Java Servlet Web Programming

(Integrative-Generative AI Edition)

Java EE and Servlet Fundamentals
Creating and Configurets Java Servlets
HTTP Protocol and Serviel Lifecycle
Form Handling and User Input in Servlets
Integrating Servlets JSP and Session
Session and Cookie Management
Session and Cookie Management
Database Connectiıy JDBC s the DAO Pattern
Filters, Listenrs, and Error Handling
Filters, Listeners, and Error Handling
Security, Authenticatior Web Services
Advanced Topics: WebSockets,
Microservices, and Cloud Deployment

Student Price Book Center

คำนำ

ในยุคที่เทคโนโลยีเว็บแอปพลิเคชันมีบทบาทสำคัญในการขับเคลื่อนนวัตกรรม การเรียนรู้การพัฒนาเว็บด้วย **Java Servlet** ถือเป็นรากฐานที่มั่นคงสำหรับนักพัฒนาซอฟต์แวร์ที่ต้องการเข้าใจระบบฝั่งเซิร์ฟเวอร์อย่างลึกซึ้ง หนังสือเล่มนี้จึงถูกจัดทำขึ้นเพื่อเป็นแนวทางการเรียนรู้ **Java Servlet ตั้งแต่ระดับพื้นฐานไปจนถึงระดับ Enterprise** โดยเรียงลำดับเนื้อหาอย่างเป็นระบบ พร้อมตัวอย่างโค้ดและแบบฝึกหัดในแต่ละบท เพื่อให้ง่ายต่อการปฏิบัติจริง

เริ่มต้นจากการปูพื้นฐานของ Java EE และ Servlet API เพื่อให้ผู้อ่านเข้าใจโครงสร้างหลักของเทคโนโลยี จากนั้นพาไปรู้จักกับการสร้าง Servlet แรก การทำงานของ HTTP Request/Response และ Lifecycle ของ Servlet ต่อด้วยการผสานการทำงานกับ **JSP, Form Handling และ Session Management** ซึ่งเป็นหัวใจของการพัฒนาเว็บแบบไดนามิก

เมื่อมีพื้นฐานเพียงพอ ผู้อ่านจะได้เรียนรู้เทคนิคระดับกลาง เช่น **การใช้ Filters, Listeners, การจัดการข้อผิดพลาด และระบบรักษาความปลอดภัย (Security & Authentication)** พร้อมทั้งการเชื่อมต่อฐานข้อมูลด้วย JDBC และการจัดโครงสร้างโค้ดด้วย **DAO Pattern**

ระดับสูงขึ้นไป หนังสือจะพาไปรู้จักกับการสร้าง **RESTful API**, การใช้ JSON กับ Servlet, การเขียน **WebSocket Application**, และการพัฒนาแอปพลิเคชันแบบ **Asynchronous Processing** รวมถึงแนวทางการพัฒนา **Microservices** ด้วย Spring Boot และการ Deploy ระบบด้วย Docker ท้ายที่สุด ผู้อ่านจะได้เข้าสู่การใช้งาน Servlet ในระดับ Enterprise ด้วยสถาปัตยกรรมแบบ **MVC (Model-View-Controller)** การใช้ **CDI, JPA และ EJB** ตลอดจนการทำ **CI/CD บน Cloud Platform** ด้วยเครื่องมือสมัยใหม่อย่าง Jenkins และ Kubernetes

หนังสือเล่มนี้เหมาะสำหรับทั้งผู้เริ่มต้นที่ต้องการปูพื้นฐานอย่างมั่นคง ไปจนถึงนักพัฒนาที่ต้องการยกระดับทักษะสู่ระดับ Enterprise โดยหวังเป็นอย่างยิ่งว่าจะเป็นคู่มือที่ช่วยให้ผู้อ่านสามารถนำ Servlet ไปประยุกต์ใช้ได้จริงในโปรเจกต์ของตนเอง และพัฒนาทักษะในสายงาน Java Web ได้อย่างมั่นใจ

“เริ่มจากพื้นฐานอย่างมั่นคง ก้าวสู่ระดับสูงอย่างเป็นระบบ”

คือแนวคิดหลักของหนังสือเล่มนี้

ขอให้ทุกท่านมีความสุขกับการเรียนรู้และเขียนโปรแกรมด้วย **Java Servlet** อย่างมีประสิทธิภาพ

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 1

พื้นฐาน Java EE และ Servlet API

(Java EE and Servlet API Basic)

เนื้อหา

- Java EE คืออะไร?
- ความแตกต่างระหว่าง Java SE และ Java EE
- โครงสร้างของ **Servlet API (javax.servlet และ javax.servlet.http)**
- วิธีติดตั้ง **Servlet Container** (Apache Tomcat, GlassFish)

ในยุคที่ระบบเว็บแอปพลิเคชันมีความซับซ้อนและต้องตอบสนองต่อผู้ใช้งานจำนวนมาก การพัฒนาแอปพลิเคชันที่มีความยืดหยุ่น ปรับขยายได้ง่าย และมีโครงสร้างชัดเจนจึงเป็นสิ่งสำคัญ หนึ่งในเทคโนโลยีที่ตอบโจทย์นี้ได้อย่างดีคือ **Java EE (Jakarta EE)** ซึ่งเป็นแพลตฟอร์มสำหรับพัฒนาแอปพลิเคชันระดับองค์กรที่มีขนาดใหญ่ โดยเฉพาะอย่างยิ่งในส่วนของการพัฒนาเว็บแอปพลิเคชันแบบเซิร์ฟเวอร์ฝั่ง Java ซึ่งมี **Servlet API** เป็นพื้นฐานสำคัญในการควบคุมการรับ-ส่งข้อมูลผ่าน HTTP

Java EE (Enterprise Edition) แตกต่างจาก **Java SE (Standard Edition)** ตรงที่ Java EE มีเครื่องมือและไลบรารีเฉพาะที่ออกแบบมาเพื่อรองรับงานระดับองค์กร เช่น EJB, JPA, CDI และแน่นอนว่า Servlet เองก็เป็นหนึ่งในองค์ประกอบหลักของ Java EE ด้วย โดย Servlet API มีโครงสร้างอยู่ในแพ็กเกจหลักคือ javax.servlet และ javax.servlet.http ซึ่งให้นักพัฒนาสามารถจัดการกับคำขอ (request) และผลลัพธ์ (response) ได้อย่างยืดหยุ่นและเป็นระบบ ไม่ว่าจะเป็นการจัดการ session, cookie, การรับค่าจากฟอร์ม หรือแม้แต่การควบคุมเส้นทาง (routing) ของแอปพลิเคชัน

ก่อนจะเริ่มเขียนโค้ด Servlet ได้ ผู้พัฒนาต้องติดตั้ง **Servlet Container** หรือ Web Server ที่รองรับ Servlet API เสียก่อน โดยในหนังสือเล่มนี้จะกล่าวถึงการติดตั้งและใช้งาน Apache Tomcat ซึ่งเป็น Servlet Container ยอดนิยม และ GlassFish ซึ่งเป็น Application Server ที่รองรับ Java EE อย่างเต็มรูปแบบ ผู้อ่านจะได้เรียนรู้วิธีการติดตั้ง การตั้งค่าเบื้องต้น รวมถึงวิธีนำโค้ด Servlet ไปรันในสภาพแวดล้อมจริง เพื่อวางรากฐานให้มั่นคงก่อนเข้าสู่การพัฒนาแอปพลิเคชันแบบเต็มรูปแบบในบทถัดไป

พื้นฐานความเข้าใจ Java EE และ Servlet API

☐ Java EE คืออะไร?

Java EE (Java Platform, Enterprise Edition) หรือชื่อใหม่ว่า **Jakarta EE** คือชุดเทคโนโลยีที่ออกแบบมาเพื่อการพัฒนา **Enterprise Applications** ซึ่งเป็นแอปพลิเคชันขนาดใหญ่ที่มีความซับซ้อน เช่น ระบบธนาคาร, ระบบโรงพยาบาล, ระบบขายสินค้าออนไลน์ ฯลฯ

Java EE ประกอบด้วย API และสเปกที่ออกแบบมาสำหรับ:

- การพัฒนา **Web Application (Servlet, JSP)**
- การจัดการฐานข้อมูล (JPA, JDBC)
- การควบคุม Transaction (JTA)
- การทำ Dependency Injection (CDI)
- การจัดการ Session และ Security
- การใช้ EJB, JMS, Web Services ฯลฯ

Java EE เป็นส่วนขยายของ **Java SE (Standard Edition)** โดยใช้พื้นฐานภาษา Java เดียวกันแต่เพิ่มขีดความสามารถทางฝั่งเซิร์ฟเวอร์

☐ **ความแตกต่างระหว่าง Java SE และ Java EE**

คุณสมบัติ	Java SE	Java EE
ขอบเขต	ใช้สร้างโปรแกรมทั่วไป เช่น Desktop App, Console App	ใช้สร้างระบบฝั่งเซิร์ฟเวอร์ เช่น Web App, Enterprise App
API หลัก	java.lang, java.util, java.io	javax.servlet, javax.ejb, javax.persistence, javax.websocket
เหมาะกับ	โปรแกรมเมอร์ทั่วไป, นักเรียน	นักพัฒนาเว็บ, นักพัฒนาระบบองค์กร
การทำงาน	ทำงานแบบ Standalone	ต้องรันใน Application Server เช่น Tomcat, GlassFish

---

☐ **โครงสร้างของ Servlet API**

☐ **Package หลักใน Servlet API:**

1. **javax.servlet** – ให้คลาสพื้นฐานของ Servlet เช่น Servlet, ServletRequest, ServletResponse, Filter, ServletContext
2. **javax.servlet.http** – ขยายจาก javax.servlet โดยรองรับโปรโตคอล HTTP เช่น:
    - HttpServlet
    - HttpServletRequest
    - HttpServletResponse
    - HttpSession
    - Cookie

☐ **ตัวอย่างการใช้ HttpServlet:**

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

public class HelloServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html");
        response.getWriter().println("<h1>Hello from Servlet!</h1>");
    }
}
```

HttpServlet เป็นคลาสหลักที่เราจะสืบทอดเพื่อเขียน Web App ฝั่ง Server

---

วิธีติดตั้ง Servlet Container (Apache Tomcat, GlassFish)

☐ **1. Apache Tomcat**

Tomcat เป็น Servlet Container ที่ได้รับความนิยมสูงสุด และรองรับสเปก Servlet/JSP

- **เว็บไซต์ดาวน์โหลด**: https://tomcat.apache.org/
- **ขั้นตอนติดตั้ง**:
    1. ดาวน์โหลด Apache Tomcat เวอร์ชันล่าสุด
    2. แตก zip ไฟล์
    3. รันคำสั่ง:
    4. ./bin/startup.sh # สำหรับ Linux/macOS
    5. ./bin/startup.bat # สำหรับ Windows
    6. เข้าสู่ระบบ: http://localhost:8080

☐ **2. GlassFish Server**

GlassFish เป็น Full Application Server ที่รองรับ Java EE (Jakarta EE) ทั้งชุด

- **เว็บไซต์ดาวน์โหลด**: https://projects.eclipse.org/projects/ee4j.glassfish
- **ขั้นตอนติดตั้ง**:
    1. ดาวน์โหลด GlassFish ZIP ไฟล์
    2. แตกไฟล์แล้วเข้าไปยังโฟลเดอร์ glassfish/bin
    3. เริ่มต้นเซิร์ฟเวอร์:
    4. ./asadmin start-domain

5. เข้าไปที่ http://localhost:8080 (เว็บแอป) และ http://localhost:4848 (admin console)

☐ ใช้งานร่วมกับ NetBeans

NetBeans สามารถเชื่อมต่อกับทั้ง Tomcat และ GlassFish ได้อย่างง่าย:

- ไปที่ **Tools > Servers > Add Server**
- เลือกชนิดเซิร์ฟเวอร์ที่ต้องการติดตั้ง
- ระบุ path ของ Tomcat หรือ GlassFish ที่ติดตั้งไว้
- เมื่อสร้าง Project ใหม่ เลือกชนิดว่าเป็น "Java Web with Ant" หรือ "Maven Web App" และผูกกับ Server ที่ตั้งไว้

☐ สรุป

ในหัวข้อนี้เราได้เรียนรู้ว่า:

- **Java EE** คือชุดเครื่องมือสำหรับพัฒนาเว็บและระบบองค์กร
- **Java SE** เหมาะกับโปรแกรมทั่วไป ขณะที่ Java EE เหมาะกับระบบฝั่ง Server
- **Servlet API** มีโครงสร้างที่แบ่งเป็น javax.servlet และ javax.servlet.http
- การติดตั้ง **Servlet Container** อย่าง Tomcat หรือ GlassFish เป็นขั้นตอนเริ่มต้นที่สำคัญ

เพิ่มเติมเชิงลึก สำหรับแต่ละหัวข้อใน **ทำความเข้าใจ Java EE และ Servlet API** โดยละเอียด พร้อมคำอธิบายเสริมและตัวอย่างประกอบ:

☐ Java EE (Jakarta EE) คืออะไร?

Java EE (หรือชื่อใหม่ว่า **Jakarta EE** ตั้งแต่ปี 2019 เป็นต้นมา) คือแพลตฟอร์มที่ Oracle และชุมชน Jakarta EE พัฒนาร่วมกันเพื่อให้ใช้สร้างระบบเว็บและแอปพลิเคชันระดับองค์กร ซึ่งรวม:

- **Web Layer** (Servlet, JSP, JSF)
- **Business Layer** (EJB, CDI)
- **Persistence Layer** (JPA)
- **Integration Layer** (JMS, Web Services)

☐ จุดเด่นของ Java EE:

- ทำงานบน **Application Server** ที่จัดการ Lifecycle, Thread, Security ให้โดยอัตโนมัติ
- รองรับ **Dependency Injection (DI)** และ **MVC**
- มีมาตรฐานกลาง ทำให้โค้ดย้ายไประบบอื่นได้ง่าย

☐ ความแตกต่างระหว่าง Java SE กับ Java EE (Jakarta EE)

หัวข้อ	Java SE	Java EE

หัวข้อ	Java SE	Java EE
ใช้ทำอะไร	แอปทั่วไป: Desktop, Console	Web App, ระบบองค์กร
API	java.lang, java.io, java.util	javax.servlet, javax.ejb, javax.persistence
การจัดการ	เขียนโค้ดควบคุมเอง	Managed by Container (เช่น GlassFish, Tomcat)
สถาปัตยกรรม	Standalone	Client/Server หรือ Distributed
ตัวอย่าง	โปรแกรมคำนวณ, เกมเล็ก ๆ	เว็บขายของ, ระบบบัญชี, ระบบโรงพยาบาล

☐ **โครงสร้างของ Servlet API**

☐ **javax.servlet:**

- Servlet: อินเทอร์เฟซหลักที่ทุก Servlet ต้อง implement
- ServletRequest: ข้อมูลคำร้องจาก client
- ServletResponse: การตอบกลับจาก server
- Filter: ดักจับ request/response ก่อน/หลังเข้า servlet
- ServletContext: ข้อมูลระดับแอปพลิเคชัน

☐ **javax.servlet.http:**

- HttpServlet: คลาสพื้นฐานที่ใช้งานง่ายสำหรับ HTTP
- HttpServletRequest: รับข้อมูล URL, form, cookie, session
- HttpServletResponse: สร้าง HTML หรือ JSON ตอบกลับ
- HttpSession: จัดการ session ของผู้ใช้
- Cookie: จัดการข้อมูลฝั่ง client (จำ username ได้ ฯลฯ)

☐ **ตัวอย่างการใช้ HttpServletRequest:**

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String username = request.getParameter("username");
    response.getWriter().println("Welcome, " + username);
}
```

☐ **การติดตั้ง Servlet Container**

☐ **Apache Tomcat (เหมาะสำหรับเริ่มต้น)**

Tomcat เป็น Web Container ที่เบาและนิยมมาก

ขั้นตอน:

1. ดาวน์โหลดจาก https://tomcat.apache.org

2. แตก zip และเข้าโฟลเดอร์ /bin
3. รัน:
4. startup.bat (Windows)
5. ./startup.sh (Linux/macOS)
6. เปิดเบราว์เซอร์: http://localhost:8080

✔ □ **โฟลเดอร์สำคัญ:**

- /webapps/ – ไฟล์ .war ที่ deploy จะอยู่ที่นี่
- /conf/ – ใช้กำหนดค่า เช่น server.xml, web.xml

□ **GlassFish (Full Java EE Server)**

GlassFish รองรับทุก API ของ Java EE เช่น EJB, JPA, CDI, JMS

ขั้นตอน:

1. ดาวน์โหลดจาก https://glassfish.org
2. แตกไฟล์ แล้วเปิดเทอร์มินัล:
3. ./asadmin start-domain
4. เบราว์เซอร์:
    - แอป: http://localhost:8080
    - Admin Console: http://localhost:4848

□ **เสริม: วิธี Deploy Servlet บน NetBeans (กับ Tomcat หรือ GlassFish)**

1. **ติดตั้ง Server ใน NetBeans**
    - Tools ⟶ Servers ⟶ Add Server
    - เลือก **Apache Tomcat** หรือ **GlassFish**
    - ระบุ path ไปยังโฟลเดอร์ที่ติดตั้งไว้
2. **สร้างโปรเจกต์ Servlet ใหม่**
    - File > New Project > Java Web > Web Application
    - ตั้งชื่อ > เลือก Server > เลือก Java EE version > Finish
3. **สร้าง Servlet**
    - คลิกขวาที่ "Source Packages" ⟶ New ⟶ Servlet
    - ตั้งชื่อเช่น HelloServlet และ URL pattern เช่น /hello
4. **เขียนโค้ด**

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
```

```
        throws ServletException, IOException {
        res.getWriter().println("<h1>Hello World!</h1>");
    }
}
```

5. **รันโปรเจกต์** ⟶ เบราว์เซอร์จะเปิดไปยัง http://localhost:8080/YourApp/hello

---

☐ **สรุปภาพรวม**

หัวข้อหลัก	รายละเอียด
Java EE	แพลตฟอร์มสำหรับ Web & Enterprise Apps
Java SE vs EE	EE มีเครื่องมือจัดการฝั่ง Server
Servlet API	มี javax.servlet และ javax.servlet.http
Tomcat	Web Container เบา ใช้ง่าย
GlassFish	Full Java EE Server ใช้ในองค์กรใหญ่
NetBeans	IDE ที่ผูกกับ Server และสร้าง Servlet ได้ง่าย

สรุป

Java EE (หรือ Jakarta EE ในปัจจุบัน) คือแพลตฟอร์มที่พัฒนาต่อยอดจาก Java SE เพื่อรองรับการสร้างแอปพลิเคชันระดับองค์กร โดยเฉพาะระบบที่มีความซับซ้อน เช่น เว็บแอปพลิเคชันและระบบกระจายศูนย์ (distributed systems) ความแตกต่างสำคัญระหว่าง Java SE และ Java EE คือ Java SE ให้เครื่องมือพื้นฐานสำหรับเขียนโปรแกรมทั่วไป ในขณะที่ Java EE เพิ่ม API และคอมโพเนนต์ที่ออกแบบมาเพื่อจัดการงานฝั่งเซิร์ฟเวอร์ เช่น Servlet, JSP, JPA, EJB และอื่น ๆ ซึ่งช่วยให้นักพัฒนาสามารถเขียนแอปพลิเคชันที่มีโครงสร้างชัดเจนและขยายได้ง่ายขึ้น

หนึ่งในองค์ประกอบหลักของ Java EE คือ **Servlet API** ซึ่งแบ่งออกเป็นสองแพ็กเกจหลัก ได้แก่ javax.servlet ที่ใช้จัดการกับคอนเทนเนอร์โดยรวม และ javax.servlet.http ที่ใช้จัดการกับ HTTP Protocol โดยตรง เช่น คำสั่ง GET, POST, การจัดการ session และ cookie สำหรับการทดสอบและใช้งาน Servlet จริง จำเป็นต้องติดตั้ง **Servlet Container** เช่น Apache Tomcat ซึ่งเหมาะสำหรับการพัฒนาและรันแอปพลิเคชัน Servlet และ JSP เบื้องต้น หรือ GlassFish ที่รองรับ Java EE อย่างครบถ้วน เหมาะสำหรับงานระดับองค์กร การติดตั้งและตั้งค่าเหล่านี้เป็นขั้นตอนพื้นฐานที่ช่วยให้นักพัฒนาสามารถเริ่มต้นสร้างเว็บแอปพลิเคชันด้วย Java ได้อย่างมีประสิทธิภาพ

บทที่ 2

เริ่มต้นสร้าง Java Servlet

(First Java Servlet)

เนื้อหา

- ติดตั้ง **NetBeans + GlassFish** หรือ **Tomcat**
- การเขียน **Basic Servlet (HttpServlet)**
- การกำหนดค่า **URL Mapping (web.xml, @WebServlet)**

หลังจากที่ผู้อ่านได้เรียนรู้แนวคิดพื้นฐานของ Java EE และโครงสร้างของ Servlet API ไปแล้ว ในบทนี้เราจะเข้าสู่การปฏิบัติจริง ด้วยการสร้าง Java Servlet ตัวแรกของเรา โดยเริ่มจากการเตรียมเครื่องมือพัฒนาและสภาพแวดล้อมให้พร้อมใช้งาน ซึ่งจะช่วยให้ผู้อ่านสามารถทดสอบโค้ดและทดลองรันแอปพลิเคชันแบบไดนามิกได้อย่างราบรื่น บทนี้จะเน้นการติดตั้งและใช้งาน **NetBeans** คู่กับ **GlassFish** หรือ **Apache Tomcat** ซึ่งเป็นเครื่องมือยอดนิยมและใช้งานได้ฟรี

NetBeans IDE เป็นเครื่องมือที่เหมาะสำหรับผู้เริ่มต้น เนื่องจากรองรับการพัฒนาเว็บแอปพลิเคชันด้วย Java EE อย่างครบถ้วน และสามารถเชื่อมต่อกับ GlassFish หรือ Tomcat ได้โดยตรงโดยไม่ต้องตั้งค่ามากนัก หลังจากติดตั้ง IDE และ Web Server เสร็จเรียบร้อย ผู้อ่านจะได้เรียนรู้การสร้าง Project แบบ Web Application และเริ่มต้นเขียน **Servlet เบื้องต้น** ด้วยการสืบทอดคลาส HttpServlet ซึ่งเป็นโครงสร้างพื้นฐานสำหรับจัดการคำขอและการตอบกลับผ่าน HTTP

เนื้อหาจะเริ่มจากการเขียน Servlet ที่แสดงข้อความง่าย ๆ บนหน้าเว็บ โดยใช้เมธอด doGet() ซึ่งจะถูกเรียกเมื่อมีคำขอแบบ HTTP GET เข้ามา จากนั้นจะใช้ HttpServletRequest และ HttpServletResponse ในการจัดการข้อมูลเข้า-ออก และส่ง HTML กลับไปยังเบราว์เซอร์ของผู้ใช้ การเข้าใจโครงสร้างนี้จะเป็นพื้นฐานสำคัญสำหรับการพัฒนาเว็บแอปพลิเคชันแบบไดนามิกต่อไปในอนาคต

สุดท้าย บทนี้จะอธิบายวิธีการกำหนด **URL Mapping** เพื่อให้สามารถเข้าถึง Servlet ผ่านเว็บเบราว์เซอร์ได้ ทั้งในรูปแบบของการกำหนดในไฟล์ web.xml และการใช้ annotation อย่าง @WebServlet ซึ่งเป็นวิธีที่สะดวกและทันสมัยกว่า การกำหนด URL Mapping อย่างถูกต้องจะทำให้ผู้ใช้สามารถเรียกใช้งาน Servlet ได้ผ่าน URL ที่ชัดเจนและเป็นระบบ พร้อมรองรับการพัฒนาในสเกลที่ใหญ่ขึ้นในบทถัดไป

การติดตั้ง NetBeans + GlassFish หรือ Tomcat

☐ **ขั้นตอนที่ 1: ดาวน์โหลด NetBeans**

- ไปที่: https://netbeans.apache.org/download/

- เลือก **"Java EE" หรือ "All" bundle** ⟶ ติดตั้ง

☐ **ขั้นตอนที่ 2: ติดตั้ง GlassFish หรือ Tomcat**

☐ **ติดตั้ง GlassFish:**

1. ไปที่: https://projects.eclipse.org/projects/ee4j.glassfish/downloads
2. ดาวน์โหลด ⟶ แตก zip ไว้ในที่ปลอดภัย
3. ใน NetBeans:
    - เมนู Tools ⟶ Servers ⟶ Add Server
    - เลือก GlassFish ⟶ Browse ไปยังโฟลเดอร์ที่แตก zip

☐ **หรือติดตั้ง Tomcat:**

1. ไปที่: https://tomcat.apache.org/download-90.cgi
2. ดาวน์โหลด ⟶ แตก zip
3. ใน NetBeans: Tools ⟶ Servers ⟶ Add Server ⟶ เลือก Tomcat

☐ **2. การเขียน Basic Servlet (HttpServlet)**

☐ **ขั้นตอนที่ 1: สร้างโปรเจกต์ Web Application**

1. File ⟶ New Project ⟶ Java Web ⟶ Web Application
2. ตั้งชื่อโปรเจกต์ เช่น MyFirstServlet
3. เลือก Server (Tomcat หรือ GlassFish)
4. เลือก Java EE 8 (หรือ Jakarta EE 10 ขึ้นไป) แล้วคลิก Finish

---

☐ **ขั้นตอนที่ 2: สร้าง Servlet ใหม่**

1. คลิกขวาที่ Source Packages ⟶ New ⟶ Servlet
2. ตั้งชื่อ: HelloServlet
3. ตั้ง URL Mapping: /hello

☐ **โค้ดตัวอย่าง: HelloServlet.java**

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
// กำหนด URL Mapping ด้วย Annotation
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {

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

        response.setContentType("text/html;charset=UTF-8");

        // เขียน HTML ออกไปยังเบราว์เซอร์
        response.getWriter().println("<html><body>");
        response.getWriter().println("<h1>Hello from Java Servlet!</h1>");
        response.getWriter().println("</body></html>");
    }
}
```

☐ **3. การกำหนดค่า URL Mapping**

☐ **วิธีที่ 1: ใช้ Annotation (@WebServlet)**

เป็นวิธีแนะนำ เพราะง่ายและไม่ต้องแก้ไขไฟล์ web.xml

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet { ... }
```

เมื่อเข้าที่ http://localhost:8080/YourAppName/hello จะเห็นผลลัพธ์

☐ **วิธีที่ 2: ใช้ไฟล์ web.xml (สำหรับ Java EE เวอร์ชันเก่า)**

ไฟล์นี้อยู่ใน: Web Pages/WEB-INF/web.xml

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">
    <servlet>
        <servlet-name>HelloServlet</servlet-name>
        <servlet-class>HelloServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>HelloServlet</servlet-name>
```

```
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
</web-app>
```

☐ ผลลัพธ์ที่ได้

เมื่อรันโปรเจกต์แล้วเปิดเบราว์เซอร์:

http://localhost:8080/MyFirstServlet/hello

จะเห็นข้อความ:

```
<h1>Hello from Java Servlet!</h1>
```

☐ สรุปภาพรวม

หัวข้อ	รายละเอียด
NetBeans + Server	ใช้ NetBeans ร่วมกับ GlassFish หรือ Tomcat
Basic Servlet	สร้าง class ที่ extends HttpServlet แล้ว override doGet()
การ Mapping	กำหนด URL ด้วย @WebServlet("/path") หรือแก้ใน web.xml
การใช้งาน	เรียกผ่านเบราว์เซอร์ตาม URL Mapping ที่ตั้งไว้

ต่อไปนี้คือตัวอย่างเพิ่มเติมที่แสดงการ **เขียน Java Servlet เบื้องต้น** อย่างละเอียด พร้อมการ **รับค่า Form**, **แสดงผล HTML**, และ **URL Mapping** ครบถ้วนครับ:

☐ ตัวอย่างที่ 1: Servlet แสดงข้อความต้อนรับจาก Form

☐ 1. ฟอร์ม HTML (form.html)

```
<!DOCTYPE html>
<html>
<head>
    <title>Welcome Form</title>
</head>
<body>
    <h2>กรอกชื่อของคุณ:</h2>
    <form action="welcome" method="POST">
        <input type="text" name="name" placeholder="กรอกชื่อของคุณ" required>
        <input type="submit" value="ส่ง">
    </form>
```

```
</body>
</html>
```

วางไว้ในโฟลเดอร์ Web Pages ใน NetBeans

☐ 2. Servlet รับค่าจากฟอร์ม (WelcomeServlet.java)

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/welcome") // URL Mapping
public class WelcomeServlet extends HttpServlet {

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

        // ดึงค่าชื่อจากฟอร์ม
        String name = request.getParameter("name");

        // ตั้งค่า content type
        response.setContentType("text/html;charset=UTF-8");

        // แสดงผล HTML ตอบกลับ
        response.getWriter().println("<html><body>");
        response.getWriter().println("<h1>สวัสดีคุณ " + name + "!</h1>");
        response.getWriter().println("</body></html>");
    }
}
```

☐ 3. การรันโปรแกรม

1. รันโปรเจกต์ใน NetBeans

2. เปิดเบราว์เซอร์ไปที่:

http://localhost:8080/YourProjectName/form.html

3. พิมพ์ชื่อ ⟶ คลิก "ส่ง" ⟶ ระบบจะแสดงข้อความ:

สวัสดีคุณ สมชาย!

ตัวอย่างที่ 2: การใช้ web.xml แทน Annotation

หากคุณไม่ต้องการใช้ @WebServlet ให้ใช้ไฟล์ web.xml ดังนี้:

web.xml

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee" version="3.1">
    <servlet>
        <servlet-name>WelcomeServlet</servlet-name>
        <servlet-class>WelcomeServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>WelcomeServlet</servlet-name>
        <url-pattern>/welcome</url-pattern>
    </servlet-mapping>
</web-app>
```

หมายเหตุ: ถ้าใช้ web.xml อย่าใส่ @WebServlet ซ้ำในคลาส

ตัวอย่างที่ 3: สร้าง Servlet ที่แสดงวันและเวลา

```
import java.io.IOException;
import java.time.LocalDateTime;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/time")
public class TimeServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        LocalDateTime now = LocalDateTime.now();
```

```
        response.setContentType("text/html;charset=UTF-8");
        response.getWriter().println("<html><body>");
        response.getWriter().println("<h2>เวลาปัจจุบันคือ: " + now + "</h2>");
        response.getWriter().println("</body></html>");
    }
}
```

เปิดที่:

http://localhost:8080/YourProjectName/time

☐ **สรุปความเข้าใจเพิ่มเติม**

สิ่งที่เรียนรู้	รายละเอียด
HTML Form ⟶ Servlet	รับค่า input จากผู้ใช้ผ่าน POST
@WebServlet	ใช้กำหนด URL Path แบบ Annotation
request.getParameter()	ดึงค่าที่ผู้ใช้กรอกจากฟอร์ม
การแสดงผล HTML	ใช้ response.getWriter().println()

หากต้องการ:

- ☐ ไฟล์โปรเจกต์ NetBeans ตัวอย่าง (พร้อม .zip)
- ☐ การรับค่า GET/POST หลายฟิลด์ (name, email ฯลฯ)
- ☐ การเขียน Servlet เชื่อมต่อฐานข้อมูล (MySQL)
- ☐ ตัวอย่าง Login + Session ด้วย Servlet + JSP

ตัวอย่าง Java Servlet เพิ่มเติมอีก 5 ตัวอย่าง พร้อมคำอธิบายเพื่อให้เข้าใจได้ง่าย:

ตัวอย่าง 1: Servlet แสดงข้อความ "Hello World" แบบง่ายสุด

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/hello")
public class HelloWorldServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        response.getWriter().println("<h1>Hello World from Servlet!</h1>");
    }
}
```

เรียกใช้งานที่ URL: /hello

ตัวอย่าง 2: Servlet รับค่าพารามิเตอร์แบบ GET แล้วแสดงผล

```
@WebServlet("/greet")
public class GreetingServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String user = request.getParameter("user");
        if (user == null || user.isEmpty()) {
            user = "Guest";
        }
        response.setContentType("text/html;charset=UTF-8");
        response.getWriter().println("<h2>สวัสดี, " + user + "!</h2>");
    }
}
```

เรียกใช้งาน เช่น:

http://localhost:8080/YourProject/greet?user=Somchai

ตัวอย่าง 3: Servlet ส่ง Redirect ไปยัง URL อื่น

```
@WebServlet("/redirect")
public class RedirectServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        // ส่งไปยัง Google
        response.sendRedirect("https://www.google.com");
    }
}
```

เรียก /redirect แล้วจะถูกส่งต่อไปยัง Google โดยอัตโนมัติ

ตัวอย่าง 4: Servlet ส่ง Forward ไปยัง JSP

```
@WebServlet("/forward")
public class ForwardServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        request.setAttribute("message", "ข้อความจาก Servlet ส่งไป JSP");
        // ส่งต่อไป JSP ที่อยู่ใน /WEB-INF/view/message.jsp
        request.getRequestDispatcher("/WEB-INF/view/message.jsp").forward(request, response);
    }
}
```

message.jsp:

```
<html>
<body>
    <h2>ข้อความที่ได้รับ: ${message}</h2>
</body>
</html>
```

ตัวอย่าง 5: Servlet สร้างและอ่าน Cookie

```
@WebServlet("/cookie")
public class CookieServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        // สร้าง cookie ชื่อ user ชื่อ Somchai
        Cookie cookie = new Cookie("user", "Somchai");
        cookie.setMaxAge(24 * 60 * 60); // อายุ 1 วัน
        response.addCookie(cookie);

        // อ่าน cookie ทั้งหมด
        Cookie[] cookies = request.getCookies();
        response.setContentType("text/html;charset=UTF-8");
        response.getWriter().println("<h3>Cookie ทั้งหมด:</h3>");
        if (cookies != null) {
            for (Cookie c : cookies) {
```

```
                response.getWriter().println("Name: " + c.getName() + ", Value: " + c.getValue() +
"<br>");
            }
        } else {
            response.getWriter().println("ไม่พบ cookie ใดๆ");
        }
    }
}
```

สรุป

ตัวอย่าง	ฟีเจอร์ที่แสดง
1	Servlet พื้นฐาน แสดงข้อความ HTML
2	รับค่าพารามิเตอร์ผ่าน URL (GET)
3	Redirect ไป URL อื่น
4	Forward ไป JSP พร้อมส่งข้อมูล
5	สร้างและอ่าน Cookie

สรุป

บทนี้นำเสนอขั้นตอนการสร้าง **Java Servlet แรก** โดยเริ่มจากการติดตั้งเครื่องมือที่จำเป็น ได้แก่ **NetBeans IDE** ซึ่งเป็นเครื่องมือสำหรับพัฒนา Java ที่รองรับการสร้าง Web Application ได้อย่างสะดวก และ **GlassFish** หรือ **Tomcat** ซึ่งทำหน้าที่เป็น Servlet Container สำหรับรันแอปพลิเคชัน โดยผู้อ่านสามารถเลือกติดตั้งให้เหมาะสมกับความต้องการและสภาพแวดล้อมของตน

จากนั้นจะเข้าสู่การเขียน Servlet เบื้องต้นโดยใช้คลาส HttpServlet ซึ่งรองรับการรับคำขอและส่งผลลัพธ์ผ่านโปรโตคอล HTTP พร้อมอธิบายการกำหนดเส้นทาง (URL Mapping) ให้กับ Servlet เพื่อให้สามารถเข้าถึงผ่านเว็บเบราว์เซอร์ได้ ทั้งในรูปแบบเดิมผ่านไฟล์ web.xml และแบบใหม่ด้วย annotation @WebServlet ซึ่งช่วยให้การกำหนดค่าเป็นไปอย่างยืดหยุ่นและสะดวกมากขึ้น

บทที่ 3

การทำงานของ HTTP และ Servlet Lifecycle

(HTTP and Servlet Lifecycle)

เนื้อหา

- HTTP Request & Response (HttpServletRequest, HttpServletResponse)
- Servlet Lifecycle (init(), service(), destroy())
- การส่งพารามิเตอร์ผ่าน URL (request.getParameter())

การพัฒนาเว็บแอปพลิเคชันด้วย Java Servlet จำเป็นต้องมีความเข้าใจเชิงลึกเกี่ยวกับการทำงานของโปรโตคอล HTTP ซึ่งเป็นมาตรฐานพื้นฐานในการสื่อสารระหว่างฝั่งผู้ใช้ (Client) และฝั่งเซิร์ฟเวอร์ (Server) โดยเฉพาะอย่างยิ่ง กลไกการรับและส่งข้อมูลผ่านคำขอ (Request) และการตอบกลับ (Response) มีความสำคัญต่อการประมวลผลข้อมูลของผู้ใช้ เช่น การส่งค่าผ่านแบบฟอร์ม การคลิกลิงก์ หรือการร้องขอบริการจากเว็บแอปพลิเคชัน คลาส HttpServletRequest และ HttpServletResponse ใน Java EE จึงเป็นเครื่องมือหลักที่ใช้ในการจัดการกับข้อมูลดังกล่าว

ในบทนี้จะเริ่มต้นด้วยการศึกษารูปแบบและหน้าที่ของ HTTP Request และ HTTP Response ผ่านการใช้งาน HttpServletRequest สำหรับการดึงค่าพารามิเตอร์จากคำขอ และ HttpServletResponse สำหรับการจัดเตรียมข้อมูลตอบกลับไปยังเบราว์เซอร์ของผู้ใช้ ความเข้าใจในกลไกของวัตถุทั้งสองนี้จะเอื้อต่อการเขียนโปรแกรมที่สามารถจัดการข้อมูลระหว่างฝั่งผู้ใช้และฝั่งเซิร์ฟเวอร์ได้อย่างมีประสิทธิภาพ และสอดคล้องกับมาตรฐานของการพัฒนาเว็บแอปพลิเคชันแบบสมัยใหม่

จากนั้นบทนี้จะนำเสนอแนวคิดของ **วงจรชีวิตของ Servlet (Servlet Lifecycle)** ซึ่งเป็นโครงสร้างพื้นฐานที่ควบคุมลำดับการทำงานของ Servlet ภายใต้การจัดการของ Servlet Container โดยจะกล่าวถึงเมธอดหลัก ได้แก่ init() สำหรับการเตรียมความพร้อมของ Servlet เมื่อถูกโหลดเข้าสู่หน่วยความจำ service() สำหรับการจัดการคำขอที่เข้ามา และ destroy() สำหรับการคืนทรัพยากรเมื่อ Servlet ไม่ถูกใช้งานอีกต่อไป ความเข้าใจในวงจรชีวิตนี้มีความสำคัญอย่างยิ่งในการเขียนโปรแกรมให้สามารถจัดการทรัพยากรได้อย่างเหมาะสมและมีประสิทธิภาพสูงสุด

ท้ายที่สุด บทนี้จะกล่าวถึงการส่งข้อมูลจากฝั่งผู้ใช้มายังเซิร์ฟเวอร์ผ่าน URL โดยอาศัยพารามิเตอร์ในรูปแบบของ Query String พร้อมแสดงตัวอย่างการใช้งานเมธอด getParameter() สำหรับดึงค่าพารามิเตอร์จากคำขอ ซึ่งเป็นเทคนิคที่ใช้กันอย่างแพร่หลายในระบบที่รับข้อมูลจากผู้ใช้ เช่น ระบบเข้าสู่ระบบ การคำนวณผลลัพธ์ หรือระบบแสดงผลแบบไดนามิก ความสามารถในการ

ประยุกต์ใช้เมธอดดังกล่าวจะเป็นรากฐานสำคัญของการพัฒนาเว็บแอปพลิเคชันแบบโต้ตอบในระดับสูงต่อไป

การทำงานของ HTTP และ Servlet Lifecycle

---

1. HTTP Request & Response (HttpServletRequest, HttpServletResponse)

HTTP คืออะไร?

- HTTP (HyperText Transfer Protocol) คือโปรโตคอลที่ใช้สำหรับสื่อสารข้อมูลระหว่างเว็บเบราเซอร์ (client) และเว็บเซิร์ฟเวอร์ (server) เช่น การส่งคำขอ (Request) เพื่อขอข้อมูล หรือส่งข้อมูลจากฟอร์ม แล้วเซิร์ฟเวอร์จะตอบกลับ (Response) ด้วยข้อมูลที่ต้องการ เช่น หน้าเว็บ หรือข้อมูล JSON

ใน Java Servlet

- คลาส HttpServletRequest ใช้เพื่อเก็บข้อมูลคำขอ HTTP ที่ส่งมาจากผู้ใช้ เช่น ข้อมูลพารามิเตอร์, header, cookie ฯลฯ
- คลาส HttpServletResponse ใช้สำหรับจัดการกับการตอบกลับ HTTP เช่น กำหนด Content-Type, ส่งข้อมูลกลับ, ตั้งสถานะของ HTTP Response

ตัวอย่างการใช้ HttpServletRequest และ HttpServletResponse

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        // อ่านพารามิเตอร์ "name" ที่ส่งมาทาง URL เช่น ?name=Somchai
        String name = request.getParameter("name");
        if (name == null || name.isEmpty()) {
            name = "Guest";
        }

        // กำหนด Content-Type ของการตอบกลับเป็น text/html และ charset UTF-8
        response.setContentType("text/html;charset=UTF-8");

        // เขียนข้อมูลตอบกลับไปยัง client
        response.getWriter().println("<h1>สวัสดี " + name + "!</h1>");
    }
}
```

เมื่อเรียก

http://localhost:8080/YourApp/hello?name=Somchai

จะได้

<h1>สวัสดี Somchai!</h1>

---

2. Servlet Lifecycle (init(), service(), destroy())

Servlet มี **วงจรชีวิต (Lifecycle)** ที่ควรเข้าใจ เพื่อบริหารการทำงานอย่างถูกต้อง:

Method	คำอธิบาย
init()	เรียกครั้งเดียวตอน Servlet ถูกโหลดและสร้างขึ้นใน Servlet Container (เช่น Tomcat)
service()	เรียกทุกครั้งที่มี HTTP Request มายัง Servlet เพื่อประมวลผลคำขอและส่ง Response
doGet() / doPost()	โดยทั่วไป service() จะเรียกเมธอดตาม HTTP Method เช่น doGet(), doPost()
destroy()	เรียกครั้งเดียวก่อน Servlet ถูกทำลาย (เช่น ตอน Server ปิดหรือ Redeploy) เพื่อเคลียร์ resource

ภาพรวมวงจรชีวิต Servlet

1. **โหลดและสร้าง** Servlet (init())
2. **รับคำขอและประมวลผล** (service() -> doGet()/doPost())
3. **ถูกทำลาย** เมื่อไม่ใช้งาน (destroy())

ตัวอย่าง Servlet ที่แสดง Lifecycle

```
@WebServlet("/lifecycle")
public class LifecycleServlet extends HttpServlet {

    @Override
    public void init() throws ServletException {
        System.out.println("Servlet กำลังถูกสร้างและ init()");
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        System.out.println("Servlet กำลังประมวลผล doGet()");
        response.setContentType("text/plain");
```

```
        response.getWriter().println("Hello from doGet()");
    }

    @Override
    public void destroy() {
        System.out.println("Servlet กำลังถูกทำลาย destroy()");
    }
}
```

เมื่อ Deploy และเรียก URL /lifecycle จะเห็นข้อความใน Console ว่ามีการเรียก init() และ doGet() ตามลำดับ

---

3. การส่งพารามิเตอร์ผ่าน URL (request.getParameter())

วิธีส่งข้อมูลจาก Client ไปยัง Servlet ทำได้หลายแบบ โดยแบบหนึ่งที่ง่ายคือ **ส่งผ่าน URL Query String**

ตัวอย่าง URL:

http://localhost:8080/app/hello?name=Somchai&age=30

ใน Servlet เราสามารถดึงค่าพารามิเตอร์ได้ด้วย

```
String name = request.getParameter("name");  // "Somchai"
String age = request.getParameter("age");    // "30"
```

ข้อควรระวัง:

- พารามิเตอร์ทั้งหมดจะถูกส่งเป็น String
- หากพารามิเตอร์ไม่ถูกส่งมา จะได้ค่า null
- ต้องเช็คค่า null ก่อนใช้งานหรือแปลงประเภทข้อมูล (เช่น String เป็น int)

ตัวอย่างเต็ม: รับพารามิเตอร์จาก URL แล้วแสดงผล

```
@WebServlet("/userinfo")
public class UserInfoServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        String name = request.getParameter("name");
        String ageStr = request.getParameter("age");
        int age = 0;
        if (ageStr != null) {
            try {
```

```
                age = Integer.parseInt(ageStr);
            } catch (NumberFormatException e) {
                age = -1; // กำหนดค่าไม่ถูกต้อง
            }
        }

        response.setContentType("text/html;charset=UTF-8");
        response.getWriter().println("<h2>ข้อมูลผู้ใช้</h2>");
        response.getWriter().println("ชื่อ: " + (name != null ? name : "ไม่ระบุ") + "<br>");
        response.getWriter().println("อายุ: " + (age > 0 ? age : "ไม่ระบุหรือไม่ถูกต้อง") + "<br>");
    }
}
```

สรุป

หัวข้อ	คำอธิบายสั้น ๆ
HTTP Request & Response	ใช้ HttpServletRequest อ่านข้อมูล, HttpServletResponse ส่งข้อมูลกลับ
Servlet Lifecycle	มี 3 เมธอดหลัก: init() (เริ่มต้น), service() (ประมวลผลคำขอ), destroy() (ทำลาย)
การส่งพารามิเตอร์ผ่าน URL	ใช้ request.getParameter("name") เพื่อดึงค่าที่ส่งมาทาง URL

รายละเอียดเพิ่มเติมในแต่ละหัวข้อที่เกี่ยวกับ HTTP และ Servlet Lifecycle พร้อมเทคนิคและแนวทางใช้งานจริง

HTTP Request & Response (เพิ่มเติม)

HTTP Request Methods ที่พบบ่อยใน Servlet

- **GET**: ขอข้อมูลจากเซิร์ฟเวอร์ ส่งพารามิเตอร์ผ่าน URL (Query String) เช่น ?name=abc
- **POST**: ส่งข้อมูลจาก client ไปยัง server ผ่าน body ของ HTTP request (เหมาะกับข้อมูลขนาดใหญ่ หรือข้อมูลที่ต้องการความปลอดภัยกว่า)
- **PUT, DELETE, HEAD, OPTIONS**: ใช้ใน REST API และ HTTP ขั้นสูง

การดึงข้อมูล Request Header

เราสามารถดึงข้อมูลส่วน header ที่ client ส่งมาด้วยได้ เช่น

```
String userAgent = request.getHeader("User-Agent");
String referer = request.getHeader("Referer");
```

การตั้งค่า Response Header และ Status Code

```
response.setStatus(HttpServletResponse.SC_NOT_FOUND); // กำหนด HTTP Status 404
response.setHeader("Cache-Control", "no-cache");
```

Servlet Lifecycle (เพิ่มเติม)

การควบคุมการโหลด Servlet

ในไฟล์ web.xml สามารถกำหนดให้ Servlet โหลดทันทีตอน Server Startup ด้วย <load-on-startup>

```
<servlet>
    <servlet-name>MyServlet</servlet-name>
    <servlet-class>com.example.MyServlet</servlet-class>
    <load-on-startup>1</load-on-startup> <!-- โหลดตอนเริ่มเซิร์ฟเวอร์ -->
</servlet>
```

สรุปภาพวงจรชีวิตแบบละเอียด

ระยะ	อธิบาย
โหลดคลาส Servlet	Container โหลดคลาส Servlet ครั้งแรกเมื่อมีคำขอหรือถูกกำหนดให้โหลดตอน startup
init()	เรียกครั้งเดียว ใช้สำหรับตั้งค่าเริ่มต้น เช่น เปิด connection, resource ต่าง ๆ
service()	เรียกเมื่อมีคำขอ HTTP เข้ามา ทำหน้าที่ส่งต่อไปยัง doGet(), doPost(), ฯลฯ
doGet(), doPost()	แยกการทำงานตามประเภท HTTP request (GET, POST)
destroy()	เรียกเมื่อ Servlet ถูกทำลาย ใช้ปิด resource, connection ให้เรียบร้อย

3.3 การส่งพารามิเตอร์ผ่าน URL (เพิ่มเติม)

ส่งพารามิเตอร์แบบ POST (Form Data)

สำหรับการส่งข้อมูลผ่าน <form method="POST"> ตัว Servlet สามารถอ่านค่าพารามิเตอร์ได้เช่นเดียวกับ GET

```
<form action="/yourapp/submit" method="POST">
    ชื่อ: <input type="text" name="username">
    <input type="submit" value="ส่ง">
</form>
```

Servlet

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws IOException {
    String username = request.getParameter("username");
    response.getWriter().println("คุณส่งชื่อ: " + username);
}
```

ส่งหลายค่าพารามิเตอร์ชื่อเดียวกัน

กรณี <select multiple> หรือ checkbox ที่ชื่อเดียวกันหลายค่า เราใช้

```
String[] values = request.getParameterValues("checkboxName");
if(values != null) {
    for(String val : values) {
        response.getWriter().println(val + "<br>");
    }
}
```

เทคนิคที่ช่วยพัฒนา Servlet ให้ดีขึ้น

- **Encoding:** กำหนด character encoding ก่อนอ่านค่าพารามิเตอร์เสมอ เพื่อป้องกันปัญหาเรื่องภาษาไทยหรือ Unicode

```
request.setCharacterEncoding("UTF-8");
```

- **Handling null:** ตรวจสอบค่าที่ได้จาก getParameter() ว่าไม่ใช่ null ก่อนใช้งาน เพื่อป้องกัน NullPointerException
- **Response Buffering:** ตั้ง buffer ขนาดที่เหมาะสมเพื่อเพิ่มประสิทธิภาพการส่งข้อมูลกลับ client
- **Log Lifecycle Event:** การใช้ logger ใน init(), service(), destroy() จะช่วยตรวจสอบพฤติกรรมของ Servlet ในการพัฒนาและดีบัก

ตัวอย่าง Servlet ที่ใช้งานครบวงจร

```
@WebServlet("/process")
public class ProcessServlet extends HttpServlet {
    @Override
    public void init() {
        System.out.println("Servlet init()");
    }

    @Override
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        processRequest(request, response);
    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        processRequest(request, response);
    }

    private void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        request.setCharacterEncoding("UTF-8");
        String action = request.getParameter("action");
        response.setContentType("text/html;charset=UTF-8");

        if (action == null) {
            response.getWriter().println("ไม่พบคำสั่ง");
        } else if (action.equals("hello")) {
            response.getWriter().println("<h1>สวัสดี Servlet!</h1>");
        } else {
            response.getWriter().println("คำสั่งไม่รู้จัก: " + action);
        }
    }

    @Override
    public void destroy() {
        System.out.println("Servlet destroy()");
    }
}
```

HTTP Request & Response ใน Java Servlet

1. HTTP Request (HttpServletRequest)

HttpServletRequest คืออ็อบเจ็กต์ที่เก็บข้อมูลทั้งหมดของคำขอ (Request) ที่ Client ส่งมายัง Server ผ่านโปรโตคอล HTTP โดย Servlet จะใช้ HttpServletRequest เพื่อดึงข้อมูลต่าง ๆ เช่น

- URL ที่ถูกเรียก
- HTTP Method (GET, POST, PUT, DELETE ฯลฯ)
- พารามิเตอร์ที่ส่งมาจาก Client (เช่น ค่าจากฟอร์ม)
- ข้อมูล Header ต่าง ๆ (เช่น User-Agent, Cookie)
- ข้อมูล Session

---

*ตัวอย่างเมธอดหลักใน **HttpServletRequest***

เมธอด	คำอธิบาย	ตัวอย่างการใช้งาน
getMethod()	ดึง HTTP Method เช่น "GET", "POST"	request.getMethod();
getParameter(String)	ดึงค่าพารามิเตอร์ที่ส่งมา	request.getParameter("name");
getHeader(String)	ดึงข้อมูล Header จาก Request	request.getHeader("User-Agent");
getSession()	ดึง HttpSession ปัจจุบัน (หรือสร้างใหม่)	request.getSession();
getRequestURI()	ดึง URI ของคำขอ	request.getRequestURI();

---

2. HTTP Response (HttpServletResponse)

HttpServletResponse คืออ็อบเจ็กต์ที่ใช้ในการส่งข้อมูลตอบกลับ (Response) จาก Servlet ไปยัง Client ผ่าน HTTP โดย Servlet จะใช้ HttpServletResponse เพื่อ

- กำหนดสถานะของ HTTP Response (เช่น 200 OK, 404 Not Found)
- กำหนด Content Type ของข้อมูลที่ตอบกลับ (เช่น text/html, application/json)
- เขียนข้อมูลออกไปยัง Client (เช่น HTML, JSON)
- ตั้งค่า Header ของ HTTP Response (เช่น cookie, cache control)

---

*ตัวอย่างเมธอดหลักใน **HttpServletResponse***

เมธอด	คำอธิบาย	ตัวอย่างการใช้งาน
setContentType(String)	กำหนดประเภทเนื้อหาที่ตอบกลับ	response.setContentType("text/html");
setStatus(int)	กำหนดรหัส	response.setStatus(HttpServletResponse.SC_NOT_FOUND);

เมธอด	คำอธิบาย	ตัวอย่างการใช้งาน
	สถานะ HTTP เช่น 200, 404	
getWriter()	ดึง PrintWriter สำหรับเขียนข้อความ	response.getWriter().println("Hello World");
addHeader(String, String)	เพิ่ม Header ให้ Response	response.addHeader("Cache-Control", "no-cache");

ตัวอย่างการใช้งาน HttpServletRequest และ HttpServletResponse

นี่คือตัวอย่าง Servlet ง่าย ๆ ที่แสดงการรับค่าพารามิเตอร์จาก Request และตอบกลับ HTML ไปยัง Client

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/greet")
public class GreetingServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        // 1. อ่านค่าพารามิเตอร์ชื่อ "name" ที่ส่งมาจาก URL เช่น /greet?name=John
        String name = request.getParameter("name");
```

```
        if (name == null || name.isEmpty()) {
            name = "Guest";
        }

        // 2. กำหนดประเภทเนื้อหาที่ตอบกลับเป็น HTML
        response.setContentType("text/html;charset=UTF-8");

        // 3. เขียนข้อความตอบกลับไปยัง Client
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>Greeting Page</title></head>");
        out.println("<body>");
        out.println("<h1>สวัสดี, " + name + "!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

การทดสอบ

- เรียก URL: http://localhost:8080/yourapp/greet?name=John
- Servlet จะอ่านพารามิเตอร์ name จาก URL ("John")
- ตอบกลับหน้าจอ HTML ที่แสดงข้อความ **"สวัสดี, John!"**

ตัวอย่างการใช้งาน Request Header และ Response Header

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

    // อ่าน User-Agent Header จาก Client
    String userAgent = request.getHeader("User-Agent");

    // กำหนด Content Type
    response.setContentType("text/plain;charset=UTF-8");

    // เพิ่ม Custom Header ใน Response
```

```
    response.setHeader("X-Custom-Header", "MyHeaderValue");

    PrintWriter out = response.getWriter();
    out.println("User-Agent ที่ส่งมาคือ: " + userAgent);
    out.println("Header X-Custom-Header ถูกตั้งค่าแล้ว");
}
```

สรุป

ประเภท	ใช้ทำอะไร	ตัวอย่างการใช้งาน
HttpServletRequest	รับข้อมูลจาก Client เช่น พารามิเตอร์, Header, Session	getParameter(), getHeader(), getSession()
HttpServletResponse	ส่งข้อมูลตอบกลับ Client เช่น HTML, JSON, Header, Status	setContentType(), getWriter(), setStatus(), addHeader()

ตัวอย่าง Servlet เพิ่มเติมอีก 5 ตัวอย่าง ที่แสดงการใช้งาน HttpServletRequest และ HttpServletResponse ในรูปแบบต่าง ๆ

ตัวอย่างที่ 1: รับค่าจากฟอร์ม (POST Method) แล้วแสดงผล

```
@WebServlet("/submitForm")
public class FormServlet extends HttpServlet {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String username = request.getParameter("username");
        String email = request.getParameter("email");

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<html><body>");
        out.println("<h2>ข้อมูลที่ส่งมา</h2>");
        out.println("ชื่อผู้ใช้: " + username + "<br>");
        out.println("อีเมล: " + email);
```

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

ตัวอย่างที่ 2: ตั้งสถานะ HTTP Response (404 Not Found)

```
@WebServlet("/notFoundExample")
public class NotFoundServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setStatus(HttpServletResponse.SC_NOT_FOUND);
        response.setContentType("text/plain;charset=UTF-8");
        response.getWriter().println("404 - ไม่พบหน้าเว็บนี้");
    }
}
```

ตัวอย่างที่ 3: อ่านและแสดงข้อมูล Header จาก Request

```
@WebServlet("/showHeaders")
public class HeaderServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/plain;charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("User-Agent: " + request.getHeader("User-Agent"));
        out.println("Accept-Language: " + request.getHeader("Accept-Language"));
    }
}
```

ตัวอย่างที่ 4: ส่ง Redirect ไปยัง URL อื่น

```
@WebServlet("/redirectExample")
public class RedirectServlet extends HttpServlet {
    @Override
```

```
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        // Redirect ไปยัง Google
        response.sendRedirect("https://www.google.com");
    }
}
```

ตัวอย่างที่ 5: ตั้งค่า Cookie ใน Response

```
import jakarta.servlet.http.Cookie;

@WebServlet("/setCookie")
public class CookieServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws IOException {
        Cookie cookie = new Cookie("user", "ChatGPT");
        cookie.setMaxAge(60*60);  // เก็บ cookie ไว้ 1 ชั่วโมง
        response.addCookie(cookie);

        response.setContentType("text/plain;charset=UTF-8");
        response.getWriter().println("Cookie 'user' ถูกตั้งค่าแล้ว");
    }
}
```

Servlet Lifecycle คืออะไร?

Servlet Lifecycle คือชุดขั้นตอนที่ Servlet ผ่านตั้งแต่

1. ถูกสร้างและเริ่มต้น (init)
2. รับและตอบสนองคำขอ (service)
3. ถูกทำลายและทำความสะอาด (destroy)

โดย Servlet Container จะควบคุมและเรียกเมธอดเหล่านี้ให้ตามลำดับ

รายละเอียดเมธอดหลักใน Servlet Lifecycle

เมธอด	ความหมาย	การเรียกใช้โดย

เมธอด	ความหมาย	การเรียกใช้โดย
init()	เรียกครั้งเดียวตอนสร้าง Servlet เพื่อเตรียมการตั้งค่าเริ่มต้น	Servlet Container
service()	เรียกทุกครั้งเมื่อมี HTTP Request เข้ามา เพื่อประมวลผลและตอบสนอง	Servlet Container
destroy()	เรียกครั้งเดียวก่อน Servlet ถูกลบออกจากหน่วยความจำ เพื่อเคลียร์ resource	Servlet Container

---

การทำงานแต่ละเมธอด

1. init()

- เรียกครั้งแรกตอน Servlet ถูกโหลดเข้าหน่วยความจำ
- ใช้สำหรับเตรียมตัว เช่น โหลดข้อมูล config, สร้าง connection pool
- ถ้า override ต้องเรียก super.init() ด้วย

2. service(HttpServletRequest request, HttpServletResponse response)

- รับ Request ทุกตัวที่เข้ามา
- คอยตรวจสอบ HTTP method (GET, POST, PUT, DELETE)
- เรียกเมธอดย่อย เช่น doGet(), doPost() ตาม method ของ HTTP request

3. destroy()

- เรียกก่อน Servlet ถูกยกเลิก เช่น Server ปิด, Redeploy แอป
- ใช้สำหรับปิด resource ที่เปิดไว้ เช่น ปิด connection, ล้าง cache

---

ตัวอย่าง Servlet แสดง Lifecycle Logs

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/lifecycle")
public class LifecycleServlet extends HttpServlet {

    @Override
```

```
    public void init() throws ServletException {
        super.init();
        System.out.println("Servlet is initializing...");
    }

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("Service method called...");
        super.service(req, resp); // ให้เรียก doGet/doPost ต่อไป
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("Handling GET request...");
        resp.getWriter().println("Hello from doGet()");
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        System.out.println("Handling POST request...");
        resp.getWriter().println("Hello from doPost()");
    }

    @Override
    public void destroy() {
        System.out.println("Servlet is being destroyed...");
        super.destroy();
    }
}
```

การทดสอบ

1. เมื่อโหลด Servlet ตัวนี้ครั้งแรก (เช่น เปิด URL /lifecycle)
- Console log จะเห็นข้อความ "Servlet is initializing..." แสดงว่า init() ถูกเรียก
- หลังจากนั้น "Service method called..." ตามด้วย "Handling GET request..." (ถ้าใช้ GET)
- Browser จะเห็นข้อความ Hello from doGet()
2. เมื่อส่ง HTTP POST ไปที่ URL เดียวกัน
- Log จะเห็น "Service method called..." และ "Handling POST request..."
- Browser ได้ข้อความ Hello from doPost()
3. เมื่อ Server ปิดหรือ Redeploy
- Log จะเห็น "Servlet is being destroyed..." แสดงว่า destroy() ถูกเรียก

สรุป

- init() เรียกแค่ครั้งแรกตอน Servlet โหลด
- service() เรียกทุก Request เพื่อแยก HTTP Method ไปทำงานต่อ
- destroy() เรียกก่อน Servlet หยุดทำงาน เพื่อเคลียร์ resource

---

ตัวอย่าง Servlet ที่เน้นแสดงการทำงานของ **Servlet Lifecycle (init, service, destroy)** พร้อมฟีเจอร์ที่ต่างกัน 5 ตัวอย่าง เพื่อให้เห็นภาพและเข้าใจลึกขึ้น

ตัวอย่าง 1: Lifecycle กับการเก็บจำนวนผู้เข้าชม (Visitor Counter)

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/visitor")
public class VisitorCounterServlet extends HttpServlet {
    private int visitorCount;

    @Override
    public void init() throws ServletException {
        visitorCount = 0; // เริ่มนับ 0
        System.out.println("Servlet initialized: visitorCount set to 0");
```

```
    }

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        visitorCount++;
        System.out.println("Visitor count incremented: " + visitorCount);
        super.service(req, resp);
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        resp.setContentType("text/plain");
        resp.getWriter().println("Visitor Count: " + visitorCount);
    }

    @Override
    public void destroy() {
        System.out.println("Servlet destroyed. Final visitor count: " + visitorCount);
    }
}
```

ตัวอย่าง 2: Lifecycle กับ Resource Initialization (เช่น Database Connection)

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
@WebServlet("/dbtest")
public class DatabaseServlet extends HttpServlet {
    private Connection conn;

    @Override
    public void init() throws ServletException {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/mydb", "user",
"pass");
            System.out.println("Database connection established.");
        } catch (ClassNotFoundException | SQLException e) {
            throw new ServletException("DB connection failed", e);
        }
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        resp.setContentType("text/plain");
        if (conn != null) {
            resp.getWriter().println("Database connection is ready.");
        } else {
            resp.getWriter().println("No database connection.");
        }
    }

    @Override
    public void destroy() {
        try {
            if (conn != null && !conn.isClosed()) {
                conn.close();
                System.out.println("Database connection closed.");
            }
```

```
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

ตัวอย่าง 3: Lifecycle กับ Logging Request IP Address

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/logip")
public class LogIPServlet extends HttpServlet {

    @Override
    public void init() throws ServletException {
        System.out.println("LogIPServlet initialized");
    }

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        String ip = req.getRemoteAddr();
        System.out.println("Request from IP: " + ip);
        super.service(req, resp);
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        resp.getWriter().println("Your IP has been logged.");
```

```
    }

    @Override
    public void destroy() {
        System.out.println("LogIPServlet destroyed");
    }
}
```

ตัวอย่าง 4: Lifecycle กับนับจำนวน Request แยกตาม Method (GET, POST)

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/methodcount")
public class MethodCountServlet extends HttpServlet {
    private int getCount;
    private int postCount;

    @Override
    public void init() throws ServletException {
        getCount = 0;
        postCount = 0;
        System.out.println("MethodCountServlet initialized");
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        getCount++;
        resp.getWriter().println("GET requests: " + getCount);
    }
```

```
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        postCount++;
        resp.getWriter().println("POST requests: " + postCount);
    }

    @Override
    public void destroy() {
        System.out.println("MethodCountServlet destroyed");
        System.out.println("Total GET: " + getCount);
        System.out.println("Total POST: " + postCount);
    }
}
```

ตัวอย่าง 5: Lifecycle กับแสดงเวลาที่ Servlet ถูกสร้างและถูกทำลาย

```
import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.time.LocalDateTime;

@WebServlet("/timeservlet")
public class TimeServlet extends HttpServlet {
    private LocalDateTime initTime;

    @Override
    public void init() throws ServletException {
        initTime = LocalDateTime.now();
        System.out.println("Servlet initialized at: " + initTime);
    }
```

```
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        LocalDateTime now = LocalDateTime.now();
        resp.getWriter().println("Servlet was initialized at: " + initTime);
        resp.getWriter().println("Current time: " + now);
    }

    @Override
    public void destroy() {
        LocalDateTime destroyTime = LocalDateTime.now();
        System.out.println("Servlet destroyed at: " + destroyTime);
    }
}
```

ตัวอย่าง Servlet ที่มีการส่งข้อมูลกลับเป็น HTML พร้อมโค้ดเต็ม ๆ พร้อมอธิบายประกอบให้เข้าใจง่าย

ตัวอย่าง 1: VisitorCounterServlet แสดงจำนวนผู้เข้าชมในรูปแบบ HTML

```
package com.example.servlet;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/visitor")
public class VisitorCounterServlet extends HttpServlet {
    private int visitorCount;
```

```
    @Override
    public void init() throws ServletException {
        visitorCount = 0;
    }

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        visitorCount++;
        super.service(req, resp); // เรียก doGet หรือ doPost ตาม method
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        resp.setContentType("text/html;charset=UTF-8");

        try (PrintWriter out = resp.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html><head><title>Visitor Counter</title></head><body>");
            out.println("<h1>Welcome to Our Website!</h1>");
            out.println("<p>You are visitor number: <strong>" + visitorCount + "</strong></p>");
            out.println("</body></html>");
        }
    }
}
```

ตัวอย่าง 2: TimeServlet แสดงเวลาที่ Servlet เริ่มทำงานและเวลาปัจจุบันใน HTML

```
package com.example.servlet;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
```

```
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

@WebServlet("/time")
public class TimeServlet extends HttpServlet {
    private LocalDateTime initTime;

    @Override
    public void init() throws ServletException {
        initTime = LocalDateTime.now();
    }

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        resp.setContentType("text/html;charset=UTF-8");
        LocalDateTime now = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

        try (PrintWriter out = resp.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html><head><title>Time Servlet</title></head><body>");
            out.println("<h2>Servlet Initialized at: " + initTime.format(formatter) + "</h2>");
            out.println("<h2>Current Time: " + now.format(formatter) + "</h2>");
            out.println("</body></html>");
        }
    }
}
```

ตัวอย่าง 3: ParameterEchoServlet รับพารามิเตอร์จาก URL แล้วแสดงผลใน HTML

```
package com.example.servlet;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/echo")
public class ParameterEchoServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        String name = req.getParameter("name");
        if (name == null || name.isEmpty()) {
            name = "Guest";
        }

        resp.setContentType("text/html;charset=UTF-8");

        try (PrintWriter out = resp.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html><head><title>Parameter Echo</title></head><body>");
            out.println("<h1>Hello, " + escapeHtml(name) + "!</h1>");
            out.println("<p>Try passing your name in the URL like ?name=John</p>");
            out.println("</body></html>");
        }
    }

    // ฟังก์ชันช่วยป้องกัน XSS เบื้องต้น
```

```
    private String escapeHtml(String input) {
        return input.replaceAll("&", "&amp;")
                    .replaceAll("<", "&lt;")
                    .replaceAll(">", "&gt;")
                    .replaceAll("\"", "&quot;")
                    .replaceAll("'", "&#x27;");
    }
}
```

ตัวอย่าง 4: FormServlet แสดงฟอร์ม HTML และรับข้อมูล POST

```
package com.example.servlet;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;

@WebServlet("/form")
public class FormServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        resp.setContentType("text/html;charset=UTF-8");

        try (PrintWriter out = resp.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html><head><title>Input Form</title></head><body>");
            out.println("<h1>Submit Your Name</h1>");
            out.println("<form method='post' action='form'>");
```

```
            out.println("Name: <input type='text' name='username' />");
            out.println("<input type='submit' value='Submit' />");
            out.println("</form>");
            out.println("</body></html>");
        }
    }

    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        String username = req.getParameter("username");
        if (username == null || username.isEmpty()) {
            username = "Unknown";
        }

        resp.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = resp.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html><head><title>Form Result</title></head><body>");
            out.println("<h1>Thank you, " + escapeHtml(username) + "!</h1>");
            out.println("<a href='form'>Back to form</a>");
            out.println("</body></html>");
        }
    }

    private String escapeHtml(String input) {
        return input.replaceAll("&", "&amp;")
                    .replaceAll("<", "&lt;")
                    .replaceAll(">", "&gt;")
                    .replaceAll("\"", "&quot;")
                    .replaceAll("'", "&#x27;");
    }
}
```

ตัวอย่าง 5: HeaderServlet แสดง HTTP Headers ของคำขอในรูปแบบ HTML

```
package com.example.servlet;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

@WebServlet("/headers")
public class HeaderServlet extends HttpServlet {

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
IOException {
        resp.setContentType("text/html;charset=UTF-8");

        try (PrintWriter out = resp.getWriter()) {
            out.println("<!DOCTYPE html>");
            out.println("<html><head><title>Request Headers</title></head><body>");
            out.println("<h1>HTTP Request Headers</h1>");
            out.println("<table border='1'>");
            out.println("<tr><th>Header Name</th><th>Header Value</th></tr>");

            Enumeration<String> headerNames = req.getHeaderNames();
            while (headerNames.hasMoreElements()) {
                String name = headerNames.nextElement();
                String value = req.getHeader(name);
                out.println("<tr><td>" + escapeHtml(name) + "</td><td>" + escapeHtml(value) +
"</td></tr>");
```

```
            }

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

    private String escapeHtml(String input) {
        return input.replaceAll("&", "&amp;")
                    .replaceAll("<", "&lt;")
                    .replaceAll(">", "&gt;")
                    .replaceAll("\"", "&quot;")
                    .replaceAll("'", "&#x27;");
    }
}
```

วิธีทดสอบ

- Deploy Servlet เหล่านี้บน Tomcat หรือ GlassFish
- เข้า URL เช่น http://localhost:8080/yourApp/visitor หรือ /time หรือ /echo?name=ChatGPT เป็นต้น

การส่งพารามิเตอร์ผ่าน URL

การส่งพารามิเตอร์ผ่าน URL คืออะไร?

การส่งพารามิเตอร์ผ่าน URL คือการแนบข้อมูลที่เราต้องการส่งไปยัง Servlet ลงไปใน URL โดยใช้รูปแบบ

http://server/context-path/servlet-path?key1=value1&key2=value2

ข้อมูลหลังเครื่องหมาย ? คือพารามิเตอร์ ที่มีรูปแบบเป็นคู่ key=value และแต่ละคู่จะแยกกันด้วยเครื่องหมาย &

วิธีดึงพารามิเตอร์ใน Servlet ด้วย request.getParameter()

ในฝั่ง Servlet เราสามารถดึงค่าพารามิเตอร์ที่ส่งมาจาก URL ได้ด้วย

String value = request.getParameter("key");

โดย key คือชื่อพารามิเตอร์ที่ส่งมาใน URL

ตัวอย่างการส่งพารามิเตอร์ผ่าน URL

1. ตัวอย่าง HTML ส่งพารามิเตอร์ผ่านลิงก์

```
<!DOCTYPE html>
<html>
<head>
    <title>Send Parameter via URL</title>
</head>
<body>
    <h1>Send Parameter via URL Example</h1>
    <a href="hello?name=John">Say Hello to John</a>
</body>
</html>
```

เมื่อคลิกลิงก์นี้ Browser จะส่ง HTTP GET ไปที่ URL /hello?name=John

---

2. ตัวอย่าง Servlet รับพารามิเตอร์และแสดงผล

```
import java.io.IOException;
import java.io.PrintWriter;

import jakarta.servlet.ServletException;
import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

@WebServlet("/hello")
public class HelloServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        // ดึงพารามิเตอร์ชื่อ "name" จาก URL
        String name = request.getParameter("name");

        response.setContentType("text/html;charset=UTF-8");
```

```
        PrintWriter out = response.getWriter();

        out.println("<html><body>");
        if (name != null && !name.isEmpty()) {
            out.println("<h1>Hello, " + name + "!</h1>");
        } else {
            out.println("<h1>Hello, Guest!</h1>");
        }
        out.println("</body></html>");
    }
}
```

- Servlet นี้รับคำขอแบบ GET
- ดึงค่าพารามิเตอร์ชื่อ name จาก URL
- แสดงข้อความทักทายโดยใช้ค่าที่รับมา

---

3. ตัวอย่าง URL ที่เรียก Servlet

http://localhost:8080/YourApp/hello?name=Alice

ผลลัพธ์ที่แสดงในเว็บเบราว์เซอร์คือ

Hello, Alice!

---

4. ตัวอย่างการใช้ฟอร์ม HTML ส่งพารามิเตอร์ (GET method)

```
<!DOCTYPE html>
<html>
<head>
    <title>Send Parameter via Form</title>
</head>
<body>
    <h1>Send Parameter via Form</h1>
    <form action="hello" method="get">
        Enter your name: <input type="text" name="name" />
        <input type="submit" value="Say Hello" />
    </form>
</body>
</html>
```

- เมื่อกรอกชื่อแล้วกด Submit จะส่งค่าชื่อไปที่ /hello?name=xxx
- Servlet จะรับค่าพารามิเตอร์แล้วแสดงผลเหมือนตัวอย่างก่อนหน้า

สรุป

ขั้นตอน	รายละเอียด
ส่งพารามิเตอร์ผ่าน URL	แนบข้อมูลหลังเครื่องหมาย ? ใน URL
ดึงพารามิเตอร์ใน Servlet	ใช้ request.getParameter("key")
ตัวอย่างการใช้งาน	ส่งลิงก์ หรือ ฟอร์ม HTML ที่มีพารามิเตอร์

ตัวอย่างเพิ่มเติมของการส่งพารามิเตอร์ผ่าน URL และการใช้งาน request.getParameter() ในหลายรูปแบบ

ตัวอย่าง 1: ส่งพารามิเตอร์หลายตัวผ่าน URL

URL ตัวอย่าง

http://localhost:8080/YourApp/user?name=Mark&age=25&city=Bangkok

Servlet รับพารามิเตอร์หลายตัว

```
@WebServlet("/user")
public class UserServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String name = request.getParameter("name");
        String age = request.getParameter("age");
        String city = request.getParameter("city");

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<html><body>");
        out.println("<h2>User Info</h2>");
        out.println("<p>Name: " + name + "</p>");
        out.println("<p>Age: " + age + "</p>");
        out.println("<p>City: " + city + "</p>");
        out.println("</body></html>");
```

```
    }
}
```

ตัวอย่าง 2: ส่งพารามิเตอร์จากฟอร์ม HTML แบบ POST

HTML ฟอร์ม

```
<!DOCTYPE html>
<html>
<head>
    <title>POST Form Example</title>
</head>
<body>
    <form action="submit" method="post">
        Username: <input type="text" name="username" /><br/>
        Password: <input type="password" name="password" /><br/>
        <input type="submit" value="Login" />
    </form>
</body>
</html>
```

Servlet รับพารามิเตอร์ POST

```
@WebServlet("/submit")
public class SubmitServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<html><body>");
        out.println("<h2>Login Info</h2>");
        out.println("<p>Username: " + username + "</p>");
        out.println("<p>Password: " + password + "</p>");
        out.println("</body></html>");
```

```
    }
}
```

ตัวอย่าง 3: ตรวจสอบพารามิเตอร์ว่าเป็นค่าว่างหรือไม่

```
@WebServlet("/check")
public class CheckParamServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String param = request.getParameter("data");

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<html><body>");
        if (param == null || param.trim().isEmpty()) {
            out.println("<p>No data was provided.</p>");
        } else {
            out.println("<p>Received data: " + param + "</p>");
        }
        out.println("</body></html>");
    }
}
```

ตัวอย่าง 4: ส่งพารามิเตอร์ใน URL และส่งกลับเป็น JSON

```
@WebServlet("/json")
public class JsonServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String name = request.getParameter("name");
        if (name == null) name = "Guest";

        response.setContentType("application/json;charset=UTF-8");
        PrintWriter out = response.getWriter();
```

```
        String json = String.format("{\"message\": \"Hello, %s!\"}", name);
        out.print(json);
    }
}
```

ตัวอย่าง 5: ส่งพารามิเตอร์หลายค่าซ้ำกัน (Multiple Values)

บางครั้งเราส่งพารามิเตอร์ที่มีชื่อเดียวกันแต่มีค่าหลายค่า เช่น

http://localhost:8080/YourApp/colors?color=red&color=green&color=blue

ใน Servlet เราใช้

String[] colors = request.getParameterValues("color");

ตัวอย่าง Servlet

```
@WebServlet("/colors")
public class ColorsServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        String[] colors = request.getParameterValues("color");

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();

        out.println("<html><body>");
        if (colors != null) {
            out.println("<h3>Selected Colors:</h3><ul>");
            for (String color : colors) {
                out.println("<li>" + color + "</li>");
            }
            out.println("</ul>");
        } else {
            out.println("<p>No colors selected.</p>");
        }
        out.println("</body></html>");
    }
}
```

สรุป

การทำงานของเว็บแอปพลิเคชันที่พัฒนาด้วย Java Servlet พื้นฐานสำคัญอยู่ที่การสื่อสารผ่านโปรโตคอล HTTP โดยใช้คลาส HttpServletRequest และ HttpServletResponse ในการรับและตอบสนองคำขอจากผู้ใช้ HttpServletRequest ช่วยให้สามารถดึงข้อมูลที่ผู้ใช้ส่งเข้ามา เช่น พารามิเตอร์ในแบบฟอร์มหรือ URL ขณะที่ HttpServletResponse ทำหน้าที่จัดเตรียมข้อมูลตอบกลับกลับไปยังผู้ใช้ เช่น ข้อความ HTML หรือผลลัพธ์จากการประมวลผล การเข้าใจการใช้คลาสทั้งสองนี้อย่างถูกต้องจึงเป็นรากฐานของการพัฒนาแอปพลิเคชันที่ตอบสนองได้อย่างมีประสิทธิภาพ

นอกจากนี้ การเข้าใจวงจรชีวิตของ Servlet (Servlet Lifecycle) ซึ่งประกอบด้วยเมธอด init(), service() และ destroy() จะช่วยให้สามารถควบคุมการทำงานของ Servlet ตั้งแต่เริ่มต้นจนสิ้นสุดได้อย่างมีประสิทธิผล โดย init() ใช้สำหรับเตรียมความพร้อมก่อนเริ่มให้บริการ service() ทำหน้าที่ประมวลผลคำขอทุกครั้งที่มีผู้ใช้งาน และ destroy() จะถูกเรียกเมื่อ Servlet ถูกนำออกจากหน่วยความจำ ทั้งนี้ยังมีการส่งข้อมูลระหว่างผู้ใช้และแอปพลิเคชันผ่านพารามิเตอร์ URL ซึ่งสามารถเข้าถึงได้โดยใช้ request.getParameter() เพื่อรับข้อมูลและนำไปใช้ในการประมวลผลคำขอในฝั่งเซิร์ฟเวอร์อย่างเหมาะสม