions ตลอดจนการใช้งาน Implicit Objects ที่ JSP เตรียมไว้ให้ใช้งานทันที ซึ่งเป็นเครื่องมือสำคัญในการพัฒนาเว็บที่ตอบสนองได้อย่างมีประสิทธิภาพ **บทที่ 3** เน้นไปที่การจัดการข้อมูลระหว่าง Client และ Server เช่น การรับค่าจากแบบฟอร์มผ่าน GET และ POST, การอ่านค่าด้วย request.getParameter(), การส่งข้อมูลกลับด้วย response, การใช้งาน Session และ Cookies รวมถึงการส่งต่อข้อมูลระหว่างหน้า JSP ด้วย RequestDispatcher ซึ่งล้วนเป็นพื้นฐานของการสร้างเว็บแอปแบบอินเทอร์แอคทีฟ เมื่อระบบซับซ้อนขึ้น **บทที่ 4** จะพาผู้อ่านไปเรียนรู้การจัดการ Error และ Debugging อย่างมีระบบ ไม่ว่าจะเป็นการใช้ try-catch, การตั้ง Error Page ด้วย directive, การเข้าถึง object exception หรือการดู Log File จาก Apache Tomcat ซึ่งมีความสำคัญอย่างยิ่งในการพัฒนาระบบที่มีความเสถียรและตรวจสอบง่าย นอกจากนี้ ยังมี **ภาคผนวก** ที่อธิบายเพิ่มเติมเกี่ยวกับ "ตัวแปรใน JSP" และ "JSP Flow Control" ซึ่งจะช่วยปูพื้นฐานการใช้ตัวแปรในแต่ละ scope รวมถึงการควบคุมการทำงานของโปรแกรมด้วยคำสั่งเงื่อนไขและลูป ซึ่งล้วนเป็นทักษะสำคัญที่ใช้ในการพัฒนา JSP ในระดับจริงจัง

ขอให้ผู้อ่านได้รับประโยชน์สูงสุดจากหนังสือเล่มนี้ ทั้งในฐานะคู่มือการเรียนรู้ด้วยตนเอง และเป็นแหล่งอ้างอิงในการพัฒนาโปรเจกต์ JSP จริง พร้อมก้าวไปสู่การเป็นนักพัฒนาเว็บแอปพลิเคชันด้วย Java อย่างมั่นใจและมีระบบ.

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1
แนะนำ JSP และการติดตั้งสภาพแวดล้อม
(Introduction to JSP)

เนื้อหา

- แนะนำ JSP และการติดตั้งสภาพแวดล้อม
- รายละเอียดเชิงลึก: แนะนำ JSP และการติดตั้งสภาพแวดล้อม
- ความหมายของ JSP และการใช้งานในเว็บแอป
- ความแตกต่างระหว่าง JSP vs Servlet vs HTML ธรรมดา
- การทำงานร่วมกัน (Best Practice)
- JSP Lifecycle: Translation, Compilation, Execution
- การติดตั้ง JDK และ Apache Tomcat
- รายละเอียดเชิงลึกของการติดตั้ง IDE ยอดนิยม Eclipse, IntelliJ IDEA, และ NetBeans
- การเชื่อม IDE ยอดนิยมทั้ง 3 ตัว (Eclipse, IntelliJ IDEA, NetBeans) เข้ากับ JSP และ Apache Tomcat
- การสร้างโปรเจกต์ JSP เบื้องต้นใน IDE
- การรัน JSP บน Local Server (เช่น Apache Tomcat) และการเข้าถึงผ่านเว็บเบราเซอร์

บทนำ: แนะนำ JSP และการติดตั้งสภาพแวดล้อม

ในยุคที่เว็บไซต์เป็นส่วนสำคัญของการสื่อสาร การให้บริการ และการดำเนินธุรกิจ JavaServer Pages หรือ JSP จึงกลายเป็นเทคโนโลยีที่ทรงพลังสำหรับการสร้างเว็บแอปพลิเคชันแบบไดนามิก JSP ช่วยให้นักพัฒนาสามารถฝังโค้ด Java ลงในหน้า HTML ได้อย่างสะดวก ทำให้สามารถเชื่อมต่อกับฐานข้อมูล ประมวลผลข้อมูลผู้ใช้ และควบคุมลอจิกของแอปพลิเคชันได้อย่างยืดหยุ่นภายใต้โครงสร้างที่เข้าใจง่าย

JSP ไม่ได้ทำงานแบบเดียวกับ HTML หรือ Servlet เสมอไป ความแตกต่างที่สำคัญคือ HTML เป็นเพียงหน้าแสดงผลแบบคงที่ ไม่สามารถโต้ตอบกับผู้ใช้ได้มากนัก ส่วน Servlet คือ Java class ที่เขียนล้วน ๆ ไม่มี HTML ฝังอยู่ ทำให้การแสดงผลยุ่งยากกว่า JSP ที่รวมความสามารถทั้งสองไว้ในไฟล์เดียว ด้วยเหตุนี้ JSP จึงเหมาะสำหรับการสร้างอินเทอร์เฟซผู้ใช้ที่ตอบสนองแบบเรียลไทม์และเชื่อมโยงกับ backend ได้อย่างกลมกลืน

การทำงานของ JSP มี Lifecycle ที่ชัดเจน โดยเริ่มจากขั้นตอน **Translation** ซึ่ง JSP ถูกแปลงเป็น Servlet จากนั้นจึงเข้าสู่ขั้นตอน **Compilation** เพื่อแปลงเป็น Bytecode และสุดท้าย **Execution**

คือการรันภายใน Web Container เช่น Apache Tomcat กระบวนการนี้เกิดขึ้นโดยอัตโนมัติ ทำให้นักพัฒนาไม่ต้องจัดการเองทุกขั้นตอน แต่สามารถควบคุมพฤติกรรมของเว็บแอปได้ตามต้องการ

ก่อนเริ่มพัฒนา JSP นักพัฒนาจำเป็นต้องติดตั้งสภาพแวดล้อมให้พร้อมใช้งาน โดยเริ่มจากการติดตั้ง **Java Development Kit (JDK)** ซึ่งเป็นหัวใจหลักของการเขียนโปรแกรม Java ตามด้วย **Apache Tomcat** ซึ่งเป็น Web Server/Servlet Container ที่รองรับ JSP อย่างสมบูรณ์ การติดตั้งเครื่องมือเหล่านี้อย่างถูกต้องจะทำให้สามารถทดสอบและรันเว็บแอป JSP ได้ภายในเครื่องของตนเอง

เพื่อให้การพัฒนามีประสิทธิภาพมากขึ้น นักพัฒนาสามารถใช้ IDE (Integrated Development Environment) เช่น **Eclipse**, **IntelliJ IDEA** หรือ **NetBeans** ซึ่งรองรับการสร้างและรันโปรเจกต์ JSP ได้อย่างสะดวก พร้อมเครื่องมือช่วยเติมคำ ตรวจสอบ syntax และ deployment อัตโนมัติ ช่วยประหยัดเวลาในการพัฒนาอย่างมาก

เมื่อทุกอย่างพร้อม นักพัฒนาสามารถเริ่มต้นสร้างโปรเจกต์ JSP เบื้องต้น เช่น การสร้างไฟล์ .jsp แรกที่แสดงคำทักทาย "Hello, JSP!" และตั้งค่าโฟลเดอร์โปรเจกต์ให้อยู่ในโครงสร้างที่เหมาะสม โดยรวมไว้ใน WebContent หรือ webapp ขึ้นอยู่กับ IDE ที่เลือกใช้ พร้อมเชื่อมโยงกับ Web Server ให้สามารถเข้าถึงได้ผ่าน URL

การรัน JSP บน Local Server นั้นสามารถทำได้ผ่าน Web Browser โดยพิมพ์ URL ตามพอร์ตที่กำหนด เช่น http://localhost:8080/HelloJSP/hello.jsp ซึ่งจะนำไปสู่การประมวลผล JSP และแสดงผล HTML ที่ถูกแปลงจาก Java Logic ในหน้าเว็บ การทดสอบบนเครื่องตัวเองนี้จะช่วยให้นักพัฒนาทำงานแบบ iterative พัฒนา–ทดสอบ–ปรับปรุง ได้อย่างรวดเร็ว ก่อนนำขึ้นเซิร์ฟเวอร์จริงในขั้นตอนต่อไป

บทนำนี้เป็นจุดเริ่มต้นสำหรับผู้เรียนที่จะก้าวเข้าสู่โลกของ JSP อย่างมั่นใจ โดยในบทถัดไปคุณจะได้ลงมือเขียน JSP จริง ๆ พร้อมการประยุกต์ใช้งานร่วมกับฟอร์ม HTML, ฐานข้อมูล และเทคนิคการจัดการสถานะผู้ใช้ เพื่อสร้างเว็บแอปพลิเคชันที่มีความสมบูรณ์และมืออาชีพยิ่งขึ้น.

แนะนำ JSP และการติดตั้งสภาพแวดล้อม

☐ 1.1 ความหมายของ JSP และการใช้งานในเว็บแอป

JSP (JavaServer Pages) คือเทคโนโลยีของ Java ที่ช่วยให้สามารถฝัง Java code ลงในไฟล์ HTML ได้โดยตรง และแปลผลบนฝั่ง Server ก่อนส่ง HTML กลับไปยังผู้ใช้ (Client) ผ่าน Web Browser

ลักษณะเด่นของ JSP:

- ฝังโค้ด Java ลงใน HTML โดยใช้ <% %> syntax
- ทำงานบน **Web Container** เช่น Apache Tomcat
- เหมาะสำหรับเว็บแอปที่ต้องมีการตอบสนองแบบ dynamic
- รองรับการเชื่อมต่อฐานข้อมูล, การจัดการ session และ authentication
- รองรับการเขียนแยก logic (Java Bean) กับ UI

ตัวอย่างการใช้งาน:

```
<html>
<body>
    <%
        String name = "ChatGPT";
    %>
    <h1>Hello <%= name %>!</h1>
</body>
</html>
```

☐ **1.2 ความแตกต่าง JSP vs Servlet vs HTML ธรรมดา**

ประเภท	ลักษณะการใช้งาน	ความเหมาะสม
JSP	HTML + Java Scriptlet	เหมาะสำหรับ UI ที่มี logic แบบ dynamic
Servlet	Pure Java (ไม่มี HTML โดยตรง)	เหมาะสำหรับการประมวลผล logic ที่ซับซ้อน
HTML	Static markup	ใช้สำหรับเว็บเพจธรรมดาที่ไม่เปลี่ยนแปลง

JSP เป็นส่วนต่อยอดจาก Servlet — โดยเบื้องหลัง JSP จะถูกแปลง (translated) เป็น Servlet ก่อน แล้วค่อย compile และ execute

---

☐ **1.3 JSP Lifecycle: Translation, Compilation, Execution**

วงจรการทำงานของ JSP ประกอบด้วย 3 ขั้นตอนหลัก:

1. **Translation Phase (แปล JSP เป็น Servlet)**
    - JSP engine จะแปลงไฟล์ .jsp เป็น Java Servlet class (.java)
2. **Compilation Phase**
    - Java Compiler จะ compile ไฟล์ .java เป็น .class (Bytecode)
3. **Execution Phase**
    - Web Server เรียกใช้ Servlet class นี้โดยสร้าง HttpServletRequest และ HttpServletResponse

ภาพรวมการทำงาน:

.jsp ⟶ (แปล) ⟶ .java ⟶ (compile) ⟶ .class ⟶ (run) ⟶ HTML กลับสู่ client

หมายเหตุ: Web Container จะทำขั้นตอนนี้อัตโนมัติในครั้งแรกที่ JSP ถูกเรียก

☐ **1.4 การติดตั้ง JDK และ Apache Tomcat**

ขั้นตอน:

☐ **ติดตั้ง Java JDK (เช่น JDK 17 หรือสูงกว่า)**

- ดาวน์โหลดจาก https://jdk.java.net
- เพิ่ม path ของ JDK ลงในระบบ (เช่น JAVA_HOME, PATH)

☐ **ติดตั้ง Apache Tomcat (Web Server/Container)**

- ดาวน์โหลดจาก https://tomcat.apache.org
- แตก zip ไปยังโฟลเดอร์ที่ต้องการ เช่น C:\tomcat
- เรียกใช้งานผ่าน bin/startup.bat (Windows) หรือ startup.sh (Linux/Mac)

ทดสอบ:

เปิดเบราว์เซอร์แล้วเข้าที่

http://localhost:8080

หากเห็นหน้า Tomcat แสดงว่า server ทำงานแล้ว

☐ **1.5 การสร้างโปรเจกต์ JSP เบื้องต้นใน IDE (Eclipse, IntelliJ)**

☐ **วิธีสร้าง JSP Project บน Eclipse (EE version):**

1. เปิด Eclipse และเลือกเมนู File ⟶ New ⟶ Dynamic Web Project
2. ตั้งชื่อโปรเจกต์ เช่น HelloJSP
3. เลือก Target Runtime เป็น Apache Tomcat ที่ติดตั้งไว้
4. คลิก Finish

โครงสร้างโปรเจกต์จะได้แบบนี้:

```
HelloJSP/
├── WebContent/
│   └── index.jsp
├── WEB-INF/
│   └── web.xml
```

☐ **ตัวอย่างไฟล์ index.jsp:**

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
  <body>
    <h2>Hello from JSP!</h2>
    <p>Current time: <%= new java.util.Date() %></p>
  </body>
</html>
```

☐ **1.6 การรัน JSP บน Local Server และการเข้าถึงผ่านเบราเซอร์**

ขั้นตอน:

1. คลิกขวาที่ Project ⟶ Run As ⟶ Run on Server
2. เลือก Apache Tomcat ⟶ Finish
3. JSP จะรันที่ URL เช่น

   http://localhost:8080/HelloJSP/index.jsp

ตัวอย่างผลลัพธ์ในเบราว์เซอร์:

Hello from JSP!

Current time: Wed Jul 10 09:25:34 ICT 2025

☐ **สรุปความเข้าใจ**

หัวข้อ	สรุปสั้น
JSP คือ	HTML + Java รันฝั่ง Server
ทำงานยังไง	แปลงเป็น Servlet ⟶ Compile ⟶ รัน
ต่างจาก Servlet	JSP เน้น UI, Servlet เน้น Logic
ติดตั้ง	JDK + Tomcat + IDE (Eclipse/IntelliJ)
โครงสร้างไฟล์	index.jsp, web.xml, WEB-INF/
วิธีรัน	Run on Server ⟶ เปิดผ่านเบราว์เซอร์

รายละเอียดเชิงลึก: แนะนำ JSP และการติดตั้งสภาพแวดล้อม

รายละเอียดเชิงลึก: แนะนำ JSP และการติดตั้งสภาพแวดล้อม แยกเป็นหัวข้อย่อย พร้อมข้อมูลเบื้องหลังระดับเทคนิคเพื่อปูพื้นฐานอย่างมั่นคงสำหรับการพัฒนา JSP:

☐ **1.1 ความหมายของ JSP และการใช้งานในเว็บแอป (JSP Deep Dive)**

JSP คืออะไร?

- **JSP (JavaServer Pages)** เป็นเทคโนโลยีฝั่ง server ที่ออกแบบมาให้เขียน **HTML ที่ฝัง Java Code ได้** โดยทำงานร่วมกับ **Servlet API**
- ช่วยให้เขียนเว็บแบบ dynamic ได้โดยไม่ต้องเขียน Servlet ล้วน ๆ
- ใช้แนวคิดคล้าย PHP, ASP, Razor ที่ฝังโค้ด server ลงในหน้า HTML

ทำไมต้องใช้ JSP?

- ลดภาระในการจัดการ HTML ด้วย Servlet (ซึ่งต้องใช้ out.println() เยอะ)
- ผสมผสาน HTML + Java อย่างกลมกลืน
- รองรับ MVC Pattern โดย JSP ทำหน้าที่ “View”

ใช้งานในเว็บแอปอย่างไร?

- ทุกคำร้องขอ (request) ของผู้ใช้ เช่นการคลิกลิงก์หรือส่งฟอร์ม จะถูกส่งไปยังไฟล์ .jsp
- JSP จะทำงานฝั่ง server สร้าง HTML ที่เหมาะสม แล้วส่งกลับไปยัง browser

sequenceDiagram

Client->>JSP File: HTTP Request (.jsp)

JSP File->>JSP Engine: Compile & Translate

JSP Engine->>Servlet: Execute

Servlet->>HTML: Generate Response

HTML->>Client: Send HTML

☐ **1.2 ความแตกต่างระหว่าง JSP, Servlet และ HTML ธรรมดา**

Feature	HTML	Servlet	JSP
ประเภท	Static	Java class	Hybrid (HTML + Java)
ภาษา	HTML only	Java only	HTML + Java
รันบน	Browser	Web Container	Web Container
จุดเด่น	เร็ว, ง่าย	Logic จัดเต็ม	สะดวกเขียน UI แบบ dynamic
โครงสร้าง	UI อย่างเดียว	ต้องใช้ out.write()	เขียน HTML ได้โดยตรง
เหมาะสำหรับ	หน้าเว็บแบบคงที่	Controller/Business Logic	View Template

☐ **Tip:**

- JSP ถูกแปลงเป็น Servlet เบื้องหลัง
- สามารถส่ง Request ⟶ Servlet ⟶ สร้างข้อมูล ⟶ Forward มายัง JSP เพื่อ render

---

☐ **1.3 วงจรการทำงานของ JSP (JSP Lifecycle)**

1. **Request มาถึง JSP ครั้งแรก**
2. **JSP Engine (ใน Tomcat)** ตรวจสอบว่าเคย compile ไฟล์นี้หรือยัง
    - ถ้ายัง: แปลง .jsp ⟶ .java (extends HttpServlet)
    - แล้ว compile เป็น .class
3. **รัน JSP ที่เป็น Servlet แล้ว** สร้าง response ส่งกลับ client

ภาพรวมไฟล์แปลงอัตโนมัติ:

<!-- hello.jsp -->

<%= "Hello" %>

จะถูกแปลงเป็น:

```
public class hello_jsp extends HttpServlet {
   protected void _jspService(HttpServletRequest req, HttpServletResponse res) {
      res.getWriter().write("Hello");
   }
}
```

Lifecycle method สำคัญของ JSP:

Method	หน้าที่
_jspInit()	เรียกตอนเริ่ม JSP
_jspService()	เรียกทุกครั้งที่มี request
_jspDestroy()	เรียกตอน JSP ถูกลบ

☐ **1.4 การติดตั้ง JDK และ Apache Tomcat**

☐ **ติดตั้ง Java JDK**

- ดาวน์โหลด: https://jdk.java.net
- เพิ่ม Environment Variable (Windows):
  - JAVA_HOME ⟶ path ที่ติดตั้ง JDK
  - PATH ⟶ %JAVA_HOME%\bin

☐ **ติดตั้ง Tomcat**

- ดาวน์โหลด: https://tomcat.apache.org
- แตก zip ⟶ ตั้งค่า port, log, memory ได้ใน conf/server.xml
- ใช้งาน:
  - Windows: bin/startup.bat
  - macOS/Linux: bin/startup.sh

☐ **ทดสอบ**

เข้า http://localhost:8080 ต้องเห็นหน้า Tomcat dashboard

☐ **1.5 การสร้าง JSP Project บน IDE (Eclipse / IntelliJ IDEA)**

☐ **Eclipse (Enterprise Edition):**

1. File ⟶ New ⟶ Dynamic Web Project
2. ตั้งชื่อ ⟶ เลือก Target Runtime: Apache Tomcat
3. หน้า Project Explorer จะมี:
4. HelloJSP/
5. ├── WebContent/

```
6.  │   └── index.jsp
7.  └── WEB-INF/
8.      └── web.xml
```

web.xml (Deployment Descriptor):

```
<web-app>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

หมายเหตุ: ตั้งค่าให้ index.jsp ทำงานโดยอัตโนมัติเมื่อเปิด root path

---

☐ 1.6 การรัน JSP บน Local Server

วิธีการ:

- คลิกขวาที่โปรเจกต์ ⟶ Run on Server
- เลือก Tomcat
- รันที่: http://localhost:8080/HelloJSP/index.jsp

การเข้าถึงเบราว์เซอร์:

http://localhost:8080/HelloJSP/index.jsp

ข้อผิดพลาดที่พบบ่อย:

Error	สาเหตุ
404 Not Found	JSP ไฟล์ไม่มี หรือพิมพ์ path ผิด
Port Already in Use	มีโปรแกรมอื่นใช้ port 8080
JSP Compilation Error	เขียน Java code ผิด syntax
Server not starting	JAVA_HOME หรือ Tomcat path ไม่ถูกต้อง

☐ โครงสร้างโปรเจกต์ JSP ที่ดี (เบื้องต้น)

```
HelloJSP/
├── WebContent/
│   ├── index.jsp
│   ├── login.jsp
│   └── styles/
│       └── style.css
```

```
├── src/
│   └── com/example/
│       └── LoginServlet.java
├── WEB-INF/
│   ├── web.xml
│   └── lib/
│       └── (JAR dependencies)
```

☐ **เสริม: เครื่องมือที่แนะนำ**

เครื่องมือ	ความสามารถ
Eclipse EE	ทำ JSP ได้ดี มี Wizard สร้าง Project
IntelliJ IDEA Ultimate	มีตัวจัดการ Tomcat + JSP Debug
Tomcat Manager	จัดการ App, Restart, Undeploy
Postman	ทดสอบ request เมื่อเริ่มใช้ Servlet
JSTL + EL	ทำงานร่วมกับ JSP อย่างสะอาด

หากคุณต้องการ:

☐ โค้ด JSP แบบเต็มไฟล์

☐ ตัวอย่างหน้า Login + Session

☐ JSP + JDBC (ฐานข้อมูล MySQL)

☐ หรือ JSP + Servlet แบบ MVC

ความหมายของ JSP และการใช้งานในเว็บแอป

☐ **JSP คืออะไร?**

JSP (JavaServer Pages) คือเทคโนโลยีฝั่งเซิร์ฟเวอร์ที่พัฒนาโดย **Sun Microsystems** (ปัจจุบันคือ Oracle) ซึ่งทำงานบน **Java EE (Jakarta EE)** เพื่อใช้สร้าง **เว็บเพจแบบไดนามิก (Dynamic Web Pages)** โดยการฝัง **Java code** ลงในหน้า **HTML** ได้โดยตรง

JSP มีลักษณะคล้าย PHP, ASP.NET หรือ Razor ในโลกของภาษาอื่น คือช่วยให้คุณผสม **โค้ดฝั่งเซิร์ฟเวอร์** กับ HTML ในไฟล์เดียว

☐ **จุดเด่น:**

- ผสม Java กับ HTML ได้โดยตรง

- ใช้ซ้ำ code ได้ด้วย **JSP include**, **taglib**, **JSTL**
- ทำงานบน **Servlet container** เช่น Apache Tomcat
- สามารถเรียกใช้ **JavaBean**, **JDBC**, **Session**, **Cookies** ได้

□ โครงสร้างพื้นฐานของไฟล์ JSP

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
<head><title>Welcome</title></head>
<body>
    <%
        String username = "ผู้ใช้";
    %>
    <h1>ยินดีต้อนรับ <%= username %></h1>
</body>
</html>
```

ส่วนใน JSP	ความหมาย
<%@ page ... %>	Page Directive – กำหนดการตั้งค่าหน้า
<% %>	Scriptlet – เขียน Java Code
<%= %>	Expression – แสดงผลค่าตัวแปร

□ JSP กับ Web Application: ทำงานอย่างไร?

เมื่อผู้ใช้ร้องขอ (เช่นเปิด URL index.jsp) เบราว์เซอร์จะส่ง **HTTP Request** ไปยังเซิร์ฟเวอร์

เซิร์ฟเวอร์ (เช่น Tomcat):

1. ตรวจสอบไฟล์ .jsp
2. แปลงเป็น **Servlet Java class**
3. Compile ⟶ Run ⟶ สร้าง HTML
4. ส่งกลับ HTML ไปยังเบราว์เซอร์

⚙□ วงจรการทำงาน:

Request ⟶ JSP Engine (แปลง JSP ⟶ Servlet) ⟶ Compile ⟶ Run ⟶ Response (HTML)

□ เปรียบเทียบ JSP กับ HTML ธรรมดา

คุณสมบัติ	HTML ธรรมดา	JSP

คุณสมบัติ	HTML ธรรมดา	JSP
Dynamic Content	☐ ไม่ได้	☐ ได้
ควบคุมด้วย Java	☐ ไม่ได้	☐ ได้
ติดต่อฐานข้อมูล	☐ ไม่ได้	☐ ได้ (ผ่าน JDBC)
ใช้งาน Session / Cookie	☐ ไม่ได้	☐ ได้
ฝัง logic	☐ ไม่ได้	☐ ทำได้เต็มรูปแบบ

☐ **ตัวอย่างการใช้งาน JSP จริงในเว็บแอป**

☐ **ไฟล์: login.jsp**

```
<form method="post" action="LoginServlet">
  Username: <input type="text" name="username" />
  Password: <input type="password" name="password" />
  <input type="submit" value="Login" />
</form>
```

☐ **Servlet: LoginServlet.java**

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    String user = request.getParameter("username");
    String pass = request.getParameter("password");

    if("admin".equals(user) && "1234".equals(pass)) {
        request.getSession().setAttribute("user", user);
        response.sendRedirect("dashboard.jsp");
    } else {
        response.sendRedirect("login.jsp?error=true");
    }
}
```

☐ **dashboard.jsp**

```
<% String user = (String) session.getAttribute("user"); %>
<h2>Welcome, <%= user %></h2>
```

☐ **สรุปแนวคิดสำคัญ**

ประเด็น	รายละเอียด
JSP คือ	HTML ที่ฝัง Java เพื่อสร้างหน้าเว็บแบบ dynamic
ทำงานอย่างไร	แปลงเป็น Servlet แล้ว compile และ execute
ใช้งานในเว็บแอป	รับ input จากฟอร์ม, เชื่อมฐานข้อมูล, แสดงผลลัพธ์
จุดเด่น	สะดวกกว่า Servlet, ใช้ Java เต็มรูปแบบ, รันบนเซิร์ฟเวอร์
จุดอ่อน	ถ้า logic เยอะ ควรย้ายไป Servlet แล้วใช้ JSP แค่ View

ความแตกต่างระหว่าง JSP vs Servlet vs HTML ธรรมดา

✳ □ **เปรียบเทียบภาพรวม:**

คุณสมบัติ	HTML ธรรมดา	Servlet	JSP
ฝั่งที่ประมวลผล	Client (เบราว์เซอร์)	Server (Java)	Server (Java)
ภาษา	HTML, CSS, JS	Java ล้วน	Java + HTML
เน้นใช้งาน	สร้าง UI แบบคงที่	ควบคุม Logic	สร้าง UI แบบ Dynamic
ทำงานร่วมกับฐานข้อมูล	□ ไม่ได้โดยตรง	□ ได้เต็มรูปแบบ	□ ได้ (ผ่าน Java code)
ประมวลผลแบบ Dynamic	□ ไม่ได้	□ ได้	□ ได้
เหมาะสำหรับ	Static Page	Business Logic, Controller	View Template, UI
ต้อง Compile?	□ ไม่ต้อง	□ Compile class	□ แปลงเป็น Servlet แล้ว compile
ความยืดหยุ่น	ต่ำ	สูง	สูง (ด้าน UI)

□ 1. HTML ธรรมดา: (Static HTML)

ลักษณะ:

- ใช้สร้างหน้าเว็บคงที่ (Static)
- ไม่มี logic ใด ๆ บนเซิร์ฟเวอร์
- ไม่มีการเชื่อมต่อฐานข้อมูลหรือ session

- ถูกโหลดและแสดงโดย browser โดยตรง

ตัวอย่าง:

```
<html>
  <body>
    <h1>Welcome to my site!</h1>
  </body>
</html>
```

ข้อดี:

- โหลดเร็ว
- ง่าย

ข้อจำกัด:

- ไม่มีความสามารถฝั่งเซิร์ฟเวอร์
- ต้องใช้ภาษาอื่น (เช่น PHP หรือ JSP) ถ้าต้องการ Dynamic content

---

☐ **2. Servlet (Java Servlet)**

ลักษณะ:

- คือ Java class ที่ implement interface HttpServlet
- ใช้จัดการ request/response โดยตรง เช่น การอ่าน form, เขียน HTML ด้วย Java
- ทำงานผ่าน Servlet Container (เช่น Apache Tomcat)
- ต้อง compile และ deploy

ตัวอย่าง:

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {
    PrintWriter out = res.getWriter();
    out.println("<html><body>");
    out.println("<h1>Hello from Servlet</h1>");
    out.println("</body></html>");
  }
}
```

ข้อดี:

- เขียน logic ได้ยืดหยุ่นมาก
- เชื่อมต่อฐานข้อมูล, ตรวจสอบ session, คำนวณได้เต็มที่

ข้อจำกัด:

- การเขียน HTML ด้วย Java ทำให้โค้ดดูรก
- ไม่เหมาะกับงานด้าน UI

☐ 3. JSP (JavaServer Pages)

ลักษณะ:

- ไฟล์ .jsp ที่ผสม HTML กับ Java ได้
- เหมาะกับการแสดงผลลัพธ์ที่ dynamic เช่นตาราง, ฟอร์ม, รายการ
- ฝัง Java scriptlet (<% %>) ลงใน HTML ได้โดยตรง
- เบื้องหลังจะถูกแปลงเป็น Servlet โดยอัตโนมัติ

ตัวอย่าง:

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<html>
<body>
  <h1>Hello JSP</h1>
  <% out.print("Today: " + new java.util.Date()); %>
</body>
</html>
```

ข้อดี:

- สะดวกในการเขียน UI
- อ่านง่ายกว่า Servlet
- ใช้ร่วมกับ JSTL, EL ได้ เพื่อแยก logic ออกจาก view

ข้อจำกัด:

- ไม่เหมาะกับ logic ที่ซับซ้อน
- ถ้าฝัง Java เยอะเกิน จะอ่านยากและดูไม่เป็นระเบียบ (Anti-pattern)

การทำงานร่วมกัน (Best Practice)

ในระบบขนาดกลางขึ้นไป ควรใช้ JSP และ Servlet **ร่วมกันในรูปแบบ MVC (Model-View-Controller)**:

- **Servlet** = Controller ⟶ ประมวลผลข้อมูล, เรียกฐานข้อมูล, จัดการ session
- **JavaBean / DAO** = Model ⟶ โค้ดเชิงธุรกิจและข้อมูล
- **JSP** = View ⟶ แสดงผลให้ผู้ใช้เห็น

ตัวอย่าง Flow:

[Browser] ⟶ [Servlet] ⟶ [JavaBean/Database] ⟶ [set data in request] ⟶ [JSP View]

☐ **สรุปความแตกต่างแบบกระชับ**

หัวข้อ	HTML	Servlet	JSP
ตำแหน่งประมวลผล	Client	Server	Server
เหมาะกับ	Static Page	Logic/Controller	UI/View
เขียน HTML	✔ ☐ โดยตรง	☐ ต้องใช้ out.println()	✔ ☐ สะดวก
ฝัง Java ได้ไหม	☐	✔ ☐ เต็มรูปแบบ	✔ ☐ จำกัดเฉพาะ UI logic
ซับซ้อน	ง่ายมาก	ซับซ้อน	ปานกลาง

JSP Lifecycle: Translation, Compilation, Execution

หรือ "วงจรการทำงานของ JSP" ตั้งแต่ผู้ใช้ร้องขอไฟล์ .jsp จนได้รับหน้าเว็บที่แสดงผลลัพธ์

☐ **ภาพรวม JSP Lifecycle**

เมื่อผู้ใช้ร้องขอ (request) ไฟล์ .jsp ผ่านเบราว์เซอร์ เช่น http://localhost:8080/Hello/index.jsp
เซิร์ฟเวอร์ (เช่น Apache Tomcat) จะประมวลผลไฟล์ .jsp ด้วย **ขั้นตอนต่อไปนี้**:

1. Translation Phase (แปล JSP ⟶ Java Servlet class)
2. Compilation Phase (compile Java class ⟶ .class)
3. Loading & Instantiation Phase (โหลดและสร้าง instance)
4. Initialization (_jspInit())
5. Execution (_jspService())
6. Destruction (_jspDestroy())

☐ **1. Translation Phase (การแปล JSP เป็น Servlet)**

- JSP Engine จะตรวจสอบว่าไฟล์ .jsp ถูกแปลและ compile แล้วหรือยัง
- ถ้ายัง ⟶ จะ **แปลง JSP ⟶ Java Servlet class (.java)** โดยใช้โครงสร้างของ HttpServlet
- JSP แต่ละไฟล์จะถูกแปลงเป็น Java class ที่มีชื่อ เช่น index_jsp.java

☐ **ตัวอย่าง:**

```
<!-- index.jsp -->
<h1>Hello JSP</h1>
<%= new java.util.Date() %>
```

จะถูกแปลงเป็น:

```
public final class index_jsp extends HttpServlet {
    public void _jspService(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException {
        JspWriter out = response.getWriter();
        out.write("<h1>Hello JSP</h1>");
        out.write(new java.util.Date().toString());
    }
}
```

ไฟล์ .java นี้จะถูกเก็บไว้ในโฟลเดอร์ชั่วคราว เช่น work/Catalina/localhost/project/org/apache/jsp

---

☐ **2. Compilation Phase (การคอมไพล์ Java class เป็น Bytecode)**

- Java Compiler จะ compile ไฟล์ .java ⟶ .class (bytecode)
- ทำให้ไฟล์ .class พร้อมถูกโหลดเข้าสู่ JVM
- กระบวนการนี้เกิดขึ้น **โดยอัตโนมัติ** ในครั้งแรกที่ JSP ถูกร้องขอ

☐ **3. Loading & Instantiation Phase (โหลด class และสร้าง object)**

- Web Container (Tomcat) โหลด class ที่ compile แล้วเข้าสู่ JVM
- สร้าง instance ของ servlet ที่ได้จาก JSP
- เก็บไว้ใน memory และใช้ซ้ำใน request ถัดไป

☐ **4. Initialization Phase (_jspInit() method)**

- เรียกใช้เมธอด _jspInit() เพียงครั้งเดียวตอนที่ JSP ถูกโหลด
- ใช้สำหรับเตรียมการ (Initialization) เช่นเปิด connection ล่วงหน้า หรือโหลด config

```
public void _jspInit() {
  // เช่น โหลด Resource, อ่าน Properties
}
```

☐ **5. Execution Phase (_jspService() method)**

- เรียกใช้เมธอด _jspService(HttpServletRequest req, HttpServletResponse res) **ทุกครั้งที่มีคำร้องขอ (request)**
- เป็นเมธอดหลักที่ใช้ **สร้าง HTML output** และตอบกลับ client

```
public void _jspService(HttpServletRequest req, HttpServletResponse res) {
   // สร้าง HTML ด้วย out.write()
}
```

Developer ไม่สามารถ override _jspService() ได้ด้วยตนเอง

☐ **6. Destruction Phase (_jspDestroy() method)**

- เรียกเมื่อ JSP ถูกถอดออกจาก memory (undeployed หรือ server ปิดตัวลง)
- ใช้สำหรับการ **ปิด resource**, ปิด connection, clean up memory

```
public void _jspDestroy() {
   // เช่น conn.close();
}
```

☐ **สรุป JSP Lifecycle แบบภาพรวม**

graph TD

A[Client Request: index.jsp] --> B[Translation: index.jsp ⟶ index_jsp.java]

B --> C[Compilation: index_jsp.java ⟶ index_jsp.class]

C --> D[Loading & Instantiation: โหลด class เข้าสู่ JVM]

D --> E[_jspInit(): เรียกเมื่อโหลดครั้งแรก]

E --> F[_jspService(): เรียกทุกครั้งที่มี request]

F --> G[Response: ส่ง HTML กลับไปยัง Client]

G --> H[_jspDestroy(): เมื่อ server ปิดหรือ reload]

☐ **หมายเหตุเรื่อง "Cache"**

- JSP จะไม่แปลงใหม่ทุกครั้ง ถ้าไฟล์ .jsp **ยังไม่ถูกแก้ไข**
- หากคุณแก้ไข JSP แล้ว save, Container จะ **detect การเปลี่ยนแปลงแล้วแปลใหม่**
- ดังนั้น JSP เหมาะกับการพัฒนาเว็บที่ปรับเปลี่ยน UI บ่อย

☐ **สรุปตาราง JSP Lifecycle**

ขั้นตอน	รายละเอียด
1. Translation	JSP ⟶ Java Servlet (.java)
2. Compilation	Java ⟶ Bytecode (.class)
3. Loading	โหลด .class เข้า JVM
4. _jspInit()	เรียกครั้งแรกตอนเริ่มใช้งาน
5. _jspService()	เรียกทุก request
6. _jspDestroy()	เรียกตอน shutdown/unload

การติดตั้ง JDK และ Apache Tomcat

1. การติดตั้ง JDK (Java Development Kit)

JDK คือชุดเครื่องมือสำหรับพัฒนาโปรแกรมภาษา Java ซึ่งประกอบด้วย

- Java Compiler (javac)
- Java Runtime Environment (JRE)
- Java libraries และ tools อื่น ๆ

ขั้นตอนติดตั้ง JDK

1.1 ดาวน์โหลด JDK

- เข้าเว็บไซต์ https://jdk.java.net หรือ Oracle JDK
- เลือกเวอร์ชันที่ต้องการ เช่น JDK 17 LTS (Long Term Support) หรือเวอร์ชันล่าสุด

1.2 ติดตั้ง JDK

- รันไฟล์ติดตั้งที่ดาวน์โหลดมา
- เลือกโฟลเดอร์ติดตั้ง เช่น
    - Windows: C:\Program Files\Java\jdk-17.0.x
    - macOS/Linux: /Library/Java/JavaVirtualMachines/jdk-17.0.x.jdk/Contents/Home

1.3 กำหนด Environment Variables

Windows:

- เปิด System Properties > Advanced > Environment Variables
- สร้างตัวแปรใหม่ JAVA_HOME ชี้ไปที่โฟลเดอร์ JDK เช่น

  C:\Program Files\Java\jdk-17.0.x
- แก้ไขตัวแปร Path เพิ่ม

  %JAVA_HOME%\bin

macOS/Linux:

- แก้ไขไฟล์ ~/.bash_profile หรือ ~/.zshrc เพิ่มบรรทัด
- export JAVA_HOME=/path/to/jdk
- export PATH=$JAVA_HOME/bin:$PATH
- รันคำสั่ง source ~/.bash_profile เพื่อโหลด config ใหม่

1.4 ตรวจสอบการติดตั้ง

เปิด Command Prompt หรือ Terminal แล้วพิมพ์:

java -version

javac -version