JSP
WEB PROGRAMMING: PROFESSIONAL
(INTEGRATIVE-GENERATIVE AI EDITION)

Contents: JSP and Frameworks, Professional Testing and Debugging, Deployment and CI/CD Management JSP Modular and Clean Code Advanced Security Practices, Trends and Complementary Technologies in JSP and Java Web

Student Price Book Center

คำนำ

ในยุคที่เทคโนโลยีเว็บมีการเปลี่ยนแปลงอย่างรวดเร็วและไม่หยุดนิ่ง Java Server Pages (JSP) ยังคงเป็นหนึ่งในเทคโนโลยีที่ทรงพลังและได้รับความนิยมในหมู่นักพัฒนาเว็บแอปพลิเคชัน ด้วยความสามารถในการผสานการแสดงผล (View) กับโค้ดฝั่งเซิร์ฟเวอร์ (Backend) ได้อย่างยืดหยุ่น JSP ถูกนำมาใช้ในระบบองค์กรขนาดใหญ่ ระบบธุรกิจภายในองค์กร และระบบที่ต้องการความเสถียรในระยะยาว ด้วยเหตุนี้ หนังสือ *JSP Web Programming: Professional* จึงถูกจัดทำขึ้นเพื่อตอบโจทย์นักพัฒนาเว็บที่ต้องการยกระดับทักษะ JSP สู่ระดับมืออาชีพ โดยไม่เพียงแต่ครอบคลุมพื้นฐานเท่านั้น แต่ยังขยายขอบเขตความรู้ไปยังเทคโนโลยีที่ทันสมัยและการประยุกต์ใช้ในระบบจริง

แนวโน้มของ Java Web Application ยุคใหม่เน้นการทำงานแบบกระจายตัว การตอบสนองที่รวดเร็ว และการผสานเทคโนโลยีหลายรูปแบบ JSP จึงไม่สามารถทำงานโดดเดี่ยวได้อีกต่อไป แต่ต้องสามารถทำงานร่วมกับเทคโนโลยีเสริมอื่น ๆ ได้ เช่น Servlet 4.0+, RESTful APIs, WebSocket, Spring Framework, และ JavaScript Framework ฝั่ง Frontend อย่าง Angular, React และ Vue.js หนังสือเล่มนี้ได้นำเสนอแนวทางที่ครอบคลุมการใช้ JSP ร่วมกับเทคโนโลยีเหล่านี้อย่างเป็นระบบ พร้อมแนะนำการเขียนโค้ดที่สะอาด มีโครงสร้างที่ยืดหยุ่น และพร้อมขยายในระดับโปรดักชัน

ขณะเดียวกัน เราก็ไม่สามารถมองข้ามบริบทของ *ความเปลี่ยนแปลงใน Java EE* ที่กำลังก้าวเข้าสู่ *Jakarta EE* อย่างเป็นทางการ ซึ่งส่งผลต่อการวางแผนพัฒนาแอปพลิเคชันด้วย JSP อย่างหลีกเลี่ยงไม่ได้ หนังสือเล่มนี้ได้อธิบายถึงบริบทดังกล่าวอย่างละเอียด พร้อมชี้ให้เห็นว่าการย้ายจาก Java EE สู่ Jakarta EE นั้น ไม่ได้เป็นเพียงแค่การเปลี่ยนชื่อแพ็กเกจ แต่ยังส่งผลต่อโครงสร้าง โค้ด และการใช้งานไลบรารีหลักในระยะยาว การเข้าใจความเปลี่ยนแปลงนี้จึงเป็นหัวใจสำคัญของนักพัฒนา JSP สมัยใหม่

แม้ว่าจะมีเทคโนโลยีใหม่ ๆ ที่เข้ามาแทนที่ JSP โดยเฉพาะในด้าน View Layer เช่น **JSF (JavaServer Faces)**, **Thymeleaf** และ **FreeMarker** หนังสือเล่มนี้ได้เปรียบเทียบข้อดีข้อเสียของแต่ละเทคโนโลยีอย่างตรงไปตรงมา พร้อมแสดงตัวอย่างโค้ดที่แสดงให้เห็นภาพรวมและความแตกต่างเชิงโครงสร้างอย่างชัดเจน เพื่อให้ผู้อ่านสามารถเลือกเทคโนโลยีที่เหมาะสมกับบริบทของตนเองได้โดยมีข้อมูลที่รอบด้าน

นอกจากความสามารถเชิงเทคนิคของ JSP แล้ว หนังสือเล่มนี้ยังชี้ให้เห็นถึงทิศทางใหม่ของการพัฒนาเว็บด้วย *สถาปัตยกรรม Microservices* ซึ่งกำลังได้รับความนิยมอย่างสูงในองค์กรยุคใหม่ ด้วยการออกแบบระบบให้แยกเป็นบริการย่อย ๆ ที่สามารถพัฒนา ทดสอบ และดีพลอยแยกจากกันได้ ทำให้สามารถนำ JSP มาใช้ในส่วนของบริการเฉพาะ เช่น ระบบแสดงผล Frontend ของบาง Service หรือใช้ร่วมกับ API Gateway ที่เป็นตัวกลางควบคุมการเข้าถึงบริการต่าง ๆ

ในส่วนตัวอย่างแนวประยุกต์ หนังสือได้นำเสนอ *3 โปรแกรมระดับ Production* ซึ่งแสดงการใช้ JSP ร่วมกับส่วนประกอบที่จำเป็นต่อการพัฒนาระบบยุคใหม่ ได้แก่:

1. ระบบ Authentication และ Authorization อย่างปลอดภัย
2. การใช้ API Gateway เพื่อควบคุมเส้นทางของ Request ในระบบ

3. การผสานการทำงานกับระบบ Cache เพื่อลดภาระเซิร์ฟเวอร์และเพิ่มความเร็ว

แต่ละตัวอย่างมีโครงสร้างแบบแยกโมดูล พร้อมคำอธิบายอย่างเป็นขั้นตอน และสามารถนำไปประยุกต์ใช้ได้ทันทีในองค์กรจริง

ท้ายที่สุด หนังสือยังนำเสนอ *แนวทางการใช้งาน JSP บน Cloud Services* อย่างเช่น AWS, Google Cloud และ Microsoft Azure รวมถึงการตั้งค่าเซิร์ฟเวอร์, การดีพลอยแบบ Auto Scaling, การจัดการ Configuration ด้วย Secret Manager หรือ dotenv ทำให้ผู้อ่านเข้าใจแนวทางการพัฒนา JSP Web Application ที่พร้อมใช้งานในระบบ Cloud Native ได้อย่างมั่นใจ

ด้วยโครงสร้างที่ครอบคลุมทั้งภาคทฤษฎีและการปฏิบัติ หนังสือ *JSP Web Programming: Professional* เล่มนี้ จึงเหมาะสำหรับนักพัฒนา Java ที่ต้องการยกระดับทักษะ JSP ให้ทันสมัย พร้อมรับมือกับเทคโนโลยีในอนาคต และสามารถประยุกต์ใช้งานจริงในโลกองค์กรและระบบขนาดใหญ่ได้อย่างมั่นคง

ขอให้ผู้อ่านทุกท่านได้รับประโยชน์จากเนื้อหาในหนังสือเล่มนี้ และสามารถนำความรู้ไปต่อยอดการพัฒนาเว็บในระดับมืออาชีพได้อย่างมั่นใจและยั่งยืน

ด้วยรักและปรารถนาดี

ศูนย์หนังสือราคานักเรียน

สารบัญ

หน้า

บทที่ 15

การบูรณาการ JSP กับ Frameworks

(JSP and Frameworks)

เนื้อหา

- การบูรณาการ JSP กับ Frameworks
- รายละเอียดเชิงลึก การบูรณาการ JSP กับ Frameworks
- การใช้งาน JSP ร่วมกับ Spring MVC
- การใช้ JSP ในโปรเจกต์ที่ใช้ Hibernate/JPA สำหรับ ORM
- การสร้าง RESTful API และการเรียกใช้งานผ่าน JSP
- การใช้ JSP กับ Frontend สมัยใหม่ (React, Angular) แบบ Hybrid
- ตัวอย่างบูรณาการ

บทนำหนังสือ (สำหรับบทที่ 15: การบูรณาการ JSP กับ Frameworks)

ในยุคของการพัฒนาเว็บแอปพลิเคชันที่เปลี่ยนแปลงอย่างรวดเร็ว นักพัฒนาไม่เพียงต้องเข้าใจการทำงานของ JSP (JavaServer Pages) อย่างลึกซึ้งเท่านั้น แต่ยังต้องสามารถนำไปประยุกต์ใช้ร่วมกับเฟรมเวิร์กและเทคโนโลยีสมัยใหม่ได้อย่างมีประสิทธิภาพ บทที่ 15 นี้จึงมุ่งเน้นการบูรณาการ JSP เข้ากับเฟรมเวิร์กยอดนิยม เช่น Spring MVC, Hibernate/JPA, RESTful API และเทคโนโลยี frontend สมัยใหม่ เพื่อเสริมพลังให้กับการพัฒนาเว็บที่ครบวงจรและทันสมัย

หัวข้อแรกที่เราจะกล่าวถึงคือการใช้งาน JSP ร่วมกับ Spring MVC ซึ่งเป็นหนึ่งในเฟรมเวิร์กยอดนิยมของ Java โดยเฉพาะในส่วนของ View Layer การตั้งค่า View Resolver และการส่งข้อมูลผ่าน Model ถือเป็นพื้นฐานสำคัญที่นักพัฒนาต้องเข้าใจ หากสามารถเชื่อมโยง JSP เข้ากับ Spring MVC ได้อย่างถูกต้อง จะช่วยให้การแยกหน้าที่ของ Controller และ View มีความชัดเจนและยืดหยุ่นมากขึ้น

จากนั้นเราจะต่อยอดไปสู่การใช้ JSP ในโปรเจกต์ที่มีการจัดการฐานข้อมูลด้วย Hibernate หรือ JPA ซึ่งเป็นเครื่องมือ ORM (Object-Relational Mapping) ที่ทรงพลัง ในส่วนนี้ผู้อ่านจะได้เรียนรู้การสร้าง Model, การแมปข้อมูลจากฐานข้อมูล และการแสดงผลผ่านหน้า JSP อย่างเป็นระบบ ทำให้สามารถพัฒนาเว็บแอปที่มีการจัดเก็บข้อมูลได้อย่างมีประสิทธิภาพ

เมื่อพูดถึงระบบเว็บยุคใหม่ การสร้าง RESTful API ก็เป็นสิ่งจำเป็นอย่างหลีกเลี่ยงไม่ได้ ในบทนี้เราจะเจาะลึกถึงแนวทางการสร้าง RESTful API ด้วย Spring ร่วมกับการเรียกใช้งานผ่าน JSP ไม่ว่าจะเป็นการดึงข้อมูล JSON มาจัดแสดง หรือการส่งข้อมูลกลับไปยังเซิร์ฟเวอร์ ซึ่งช่วยให้ JSP มีความสามารถในการติดต่อกับ Backend แบบ API ได้อย่างยืดหยุ่น

นอกจากนี้เรายังไม่ลืมบทบาทของ frontend framework ที่กำลังได้รับความนิยมอย่างแพร่หลาย เช่น React และ Angular ในบทนี้เราจะนำเสนอแนวทางการผสาน JSP เข้ากับเทคโนโลยีเหล่านี้ในรูปแบบ Hybrid Application ที่เปิดโอกาสให้ JSP ทำงานร่วมกับ JavaScript Framework ได้ในลักษณะที่เกื้อหนุนกัน ทำให้สามารถนำจุดเด่นของทั้งสองฝั่งมาใช้ร่วมกันอย่างมีประสิทธิภาพ

บทนี้ไม่ได้มุ่งเน้นเพียงแค่ทฤษฎี แต่ยังนำเสนอกรณีศึกษาจริง ตัวอย่างโค้ดที่ใช้งานได้จริง และแนวทางการแก้ปัญหาในสถานการณ์ที่ซับซ้อน ผู้อ่านจะได้ฝึกฝนและประยุกต์ใช้ความรู้ที่ได้จากบทก่อนหน้า มาสู่การพัฒนาเว็บแอปแบบ full-stack ที่มีทั้ง backend และ frontend ครบถ้วน

สุดท้าย ผู้อ่านจะเข้าใจว่าแม้ JSP จะเป็นเทคโนโลยีที่อยู่มานาน แต่ก็ยังสามารถยืนหยัดอยู่ในโลกของการพัฒนาเว็บได้อย่างสง่างาม หากนำมาบูรณาการกับเครื่องมือและแนวคิดใหม่ ๆ อย่างถูกวิธี บทนี้จึงเป็นอีกก้าวสำคัญที่จะพาผู้อ่านจากการเป็นนักพัฒนา JSP ระดับพื้นฐาน ไปสู่การเป็นนักพัฒนาเว็บระดับมืออาชีพที่พร้อมเผชิญกับความท้าทายของเทคโนโลยีสมัยใหม่อย่างแท้จริง

การบูรณาการ JSP กับ Frameworks

---

1. การใช้งาน JSP ร่วมกับ Spring MVC (View Resolver, Model)

- **Spring MVC** เป็น Framework ยอดนิยมสำหรับสร้างเว็บแอปที่มีโครงสร้าง MVC (Model-View-Controller)
- JSP มักใช้เป็น View เพื่อแสดงผลข้อมูลใน Spring MVC
- **View Resolver** ทำหน้าที่แปลงชื่อ View เป็นไฟล์ JSP จริง ๆ เช่น "home" -> "/WEB-INF/views/home.jsp"
- Controller จะรับคำขอ, ประมวลผล, ใส่ข้อมูลใน **Model** แล้วส่งชื่อ View กลับ
- JSP จะดึงข้อมูลจาก Model ผ่าน Expression Language (EL) แสดงผลหน้าเว็บ

ตัวอย่างโค้ด Controller (Java):

```
@Controller
public class HomeController {
    @RequestMapping("/home")
    public String home(Model model) {
        model.addAttribute("message", "Welcome to Spring MVC with JSP!");
        return "home"; // แปลเป็น /WEB-INF/views/home.jsp โดย ViewResolver
    }
}
```

ตัวอย่าง JSP (home.jsp):

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

```
    <h1>${message}</h1>
</body>
</html>
```

---

2. การใช้ JSP ในโปรเจกต์ที่ใช้ Hibernate/JPA สำหรับ ORM

- Hibernate/JPA ช่วยให้การจัดการฐานข้อมูลแบบ Object-Relational Mapping (ORM) ง่ายขึ้น
- Controller หรือ Service จะเรียกใช้งาน Entity จากฐานข้อมูลผ่าน Hibernate/JPA
- ข้อมูลที่ได้จะถูกส่งต่อไปยัง JSP ผ่าน Model
- JSP จะทำหน้าที่แสดงผลข้อมูลโดยใช้ EL และ JSTL

ตัวอย่างการเรียกข้อมูลใน Controller:

```
@Autowired
private UserRepository userRepository;

@RequestMapping("/users")
public String listUsers(Model model) {
    List<User> users = userRepository.findAll();
    model.addAttribute("users", users);
    return "users"; // users.jsp
}
```

ตัวอย่าง JSP แสดงข้อมูล List:

```
<table border="1">
  <tr><th>ID</th><th>Name</th><th>Email</th></tr>
  <c:forEach var="user" items="${users}">
    <tr>
      <td>${user.id}</td>
      <td>${user.name}</td>
      <td>${user.email}</td>
    </tr>
  </c:forEach>
</table>
```

---

3. การสร้าง RESTful API และการเรียกใช้งานผ่าน JSP

- สร้าง REST API ด้วย Spring Boot หรือ Framework อื่นๆ เพื่อให้บริการข้อมูลในรูปแบบ JSON

- JSP สามารถเรียก API เหล่านี้ผ่าน JavaScript (AJAX) เพื่อดึงข้อมูลมาแสดงผลแบบไดนามิก
- วิธีนี้ช่วยแยก Backend กับ Frontend ออกจากกันอย่างชัดเจน (Hybrid Architecture)

ตัวอย่าง REST Controller:

```
@RestController
public class UserRestController {
    @GetMapping("/api/users")
    public List<User> getUsers() {
        return userRepository.findAll();
    }
}
```

ตัวอย่าง JSP + AJAX ดึงข้อมูล JSON:

```
<html>
<head>
<script>
function loadUsers() {
    fetch('/api/users')
        .then(response => response.json())
        .then(data => {
            let table = "<table border='1'><tr><th>ID</th><th>Name</th><th>Email</th></tr>";
            data.forEach(user => {
                table += `<tr><td>${user.id}</td><td>${user.name}</td><td>${user.email}</td></tr>`;
            });
            table += "</table>";
            document.getElementById("userTable").innerHTML = table;
        });
}
</script>
</head>
<body onload="loadUsers()">
  <h2>User List (Loaded via AJAX)</h2>
  <div id="userTable"></div>
</body>
</html>
```

4. การใช้ JSP กับเทคโนโลยี Frontend สมัยใหม่ (เช่น React, Angular) แบบ Hybrid

- ในโปรเจกต์ Hybrid บางครั้งใช้ JSP สำหรับจัดการ layout หลักและ session/authentication
- ส่วน Frontend สมัยใหม่ เช่น React หรือ Angular จะทำงานบน client-side และโหลดผ่าน JSP
- JSP อาจจะส่งค่าเริ่มต้น (เช่น token, user info) ผ่านการแทรกใน <script> tag หรือ meta tag
- React/Angular จะใช้ข้อมูลนี้เชื่อมต่อกับ REST API หรือ WebSocket เพื่อแสดงข้อมูลแบบ dynamic

ตัวอย่าง JSP ฝัง React App:

```
<html>
<head>
  <title>React with JSP</title>
</head>
<body>
  <div id="root"></div>

  <script>
    // ส่งข้อมูล session/user ไป React
    window.appConfig = {
      user: "${sessionScope.user}"
    };
  </script>

  <!-- โหลด React Bundle -->
  <script src="static/js/main.js"></script>
</body>
</html>
```

React app จะอ่าน window.appConfig.user เพื่อใช้งาน

รายละเอียดเชิงลึก การบูรณาการ JSP กับ Frameworks

---

1. การใช้งาน JSP ร่วมกับ Spring MVC (View Resolver, Model)

- **Spring MVC Architecture:**

  Spring MVC แยกส่วนการทำงานออกเป็น Controller (จัดการคำขอ), Model (เก็บข้อมูล),

View (แสดงผล)

JSP ถูกใช้เป็น View Component เพื่อแสดงข้อมูลที่ Controller เตรียมไว้

- **View Resolver:**

  Spring MVC ใช้ View Resolver เพื่อแปลงชื่อ View ที่ Controller ส่งกลับ (เช่น "home") เป็น path ของ JSP จริง เช่น /WEB-INF/views/home.jsp

  ทำให้ไม่ต้องระบุ path เต็มใน Controller

- **Model:**

  Controller จะส่งข้อมูลที่ต้องการแสดงให้ JSP ผ่าน Model (เช่น model.addAttribute("key", value))

  JSP สามารถเข้าถึงข้อมูลใน Model ด้วย Expression Language (EL) เช่น ${key}

- **ข้อดี:**
  - ช่วยแยกความรับผิดชอบชัดเจน
  - ลดการใช้ Scriptlet ใน JSP ทำให้โค้ดสะอาดและง่ายต่อการดูแล
  - รองรับการทดสอบง่ายขึ้น

---

2. การใช้ JSP ในโปรเจกต์ที่ใช้ Hibernate/JPA สำหรับ ORM

- **ORM คืออะไร:**

  ORM (Object-Relational Mapping) คือเทคนิคเชื่อมโยงข้อมูลในฐานข้อมูลกับวัตถุในโปรแกรม Hibernate และ JPA เป็น Framework สำหรับ ORM

- **บทบาทของ JSP:**

  JSP รับข้อมูลที่ได้จาก ORM ผ่าน Controller หรือ Service

  ข้อมูลมักอยู่ในรูปแบบ JavaBeans หรือ Collection เช่น List, Map

- **การใช้งานใน JSP:**

  JSP ใช้ EL และ JSTL เพื่อแสดงข้อมูล เช่น
  - <c:forEach> เพื่อวนลูปแสดงข้อมูลหลายรายการ
  - EL เช่น ${user.name}, ${user.email} เพื่อดึงค่าจาก JavaBeans

- **ข้อดี:**
  - ไม่ต้องเขียน SQL ตรงใน JSP
  - การแยกชั้นชัดเจนทำให้โค้ดดูแลง่าย
  - เพิ่มประสิทธิภาพและลดข้อผิดพลาดจาก SQL แบบดิบ

---

3. การสร้าง RESTful API และการเรียกใช้งานผ่าน JSP

- **RESTful API คืออะไร:**

  เป็นสถาปัตยกรรมสำหรับสร้าง Web Service ที่ใช้ HTTP Methods (GET, POST, PUT,

DELETE)

ส่งข้อมูลระหว่าง Client กับ Server โดยปกติเป็น JSON หรือ XML

- **การเรียก API ผ่าน JSP:**

  JSP แสดงผลหน้าเว็บเป็น HTML และ JavaScript

  JavaScript (เช่น AJAX หรือ Fetch API) ดึงข้อมูล JSON จาก REST API มาประมวลผลและแสดงผลแบบไดนามิกบนหน้าเว็บ

- **ประโยชน์:**
  - เพิ่มประสิทธิภาพการโหลดข้อมูล ไม่ต้องรีเฟรชหน้าเต็ม
  - รองรับการพัฒนา Frontend ที่แยกจาก Backend ได้ง่ายขึ้น
  - ทำให้ UI ตอบสนองเร็วขึ้น
- **ความท้าทาย:**
  - ต้องดูแล CORS (Cross-Origin Resource Sharing) หาก Frontend กับ Backend อยู่คนละโดเมน
  - ต้องจัดการ Authentication ให้เหมาะสม เช่น Token-based

---

4. การใช้ JSP กับเทคโนโลยี Frontend สมัยใหม่ (เช่น React, Angular) แบบ Hybrid

- **แนวคิด Hybrid:**

  ใช้ JSP จัดการส่วน Server-Side Rendering (เช่น Layout, Authentication)

  ส่วน Frontend ทำงานบน client-side ด้วย React/Angular

- **การส่งข้อมูลจาก JSP ไป Frontend:**

  JSP สามารถฝังข้อมูล (เช่น ข้อมูล session, token) ใน <script> หรือ <meta> tags เพื่อให้ React/Angular อ่านและใช้งานต่อได้

- **ข้อดี:**
  - รักษาความสามารถ SEO จาก Server-Side Rendering
  - ใช้ความสามารถ JSP จัดการ session หรือ user authentication ได้ง่าย
  - Frontend สมัยใหม่สามารถโฟกัสกับ UI/UX ได้เต็มที่
- **แนวทางปฏิบัติ:**
  - แยก static resources (JS, CSS) ออกจาก JSP เพื่อจัดการ bundle และ cache
  - ใช้ API แยกจาก JSP สำหรับ data communication
  - จัดการ route บน client-side กับ server-side อย่างเหมาะสม

การใช้งาน JSP ร่วมกับ Spring MVC

การใช้งาน JSP ร่วมกับ Spring MVC (View Resolver, Model)

1. แนวคิดพื้นฐานของ Spring MVC กับ JSP

- Spring MVC เป็น Framework สำหรับสร้างเว็บแอปที่แยกชั้น Model, View, Controller อย่างชัดเจน
- JSP จะทำหน้าที่เป็น **View** ในการแสดงผลข้อมูล
- **Controller** จะรับคำขอ (Request), ประมวลผลธุรกิจ และส่งข้อมูลให้ JSP ผ่าน **Model**
- Spring จะใช้ **View Resolver** เพื่อแปลงชื่อ View ที่ Controller คืนค่า ไปเป็นไฟล์ JSP ที่จะเรนเดอร์จริงๆ

---

2. โครงสร้างไฟล์และตำแหน่ง JSP

- JSP มักจะวางไว้ในโฟลเดอร์ปลอดภัย เช่น /WEB-INF/views/ เพื่อป้องกันการเข้าถึงโดยตรงผ่าน URL
- ตัวอย่างโครงสร้างโปรเจกต์:

/src/main/webapp/WEB-INF/views/home.jsp

/src/main/webapp/WEB-INF/views/error.jsp

---

3. การตั้งค่า View Resolver ใน Spring MVC

- Spring ต้องการกำหนด View Resolver ให้รู้ว่า JSP อยู่ที่ไหน และใช้ suffix อะไร เช่น .jsp

ตัวอย่างการตั้งค่าแบบ Java Config:

```
@Configuration
@EnableWebMvc
@ComponentScan(basePackages = "com.example")
public class WebConfig implements WebMvcConfigurer {

    @Bean
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/views/");
        resolver.setSuffix(".jsp");
        return resolver;
    }
}
```

- เมื่อ Controller คืนชื่อ View เช่น "home"
- Spring จะหาว่า JSP จริงคือ /WEB-INF/views/home.jsp

4. การส่งข้อมูลผ่าน Model จาก Controller

- Controller จะเพิ่มข้อมูลลงใน Model เพื่อส่งให้ JSP ใช้แสดงผล
- ตัวอย่าง Controller:

```
@Controller
public class HomeController {

    @GetMapping("/home")
    public String home(Model model) {
        model.addAttribute("message", "Hello from Spring MVC!");
        return "home";  // ชื่อ View: home.jsp
    }
}
```

5. การใช้ข้อมูลใน JSP ผ่าน Expression Language (EL)

- JSP สามารถเข้าถึงข้อมูลใน Model ผ่าน EL ได้ง่าย ๆ เช่น ${message}
- ตัวอย่าง home.jsp:

```
<html>
<head><title>Home Page</title></head>
<body>
    <h1>${message}</h1>
</body>
</html>
```

6. ข้อดีของการใช้ JSP กับ Spring MVC

- **ลดการใช้ Scriptlets:** JSP ใช้ EL แทนการเขียน Java โค้ดใน JSP ทำให้โค้ดอ่านง่าย และแยก View กับ Business Logic ชัดเจน
- **ความปลอดภัย:** JSP ที่อยู่ใน /WEB-INF ไม่สามารถเข้าถึงโดยตรงผ่าน URL
- **การจัดการง่าย:** ใช้ View Resolver ช่วยให้จัดการไฟล์ JSP ได้สะดวกและเป็นระบบ
- **รองรับ MVC เต็มรูปแบบ:** ช่วยให้โค้ดโปรแกรมแบ่งแยกหน้าที่ดี ง่ายต่อการดูแลและทดสอบ

สรุป

ส่วนประกอบ	บทบาทใน Spring MVC + JSP

ส่วนประกอบ	บทบาทใน Spring MVC + JSP
Controller	รับคำขอ, ประมวลผล, เตรียมข้อมูล (Model), คืนชื่อ View
Model	เก็บข้อมูลที่ส่งจาก Controller ไป JSP
View Resolver	แปลงชื่อ View เป็น path JSP จริง
JSP	แสดงผลข้อมูลโดยใช้ EL และ JSTL

นี่คือตัวอย่างโปรแกรม Spring MVC + JSP จำนวน 3 โปรแกรมพื้นฐาน และ 3 โปรแกรมแนวประยุกต์ พร้อมโครงสร้าง คำอธิบาย และผลการรัน

ตัวอย่างพื้นฐาน (Basic Examples)

---

1. Hello World with Spring MVC + JSP

โครงสร้างโปรเจกต์

/src/main/java/com/example/controller/HomeController.java

/src/main/webapp/WEB-INF/views/hello.jsp

/src/main/java/com/example/config/WebConfig.java

WebConfig.java

```
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
public class WebConfig {

    @Bean
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver resolver = new InternalResourceViewResolver();
        resolver.setPrefix("/WEB-INF/views/");
        resolver.setSuffix(".jsp");
```

```
        return resolver;
    }
}
```

HomeController.java

```
package com.example.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HomeController {

    @GetMapping("/hello")
    public String hello(Model model) {
        model.addAttribute("message", "Hello Spring MVC with JSP!");
        return "hello"; // จะไปที่ /WEB-INF/views/hello.jsp
    }
}
```

hello.jsp

```
<html>
<head><title>Hello Page</title></head>
<body>
<h1>${message}</h1>
</body>
</html>
```

ผลการรัน

เข้าผ่าน URL: http://localhost:8080/yourApp/hello

แสดงผล:

Hello Spring MVC with JSP!

---

2. Passing Parameters and Showing List in JSP

HomeController.java (เพิ่ม)

```
@GetMapping("/users")
public String users(Model model) {
    List<String> userList = Arrays.asList("Alice", "Bob", "Charlie");
    model.addAttribute("users", userList);
    return "users"; // users.jsp
}
```

users.jsp

```
<html>
<head><title>User List</title></head>
<body>
<h2>User List</h2>
<ul>
    <c:forEach var="user" items="${users}">
        <li>${user}</li>
    </c:forEach>
</ul>
</body>
</html>
```

หมายเหตุ: ต้อง import JSTL ใน JSP ด้วย

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

ผลการรัน

URL: http://localhost:8080/yourApp/users

แสดงรายการ:

- Alice
- Bob
- Charlie

---

3. Form Submission with JSP and Spring MVC

HomeController.java (เพิ่ม)

```
@GetMapping("/form")
public String form() {
    return "form"; // form.jsp
}
```

```
@PostMapping("/submit")
public String submit(@RequestParam("name") String name, Model model) {
    model.addAttribute("userName", name);
    return "result"; // result.jsp
}
```

form.jsp

```
<html>
<head><title>Input Form</title></head>
<body>
<form method="post" action="submit">
    Name: <input type="text" name="name" />
    <input type="submit" value="Submit" />
</form>
</body>
</html>
```

result.jsp

```
<html>
<head><title>Result</title></head>
<body>
<h2>Hello, ${userName}!</h2>
</body>
</html>
```

ผลการรัน

- เข้า http://localhost:8080/yourApp/form
- กรอกชื่อแล้วกด submit
- แสดงข้อความ “Hello, [ชื่อที่กรอก]!”

ตัวอย่างแนวประยุกต์ (Advanced Examples)

---

1. Using JavaBean in Model with Spring MVC + JSP

User.java (JavaBean)

```
package com.example.model;
```

```
public class User {
    private String name;
    private int age;

    // Constructor
    public User() {}

    public User(String name, int age) {
        this.name = name; this.age = age;
    }

    // Getter & Setter
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
}
```

HomeController.java (แก้ไข)

```
@GetMapping("/user")
public String user(Model model) {
    User user = new User("John Doe", 30);
    model.addAttribute("user", user);
    return "user"; // user.jsp
}
```

user.jsp

```
<html>
<head><title>User Info</title></head>
<body>
<h2>User Information</h2>
<p>Name: ${user.name}</p>
<p>Age: ${user.age}</p>
</body>
```

```
</html>
```

ผลการรัน

URL: http://localhost:8080/yourApp/user

แสดงข้อมูล User ที่ส่งมาจาก Controller ผ่าน JSP EL

---

2. Using Model with List of JavaBeans

HomeController.java (แก้ไข)

```
@GetMapping("/userlist")
public String userList(Model model) {
    List<User> users = Arrays.asList(
        new User("Alice", 25),
        new User("Bob", 28),
        new User("Charlie", 32)
    );
    model.addAttribute("users", users);
    return "userlist"; // userlist.jsp
}
```

userlist.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head><title>User List</title></head>
<body>
<h2>Users</h2>
<table border="1">
<tr><th>Name</th><th>Age</th></tr>
<c:forEach var="user" items="${users}">
    <tr>
        <td>${user.name}</td>
        <td>${user.age}</td>
    </tr>
</c:forEach>
</table>
</body>
```

```
</html>
```

ผลการรัน

URL: http://localhost:8080/yourApp/userlist

แสดงตารางข้อมูล User List

---

3. Form Binding with Spring MVC + JSP + ModelAttribute

HomeController.java (เพิ่ม)

```
@GetMapping("/register")
public String showForm(Model model) {
    model.addAttribute("user", new User());
    return "register"; // register.jsp
}

@PostMapping("/register")
public String processForm(@ModelAttribute("user") User user, Model model) {
    model.addAttribute("message", "Registered user: " + user.getName() + ", age " +
user.getAge());
    return "registerResult"; // registerResult.jsp
}
```

register.jsp

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<html>
<head><title>Register User</title></head>
<body>
<h2>Register</h2>
<form:form method="post" modelAttribute="user" action="register">
    Name: <form:input path="name" /><br/>
    Age: <form:input path="age" /><br/>
    <input type="submit" value="Register" />
</form:form>
</body>
</html>
```

registerResult.jsp

```
<html>
<head><title>Register Result</title></head>
<body>
<h2>${message}</h2>
</body>
</html>
```

ผลการรัน

- เข้า http://localhost:8080/yourApp/register
- กรอกชื่อและอายุ แล้วส่งฟอร์ม
- แสดงข้อความยืนยันการลงทะเบียน

การใช้ JSP ในโปรเจกต์ที่ใช้ Hibernate/JPA สำหรับ ORM

---

1. แนวคิดหลัก

- **Hibernate/JPA** คือเทคโนโลยี ORM (Object-Relational Mapping) ที่ช่วยให้ Java สามารถจัดการกับฐานข้อมูลในรูปแบบวัตถุ (Objects) แทนการเขียน SQL แบบดิบ ๆ
- ในสถาปัตยกรรม MVC ที่ใช้ Spring MVC + Hibernate/JPA
    - **Model** คือ Entity และ Data Access Layer (DAO หรือ Repository) ที่ติดต่อกับฐานข้อมูลผ่าน Hibernate/JPA
    - **Controller** จะรับคำขอ ประมวลผล และนำข้อมูลที่ได้ส่งต่อไปยัง JSP ผ่าน Model
    - **View (JSP)** จะรับข้อมูลผ่าน EL และ JSTL เพื่อแสดงผลบนเว็บ

---

2. โครงสร้างโปรเจกต์แบบย่อ

/src/main/java/com/example/model/User.java (Entity)

/src/main/java/com/example/repository/UserRepository.java (JPA Repository)

/src/main/java/com/example/controller/UserController.java (Controller)

/src/main/resources/application.properties (config DB)

/src/main/webapp/WEB-INF/views/users.jsp (JSP View)

---

3. ตัวอย่างไฟล์สำคัญ

3.1 Entity Class - User.java

```
package com.example.model;
```

```
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "users")
public class User {

    @Id
    private Long id;

    private String name;
    private String email;

    // Constructors
    public User() {}
    public User(Long id, String name, String email) {
        this.id = id; this.name = name; this.email = email;
    }

    // Getters and Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

3.2 Repository Interface - UserRepository.java

```
package com.example.repository;
```

```
import com.example.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    // JpaRepository มี method พื้นฐาน เช่น findAll(), findById(), save(), delete()
}
```

3.3 Controller - UserController.java

```
package com.example.controller;

import com.example.model.User;
import com.example.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import java.util.List;

@Controller
public class UserController {

    @Autowired
    private UserRepository userRepository;

    @GetMapping("/users")
    public String users(Model model) {
        List<User> users = userRepository.findAll();
        model.addAttribute("users", users);
        return "users";  // users.jsp
    }
```

```
}
```

3.4 JSP View - users.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head><title>User List</title></head>
<body>
<h2>User List from Database</h2>
<table border="1">
<tr><th>ID</th><th>Name</th><th>Email</th></tr>
<c:forEach var="user" items="${users}">
    <tr>
        <td>${user.id}</td>
        <td>${user.name}</td>
        <td>${user.email}</td>
    </tr>
</c:forEach>
</table>
</body>
</html>
```

---

4. ตัวอย่างไฟล์ Configuration (application.properties)

```
spring.datasource.url=jdbc:mysql://localhost:3306/yourdb
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

---

5. คำอธิบายโดยสังเขป

- เมื่อผู้ใช้เข้า URL /users
- Controller เรียก userRepository.findAll() เพื่อดึงข้อมูล User ทั้งหมดจากฐานข้อมูล
- ข้อมูลส่งเข้า Model ด้วย key ชื่อ users
- JSP ใช้ <c:forEach> ลูปข้อมูล User แล้วแสดงตาราง

- ข้อมูลแต่ละเรคอร์ดแสดงผ่าน EL เช่น ${user.name}

---

6. ผลการรัน (สมมติฐาน)

ID	Name	Email
1	Alice Smith	alice@example.com
2	Bob Johnson	bob@example.com
3	Charlie Lee	charlie@example.com

---

7. สรุปข้อดี

- แยกความรับผิดชอบชัดเจน: Entity ดูแลข้อมูล, Repository ดูแลฐานข้อมูล, Controller จัดการคำขอ, JSP แสดงผล
- ไม่ต้องเขียน SQL ใน JSP หรือ Controller
- ใช้ EL/JSTL แสดงผลข้อมูลแบบง่ายและปลอดภัย
- Hibernate/JPA ช่วยจัดการเชื่อมต่อ DB, Transaction, Mapping แบบอัตโนมัติ

นี่คือตัวอย่างโปรแกรมแบบเต็มไฟล์พร้อมโครงสร้าง คำอธิบายโค้ด และผลการรัน จำนวน 3 โปรแกรมพื้นฐานและ 3 โปรแกรมแนวประยุกต์ ที่ใช้ JSP ร่วมกับ Hibernate/JPA สำหรับ ORM

ตัวอย่างพื้นฐาน 3 โปรแกรม — JSP + Hibernate/JPA

ตัวอย่างที่ 1: แสดงรายการ User ทั้งหมดจากฐานข้อมูล

โครงสร้างไฟล์

src/main/java/com/example/model/User.java

src/main/java/com/example/repository/UserRepository.java

src/main/java/com/example/controller/UserController.java

src/main/resources/application.properties

src/main/webapp/WEB-INF/views/users.jsp

src/main/java/com/example/config/WebConfig.java

---

1. User.java (Entity)

```
package com.example.model;

import jakarta.persistence.Entity;
```

```
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@Entity
@Table(name = "users")
public class User {
    @Id
    private Long id;
    private String name;
    private String email;

    public User() {}
    public User(Long id, String name, String email) {
        this.id = id; this.name = name; this.email = email;
    }

    // getters and setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

---

2. UserRepository.java

```
package com.example.repository;

import com.example.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
```

```
    // JpaRepository มี method พื้นฐาน เช่น findAll(), save(), delete()
}
```

---

3. UserController.java

```
package com.example.controller;

import com.example.model.User;
import com.example.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

import java.util.List;

@Controller
public class UserController {

    @Autowired
    private UserRepository userRepository;

    @GetMapping("/users")
    public String getUsers(Model model) {
        List<User> users = userRepository.findAll();
        model.addAttribute("users", users);
        return "users"; // ไปที่ users.jsp
    }
}
```

---

4. users.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head><title>User List</title></head>
<body>
```

```
<h2>User List</h2>
<table border="1">
<tr><th>ID</th><th>Name</th><th>Email</th></tr>
<c:forEach var="user" items="${users}">
    <tr>
        <td>${user.id}</td>
        <td>${user.name}</td>
        <td>${user.email}</td>
    </tr>
</c:forEach>
</table>
</body>
</html>
```

---

5. application.properties (ตัวอย่างเชื่อมต่อ MySQL)

```
spring.datasource.url=jdbc:mysql://localhost:3306/yourdb
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

---

6. WebConfig.java (ตั้งค่า View Resolver)

```
package com.example.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration
@EnableWebMvc
public class WebConfig {
    @Bean
```

```
    public InternalResourceViewResolver viewResolver() {
        InternalResourceViewResolver vr = new InternalResourceViewResolver();
        vr.setPrefix("/WEB-INF/views/");
        vr.setSuffix(".jsp");
        return vr;
    }
}
```

ผลการรัน

เปิดเบราเซอร์ที่ http://localhost:8080/yourApp/users

แสดงตาราง User ทั้งหมดจากฐานข้อมูล MySQL

ตัวอย่างที่ 2: เพิ่มข้อมูล User ผ่าน JSP Form และบันทึกด้วย Hibernate/JPA

---

1. UserController.java (เพิ่ม)

```
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.ModelAttribute;

@GetMapping("/userForm")
public String userForm(Model model) {
    model.addAttribute("user", new User());
    return "userForm";  // userForm.jsp
}

@PostMapping("/saveUser")
public String saveUser(@ModelAttribute("user") User user) {
    userRepository.save(user);
    return "redirect:/users";
}
```

---

2. userForm.jsp

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<html>
<head><title>User Form</title></head>
```

```
<body>
<h2>Add New User</h2>
<form:form method="post" action="saveUser" modelAttribute="user">
    ID: <form:input path="id" /><br/>
    Name: <form:input path="name" /><br/>
    Email: <form:input path="email" /><br/>
    <input type="submit" value="Save" />
</form:form>
</body>
</html>
```

ผลการรัน

- เข้า http://localhost:8080/yourApp/userForm
- กรอกข้อมูลและกด Save
- ระบบบันทึกข้อมูลและ redirect ไปหน้า /users เพื่อแสดงข้อมูลทั้งหมด

ตัวอย่างที่ 3: ลบข้อมูล User จาก JSP

---

1. UserController.java (เพิ่ม)

```
import org.springframework.web.bind.annotation.PathVariable;

@GetMapping("/deleteUser/{id}")
public String deleteUser(@PathVariable("id") Long id) {
    userRepository.deleteById(id);
    return "redirect:/users";
}
```

---

2. users.jsp (เพิ่มลิงก์ลบ)

```
<c:forEach var="user" items="${users}">
    <tr>
        <td>${user.id}</td>
        <td>${user.name}</td>
        <td>${user.email}</td>
        <td><a href="deleteUser/${user.id}">Delete</a></td>
```

```
    </tr>
</c:forEach>
```

ผลการรัน

- ที่หน้า /users จะมีลิงก์ Delete ในแต่ละแถว
- คลิกลิงก์เพื่อลบ User ตาม id และรีเฟรชหน้าแสดงข้อมูลล่าสุด

ตัวอย่างแนวประยุกต์ 3 โปรแกรม — ใช้ JSP + Hibernate/JPA + Spring MVC

ตัวอย่างที่ 1: Pagination แสดงผล User ทีละหน้า

---

1. UserController.java (แก้ไข)

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.web.bind.annotation.RequestParam;

@GetMapping("/usersPage")
public String getUsersPage(Model model,
        @RequestParam(defaultValue = "0") int page,
        @RequestParam(defaultValue = "5") int size) {

    Pageable pageable = PageRequest.of(page, size);
    Page<User> usersPage = userRepository.findAll(pageable);
    model.addAttribute("usersPage", usersPage);
    model.addAttribute("currentPage", page);
    return "usersPage"; // usersPage.jsp
}
```

---

2. usersPage.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head><title>User List Pagination</title></head>
<body>
```

```
<h2>Users (Page ${currentPage + 1})</h2>
<table border="1">
<tr><th>ID</th><th>Name</th><th>Email</th></tr>
<c:forEach var="user" items="${usersPage.content}">
    <tr>
        <td>${user.id}</td>
        <td>${user.name}</td>
        <td>${user.email}</td>
    </tr>
</c:forEach>
</table>

<c:if test="${usersPage.hasPrevious()}">
    <a href="usersPage?page=${currentPage - 1}">Previous</a>
</c:if>

<c:if test="${usersPage.hasNext()}">
    <a href="usersPage?page=${currentPage + 1}">Next</a>
</c:if>

</body>
</html>
```

ผลการรัน

- แสดงผู้ใช้ทีละหน้า 5 คน
- มีลิงก์ Next / Previous เพื่อเลื่อนดูหน้าต่อไป

ตัวอย่างที่ 2: ค้นหาข้อมูล User ด้วยชื่อ

---

1. UserRepository.java (เพิ่ม Method)

```
import java.util.List;

List<User> findByNameContainingIgnoreCase(String name);
```

---

2. UserController.java (เพิ่ม)

```
@GetMapping("/search")
public String searchUsers(@RequestParam("keyword") String keyword, Model model) {
    List<User> users = userRepository.findByNameContainingIgnoreCase(keyword);
    model.addAttribute("users", users);
    return "users"; // ใช้ users.jsp แสดงผล
}
```

---

3. searchForm.jsp

```
<html>
<head><title>Search Users</title></head>
<body>
<h2>Search Users</h2>
<form action="search" method="get">
    Name: <input type="text" name="keyword" />
    <input type="submit" value="Search" />
</form>
</body>
</html>
```

ผลการรัน

- กรอกชื่อบางส่วนในแบบฟอร์ม
- ระบบค้นหาและแสดง User ที่ชื่อใกล้เคียงในหน้า users.jsp

ตัวอย่างที่ 3: อัพเดตข้อมูล User

---

1. UserController.java (เพิ่ม)

```
@GetMapping("/editUser/{id}")
public String editUser(@PathVariable("id") Long id, Model model) {
    User user = userRepository.findById(id).orElse(null);
    model.addAttribute("user", user);
    return "editUser";  // editUser.jsp
}
```

```
@PostMapping("/updateUser")
public String updateUser(@ModelAttribute("user") User user) {
    userRepository.save(user);
    return "redirect:/users";
}
```

---

2. editUser.jsp

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>
<html>
<head><title>Edit User</title></head>
<body>
<h2>Edit User</h2>
<form:form method="post" modelAttribute="user" action="updateUser">
    ID: <form:input path="id" readonly="true" /><br/>
    Name: <form:input path="name" /><br/>
    Email: <form:input path="email" /><br/>
    <input type="submit" value="Update" />
</form:form>
</body>
</html>
```

ผลการรัน

- เปิด http://localhost:8080/yourApp/editUser/{id}
- แก้ไขข้อมูล User แล้วส่งฟอร์ม
- ระบบอัพเดตข้อมูลในฐานข้อมูลและ redirect ไปหน้าแสดงข้อมูลทั้งหมด

การสร้าง RESTful API และการเรียกใช้งานผ่าน JSP

---

1. แนวคิดหลัก

- **RESTful API** คือบริการ HTTP ที่ส่งข้อมูล JSON/XML เพื่อให้ Client (เช่น Frontend หรือ JSP) เรียกใช้ได้
- ใน Spring Boot/Spring MVC เราจะสร้าง Controller แบบ REST ที่ส่งข้อมูล JSON
- ฝั่ง JSP จะใช้ AJAX (เช่น jQuery, Fetch API) เรียก REST API แล้วนำข้อมูลมาแสดงผลโดยไม่ต้องรีเฟรชหน้า

- แยกแยะระหว่าง View (JSP) กับ Backend (API) ช่วยให้สถาปัตยกรรมโปรเจกต์ยืดหยุ่น

2. ตัวอย่างโปรเจกต์แบบเต็มไฟล์ — สร้าง REST API และเรียกจาก JSP

โครงสร้างโปรเจกต์

src/main/java/com/example/controller/RestUserController.java (REST API)

/src/main/java/com/example/controller/UserJspController.java (JSP Controller)

/src/main/java/com/example/model/User.java (Entity)

/src/main/resources/application.properties (DB config)

/src/main/webapp/WEB-INF/views/userList.jsp (JSP View)

3. โค้ดตัวอย่าง

3.1 Entity: User.java

```
package com.example.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

@Entity
public class User {
    @Id
    private Long id;
    private String name;
    private String email;

    public User() {}

    public User(Long id, String name, String email) {
        this.id = id; this.name = name; this.email = email;
    }

    // getters/setters omitted for brevity
    // ...
```

```
}
```

3.2 REST Controller: RestUserController.java

```
package com.example.controller;

import com.example.model.User;
import com.example.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/users")
public class RestUserController {

    @Autowired
    private UserRepository userRepository;

    // GET /api/users - ดึง User ทั้งหมด
    @GetMapping
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    // POST /api/users - เพิ่ม User ใหม่
    @PostMapping
    public User createUser(@RequestBody User user) {
        return userRepository.save(user);
    }
}
```

3.3 JSP Controller: UserJspController.java

```
package com.example.controller;
```

```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class UserJspController {

    @GetMapping("/userList")
    public String userListPage() {
        return "userList";  // userList.jsp
    }
}
```

3.4 JSP View: userList.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>User List with REST API</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
<h2>User List from REST API</h2>
<table border="1" id="userTable">
    <thead>
    <tr><th>ID</th><th>Name</th><th>Email</th></tr>
    </thead>
    <tbody>
    <!-- ข้อมูลจะถูกเพิ่มด้วย AJAX -->
    </tbody>
</table>

<script>
    $(document).ready(function () {
        $.ajax({
```

```
            url: '/api/users',
            method: 'GET',
            success: function (users) {
                var tbody = $('#userTable tbody');
                users.forEach(function (user) {
                    tbody.append('<tr><td>' + user.id + '</td><td>' + user.name + '</td><td>' +
user.email + '</td></tr>');
                });
            },
            error: function () {
                alert('Error fetching user data');
            }
        });
    });
</script>
</body>
</html>
```

---

4. คำอธิบายโค้ด

- **RestUserController**
  - สร้าง API /api/users ที่รองรับ GET (ดึงข้อมูล User ทั้งหมด) และ POST (เพิ่ม User ใหม่)
  - ส่งข้อมูลกลับในรูปแบบ JSON อัตโนมัติด้วย @RestController
- **UserJspController**
  - ให้ JSP แสดงหน้า userList.jsp
- **userList.jsp**
  - โหลด jQuery
  - ใช้ AJAX เรียก API /api/users ด้วย GET
  - เมื่อได้ข้อมูล User ที่เป็น JSON, วนลูปเพิ่ม <tr> เข้า <tbody> ของตาราง

---

5. ผลการรัน

- เข้า URL: http://localhost:8080/yourApp/userList
- หน้าเว็บแสดงตาราง User จากฐานข้อมูลที่โหลดผ่าน REST API แบบ AJAX
- หน้าไม่รีเฟรชข้อมูล แต่โหลดข้อมูลแบบไดนามิกจาก API

ตัวอย่างแนวประยุกต์

ตัวอย่าง 1: เพิ่ม User ผ่านฟอร์มใน JSP แล้วเรียก REST API ด้วย AJAX

- เพิ่ม <form> ใน JSP
- เมื่อ submit ใช้ AJAX POST ส่งข้อมูล JSON ไปยัง REST API /api/users
- เมื่อเพิ่มสำเร็จ โหลดข้อมูล User ใหม่จาก API มาแสดงในตาราง

ตัวอย่าง 2: ลบ User ด้วย REST API และ AJAX

- สร้าง REST API DELETE /api/users/{id}
- ใน JSP เพิ่มปุ่ม Delete ในแต่ละแถว
- ใช้ AJAX เรียก DELETE API เมื่อลบสำเร็จโหลดข้อมูลใหม่

ตัวอย่าง 3: อัพเดตข้อมูล User ด้วย REST API และ AJAX พร้อมฟอร์ม Modal

- สร้าง REST API PUT /api/users/{id}
- ใน JSP เพิ่มปุ่ม Edit แต่ละแถว
- เปิด Modal ฟอร์มแก้ไขข้อมูล
- Submit ฟอร์มด้วย AJAX PUT
- โหลดข้อมูลใหม่ในตารางหลังอัพเดต

นี่คือตัวอย่างโปรแกรมแบบเต็มไฟล์ + โครงสร้าง + คำอธิบายโค้ด + ผลการรัน จำนวน 3 โปรแกรมพื้นฐาน และ 3 โปรแกรมแนวประยุกต์ ที่สร้าง RESTful API ด้วย Spring Boot/JPA และเรียกใช้งานผ่าน JSP ด้วย AJAX

ตัวอย่างพื้นฐาน 3 โปรแกรม — RESTful API + JSP + AJAX

ตัวอย่างที่ 1: แสดงรายการ User ผ่าน REST API และ AJAX ใน JSP

โครงสร้างไฟล์

src/main/java/com/example/model/User.java

src/main/java/com/example/repository/UserRepository.java

src/main/java/com/example/controller/RestUserController.java

src/main/java/com/example/controller/UserJspController.java

src/main/resources/application.properties

src/main/webapp/WEB-INF/views/userList.jsp

---

1. User.java

```
package com.example.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;

@Entity
public class User {
    @Id
    private Long id;
    private String name;
    private String email;

    public User() {}

    public User(Long id, String name, String email) {
        this.id = id; this.name = name; this.email = email;
    }

    // getters and setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
}
```

---

2. UserRepository.java

```
package com.example.repository;

import com.example.model.User;
```

```
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {}
```

3. RestUserController.java

```
package com.example.controller;

import com.example.model.User;
import com.example.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/users")
public class RestUserController {

    @Autowired
    private UserRepository userRepository;

    @GetMapping
    public List<User> getAllUsers() {
        return userRepository.findAll();
    }
}
```

4. UserJspController.java

```
package com.example.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
```

```
public class UserJspController {

    @GetMapping("/userList")
    public String userListPage() {
        return "userList"; // userList.jsp
    }
}
```

---

5. userList.jsp

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>User List via REST API</title>
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
</head>
<body>
<h2>User List</h2>
<table border="1" id="userTable">
    <thead>
    <tr><th>ID</th><th>Name</th><th>Email</th></tr>
    </thead>
    <tbody>
    <!-- AJAX จะเติมข้อมูลตรงนี้ -->
    </tbody>
</table>

<script>
    $(function() {
        $.ajax({
            url: '/api/users',
            method: 'GET',
            success: function(users) {
                var tbody = $('#userTable tbody');
                users.forEach(function(user) {
```

```
                tbody.append('<tr><td>' + user.id + '</td><td>' + user.name + '</td><td>' +
user.email + '</td></tr>');
            });
        },
        error: function() {
            alert('Error loading user data');
        }
    });
});
</script>
</body>
</html>
```

---

6. application.properties (ตัวอย่าง)

```
spring.datasource.url=jdbc:mysql://localhost:3306/yourdb
spring.datasource.username=root
spring.datasource.password=yourpassword
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

ผลการรัน

- เข้า URL /userList
- หน้า JSP โหลดข้อมูล User ผ่าน AJAX จาก REST API และแสดงตาราง User

---

ตัวอย่างที่ 2: เพิ่ม User ผ่าน REST API ด้วย AJAX จาก JSP Form

---

1. RestUserController.java (เพิ่ม POST)

```
@PostMapping
public User addUser(@RequestBody User user) {
    return userRepository.save(user);
}
```

---

2. userAdd.jsp

```
<html>
```