



บทที่ 11

Spring MVC

ธีระยุทธ ทองเครือ

ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์

มหาวิทยาลัยขอนแก่น





Spring MVC



- ❖ Spring MVC คือ Framework ที่ใช้ในการสร้างเว็บแอปพลิเคชัน โดยมีการแยกส่วนระหว่างชั้น model และชั้น view เพื่อให้ UI programmers และ back end developers สามารถทำงานไปพร้อม ๆ กันได้
- ❖ Model Layer คือ ส่วนที่ประมวลผล business logic
- ❖ View Layer คือ ส่วนที่ render หน้า interface โดยใช้เทคโนโลยีการแสดงผลต่างๆ เช่น JSP, Facelets, FreeMarker หรือ Thymeleaf
- ❖ Controller ใ้รับ input จากผู้ใช้และส่งไปประมวลที่ Repository แล้วเก็บผลลัพธ์ไว้ที่ model object เพื่อลำเลียงไปยังส่วน View Layer
- ❖ การนำ Spring Boot มาใช้ร่วมกับ Spring MVC จะช่วยลดการ Config เพราะมีบางส่วนที่ทำงานแบบอัตโนมัติ ทำให้ลดเวลาในการพัฒนาลงได้

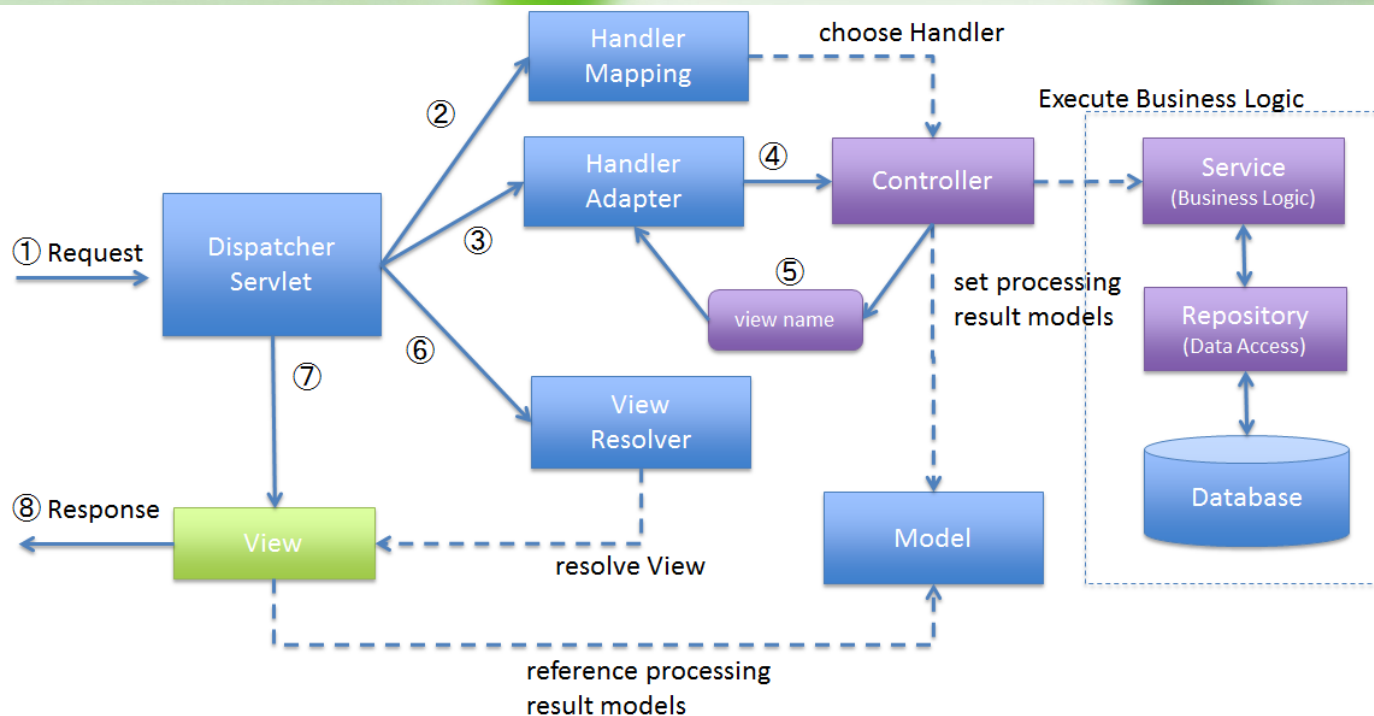


Dependency Injection



- ❖ Dependency Injection เรียกอีกชื่อหนึ่งว่า **IOC (Inversion of Control Principle)**
- ❖ เป็นวิธีการออกแบบซอฟต์แวร์ตาม pattern ที่มีการแยก component ต่าง ๆ ให้มีความเกี่ยวข้องกันน้อยที่สุด (loosely coupled)
- ❖ Component แต่ละส่วนสามารถนำมาประกอบกัน และสามารถทดสอบได้อย่างง่ายดาย
- ❖ ช่วยให้ reuse code ไปใช้ใน application อื่น ๆ ได้ง่าย
- ❖ ไม่มีการ Hard code หรือกำหนดค่าภายในคลาสต่างๆ โดยตรง แต่จะแยกไฟล์สำหรับ Config ในรูปแบบ XML configuration files หรือ จาวาคลาสสำหรับ Config ค่าโดยเฉพาะ
- ❖ Spring Container คือ ผู้ที่ทำหน้าที่ในการฉีด (Injecting) หรือสร้าง object เข้าไปยัง object ที่มีความเกี่ยวข้องกัน (dependencies of objects) เพื่อให้เกิดการทำงานร่วมกันในแอปพลิเคชัน
- ❖ Dependency Injection แบ่งออกเป็น 2 ประเภท ได้แก่ Setter Injection และ Constructor Injection

การประมวลผลคำร้องใน Spring MVC



สร้างโดยนักพัฒนา

Spring สร้างให้แล้ว

บางส่วน Spring สร้างให้แล้ว บางส่วนต้อง implement เพิ่ม

1. DispatcherServlet รับ request

2. DispatcherServlet ส่ง path ไปยัง HandlerMapping เพื่อเลือก Controller ที่มีอยู่

3. DispatcherServlet นำ Controller ที่เลือกแล้วส่งไปยัง HandlerAdapter

4. HandlerAdapter เรียกเมธอดที่นักพัฒนาได้สร้างไว้ในคลาส Controller ซึ่งอาจจะมีการประมวลทางธุรกิจ หรือเข้าถึงฐานข้อมูลโดยตรง หรือผ่าน Service

5. Controller เก็บผลลัพธ์การประมวลผลลงใน Model และระบุชื่อ View ส่งกลับไปยัง HandlerAdapter

6. DispatcherServlet เรียก ViewResolver เพื่อเลือก View ตามที่นักพัฒนากำหนด

7. DispatcherServlet นำหน้า view ที่นักพัฒนาสร้างไว้ Render เป็นผลลัพธ์

8. ส่ง response กลับไปยังผู้ใช้



รูปแบบคลาสที่เป็น Controller



ระบุ annotation เพื่อให้ Spring Boot Container รู้ว่าคลาสสามารถรับ request ได้

@Controller

@RequestMapping("/ชื่อ path ของ controller")

public class XxxxxController {

อาจไม่ใส่ annotation นี้ก็ได้ หากไม่ระบุ path ของแต่ละเมธอดจะไม่ต้องใส่

@GetMapping("/ชื่อ path เข้าถึงเมธอด")

public String ชื่อเมธอดใดๆก็ได้(@RequestParam("param1") String p1, ..., Model model) {

ดึงข้อมูลจาก request parameter มาเก็บในตัวแปร จะมีกี่ตัวก็ได้

เมธอด return ค่าเป็น String

กำหนด Model เป็น parameter ทุกครั้ง เมื่อต้องการ
บรรจุข้อมูลสำหรับนำไปใช้แสดงผลบน view

// เรียกเมธอดต่างๆ จาก Repository

// หรือประมวลผล Business Process

model.addAttribute("ชื่อข้อมูลที่จะส่งไปยัง view", ตัวแปรที่จะส่งไปยัง view);

model.addAttribute("...", ...);

model.addAttribute("...", ...);

เก็บ object ต่างๆ ลงใน model
ที่ต้องการให้ Template Engine
นำไปแสดงผล

return "ชื่อ html ที่ใช้ render ส่วน view โดยไม่ต้องใส่นามสกุล";

หรือ "redirect:/ชื่อ path ที่จะส่งต่อ";

}



การกำหนดรูปแบบการเข้าถึง Controller



- ❖ **@RequestMapping** ใช้กำหนด path ในการรับ request แบบใดๆ เช่น

```
@RequestMapping("/customer")
```

```
@RequestMapping(value="/addcustomer", method=RequestMethod.POST)
```

- ❖ **@GetMapping** ใช้กำหนด path ที่รับ request แบบ GET เช่น

```
@GetMapping("/haha")
```

- ❖ **@PostMapping** ใช้กำหนด path ที่รับ request แบบ POST เช่น

```
@PostMapping("/checkLogin")
```

- ❖ **@PutMapping** ใช้กำหนด path ที่รับ request แบบ Put มักใช้ใน Web API

```
@PutMapping("/updateUser")
```

- ❖ **@DeleteMapping** ใช้กำหนด path ที่รับ request แบบ Delete มักใช้ใน Web API

```
@DeleteMapping("/deleteUser")
```



การดึงข้อมูลที่ส่งมาจาก HTTP Request



❖ Controller สามารถรับข้อมูลจาก request ที่ client ส่งมาได้จาก Annotation ดังนี้

- **@RequestParam** ใช้รับค่าจาก URL Parameter เช่น

`http://www.aaa.com/addData?name=jim&age=20`

```
@GetMapping("/addData")
public String add(@RequestParam("name") String name, @RequestParam("age") int age, Model model) {
    ...
}
```

- **@PathVariable** ใช้รับค่าจาก Path เช่น

`http://www.aaa.com/customer/10/jim`

```
@GetMapping("/customer/{id}/{cname}")
public String check(@PathVariable("id") String cusid, @PathVariable("cname") String name, Model model) {
    ...
}
```



การดึงข้อมูลจาก HTTP Request ลง Entity Object

- ❖ Controller สามารถรับ request และเก็บลง Entity Object ได้เลย โดย client จะต้องส่งข้อมูลมายัง controller ในชื่อเดียวกับ attribute ใน Entity Object ส่วนเมธอดใน Controller จะมีพารามิเตอร์เป็น Entity class และระบุ (@ModelAttribute

Request URL:

http://localhost:8080/editnote?nid=2&task=Sleep

ค่าจะถูกเก็บลง object note อัตโนมัติ เพราะส่งข้อมูลที่มีชื่อเดียวกับ attribute ของ object

```
@GetMapping("/editnote")
public String edit(@ModelAttribute Note note, Model model) {
    noteRepo.save(note);
    return "redirect:/note";
}
```

Controller

```
@Entity
public class Note implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer nid;
    private String task;
    // Getter and Setter method
}
```




การดึงข้อมูลจาก HTTP Request ลง Entity Object

ตัวอย่าง กรณีส่งข้อมูลในแบบ Path

Request URL:

http://localhost:8080/editnote/2/Sleep

ค่าจะถูกเก็บลง object note อัตโนมัติ หากกำหนด pattern ของ path ที่มีชื่อใน { } ตรงกัน

```
@GetMapping("/editnote/{nid}/{task}")
public String edit(@ModelAttribute Note note, Model model) {
    noteRepo.save(note);
    return "redirect:/note";
}
```

Controller

```
@Entity
public class Note implements Serializable {
    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    private Integer nid;
    private String task;
    // Getter and Setter method
}
```



ระบบแนะนำเครื่องดื่ม



Select beer characteristics

Color: dark Submit

- light
- amber
- brown
- dark

ฟอร์ม HTML ที่จะ
ส่งไปยัง server

localhost:8080/BeerSelector?color=amber

Beer Recommendation

Singha
Asahi

ผลลัพธ์ที่ได้จากการ Render



การประมวลผลคำร้องใน Spring MVC

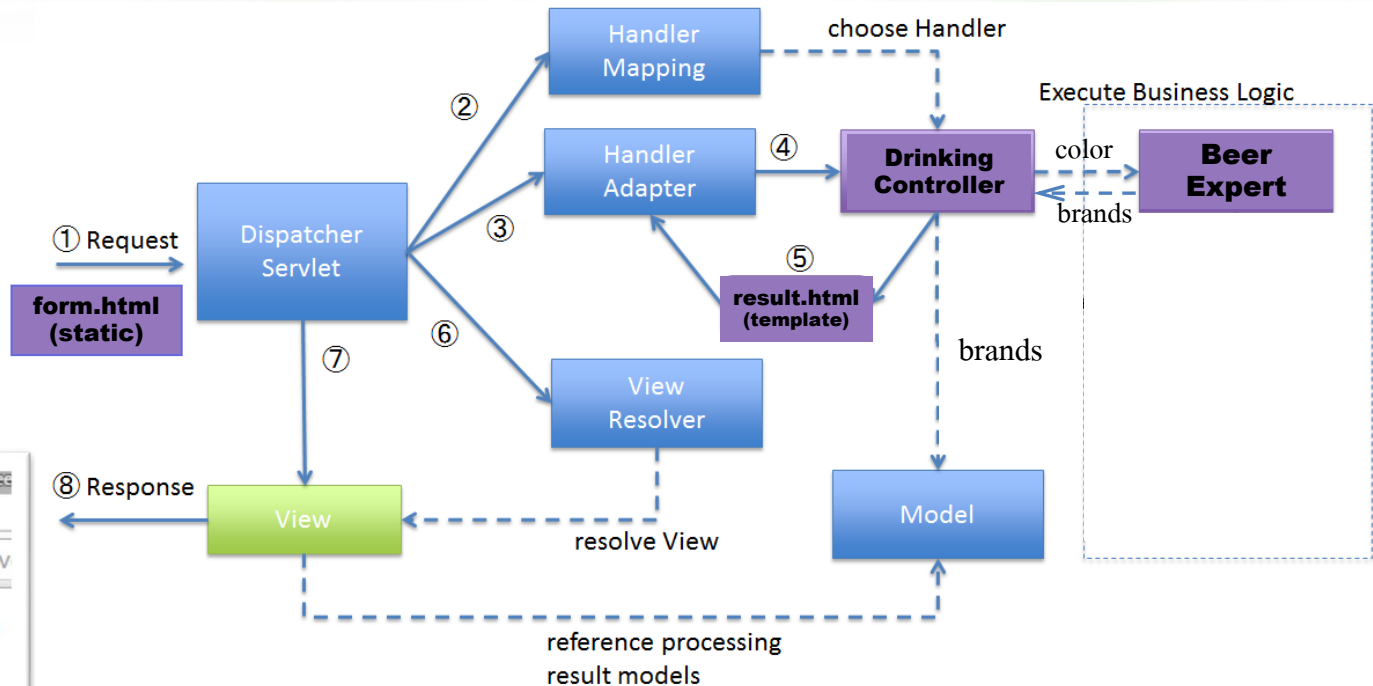


localhost:8080/mvc/form.html

Select beer characteristics

Color: Submit

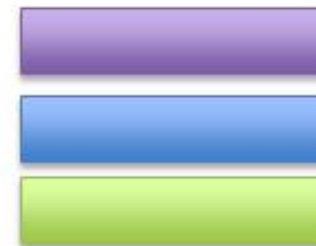
light
amber
brown
dark



localhost:8080/mvc/BeerS.html

Beer Recommendation

Singha
Asahi



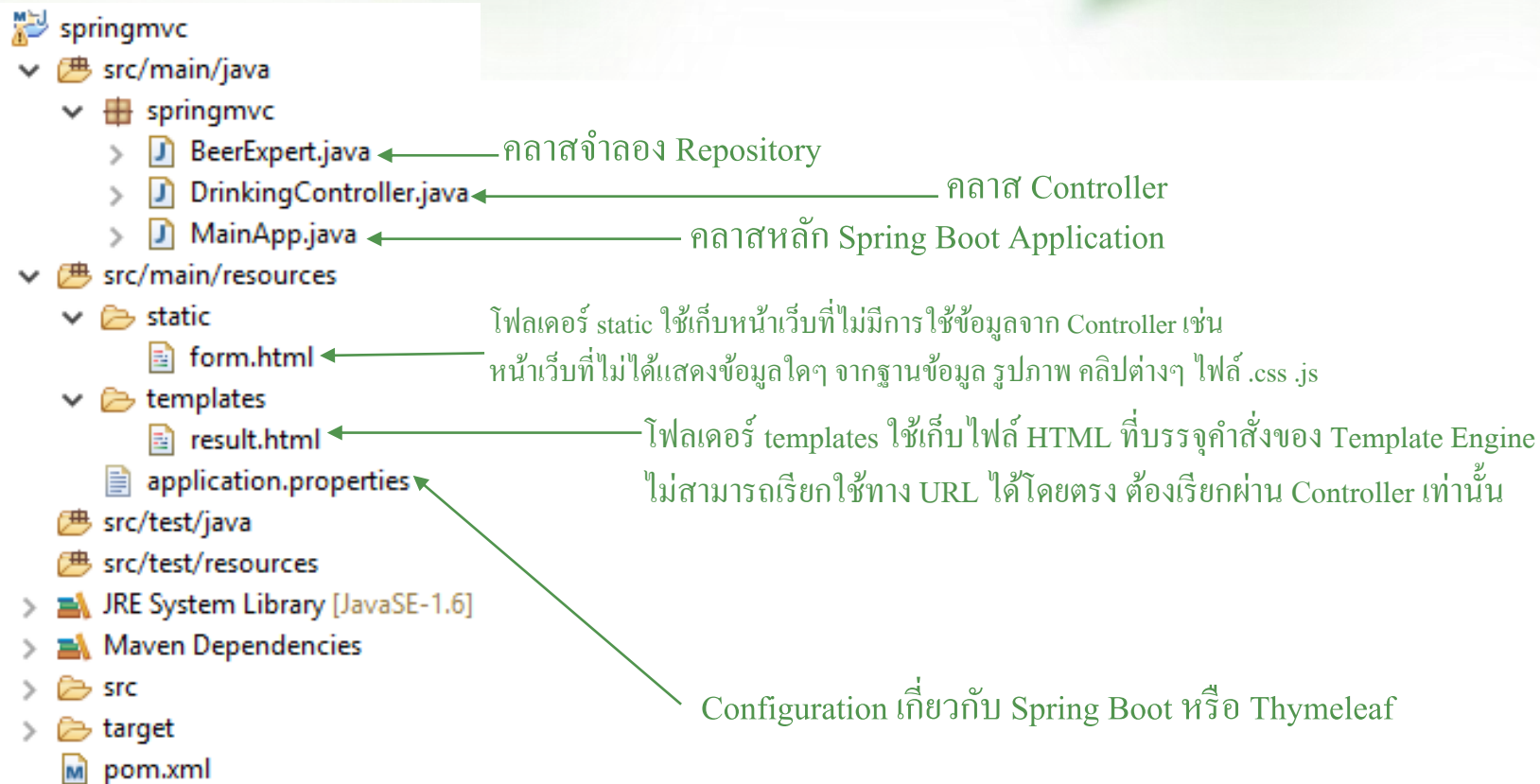
สร้างโดยนักพัฒนา

Spring สร้างให้แล้ว

บางส่วน Spring สร้างให้แล้ว บางส่วนต้อง implement เพิ่ม



โครงสร้างของ Project





pom.xml

❖ สร้าง Maven Project ใหม่ และเปิดไฟล์ pom.xml เพิ่ม Library ดังนี้

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.2.RELEASE</version>
</parent>
```

สืบทอดค่าเริ่มต้นของ Spring Boot

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
```

ชุดเริ่มต้นสำหรับสร้างเว็บ ซึ่งมี Spring MVC ด้วย

```
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
```

ใช้ Thymeleaf สำหรับ render ส่วน View

```
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
  </dependency>
</dependencies>
```

Developer Tools ช่วย reload แบบอัตโนมัติ เมื่อ
โค้ดมีการเปลี่ยนแปลง



ฟอร์มสำหรับส่งข้อมูล (ส่ง request)



ชื่อ Path ของ Controller ที่ใช้รับค่า

```
<html>
<body>
  <form action="BeerSelector">
    Select beer charateristics<br>
    Color:
    <select name="color">
      <option value="light">light</option>
      <option value="amber">amber</option>
      <option value="brown">brown</option>
      <option value="dark">dark</option>
    </select>
    <input type="submit">
  </form>
</body>
</html>
```



ส่วน Repository

```
import java.util.ArrayList;
```

```
public class BeerExpert {
```

```
    public ArrayList<String> getBrands(String color) {
```

```
        ArrayList<String> brands = new ArrayList<String>();
```

```
        if (color.equals("amber")) {
```

```
            brands.add("Singha");
```

```
            brands.add("Asahi");
```

```
        } else {
```

```
            brands.add("Carlsberg");
```

```
            brands.add("Heineken");
```

```
            brands.add("Tiger");
```

```
        }
```

```
        return brands;
```

```
    }
```

```
}
```

สร้าง object สำหรับเก็บ
ผลลัพธ์

ตรวจสอบเงื่อนไขเพื่อ
เก็บข้อมูลลงใน object

เพิ่มข้อมูล String ลงใน
ArrayList object

ส่ง ArrayList object กลับ



ส่วน Controller

@Controller

public class DrinkingController {

@GetMapping("/BeerSelector")

public String select(@RequestParam("color") String color, Model model) {

BeerExpert beerExpert = **new** BeerExpert();

ArrayList<String> brands = beerExpert.getBrands(color);

model.addAttribute("brands", brands);

return "result";

}

}

ดึงข้อมูลจาก request parameter

ชื่อ color มาเก็บในตัวแปร color

สร้าง BeerExpert object

และส่งสิ่งที่ผู้ใช้เลือกเพื่อขอข้อมูล

นำผลลัพธ์เก็บไว้ใน model object พร้อมกับ
ระบุชื่อ เพื่อใช้อ้างอิงในส่วน view

ชื่อ template ที่ใช้ render เป็นผลลัพธ์

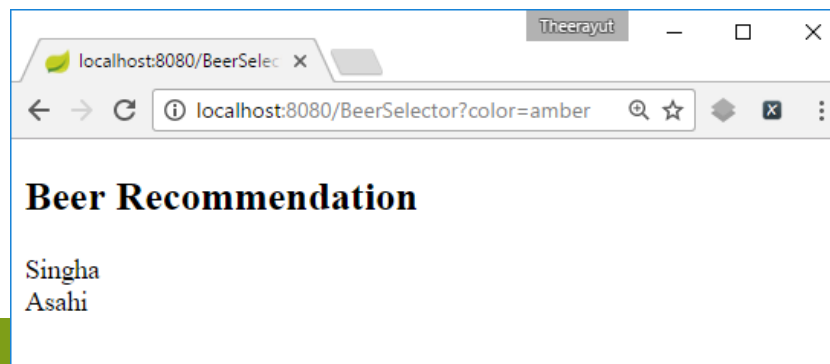


ส่วน View (result.html)

```
<!DOCTYPE html>
<html xmlns:th="http://thymeleaf.org">
  <head><meta charset="UTF-8"/></head>
  <body>
    <h2>Beer Recommendation</h2>
    <div th:each="beer: ${brands}">
      <span th:text="${beer}"></span>
    </div>
  </body>
</html>
```

ดึงผลลัพธ์จาก request object และวนลูป
ดึงสมาชิกแต่ละตัวมาเก็บไว้ที่ beer

นำ String จากสมาชิกแต่ละตัวแสดงผล





ภาพรวมโค้ด



C

```
form.html
1 <html>
2 <body>
3   <form action="/BeerSelector">
4     Select beer characteristics<br>
5     Color:
6     <select name="color">
7       <option value="light">light</option>
8       <option value="amber">amber</option>
9       <option value="brown">brown</option>
10      <option value="dark">dark</option>
11    </select>
12    <input type="submit">
13  </form>
14 </body>
15 </html>
16
```

```
*DrinkingController.java
9
10 @Controller
11 public class DrinkingController {
12
13   @GetMapping(value = "/BeerSelector")
14   public String select(@RequestParam("color") String color, Model model) {
15
16     BeerExpert beerExpert = new BeerExpert();
17     ArrayList<String> brands = beerExpert.getBrands(color);
18
19     model.addAttribute("brands", brands);
20     return "result";
21   }
22 }
23
24
```

```
result.html
1 <!DOCTYPE html>
2 <html xmlns:th="http://thymeleaf.org">
3 <head><meta charset="UTF-8"/></head>
4 <body>
5
6   <h2>Beer Recommendation</h2>
7
8   <div th:each="beer: ${brands}">
9     <span th:text="${beer}"></span>
10  </div>
11
12 </body>
13 </html>
14
15
```

```
*BeerExpert.java
5 public class BeerExpert {
6   public ArrayList<String> getBrands(String color) {
7     ArrayList<String> brands = new ArrayList<String>();
8     if (color.equals("amber")) {
9       brands.add("Singha");
10      brands.add("Asahi");
11    } else {
12      brands.add("Carlsberg");
13      brands.add("Heineken");
14      brands.add("Tiger");
15    }
16    return brands;
17  }
18 }
19
```

M



application.properties



กำหนดค่าให้ Render ส่วน View ใหม่ทุกครั้ง โดยไม่ต้องเก็บใน cache

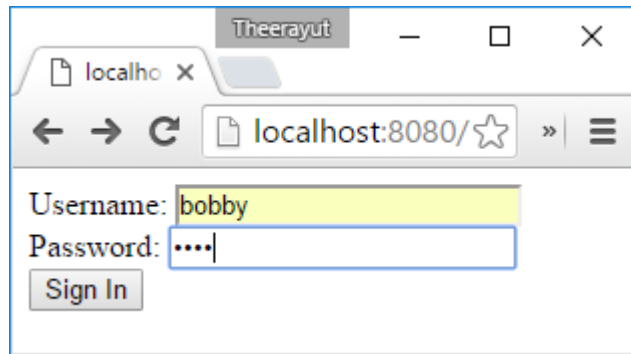
true - cache

false - hot refresh

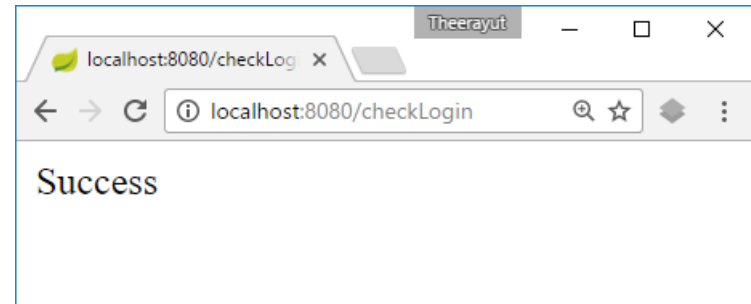
spring.thymeleaf.cache = false

กิจกรรม

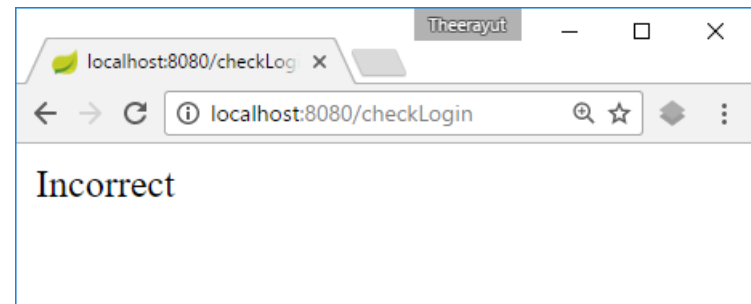
- ❖ จงสร้างเว็บไซต์สำหรับ Sign In เข้าสู่ระบบโดย Spring MVC + Spring Boot กำหนดให้ username เป็น “bobby” และ password เป็น “1234” และแสดงข้อความว่ารหัสผ่านถูกต้องหรือไม่



The screenshot shows a web browser window titled 'Theerayut'. The address bar displays 'localhost:8080/'. The page contains a login form with two input fields: 'Username:' with the value 'bobby' and 'Password:' with masked characters '....'. Below the fields is a 'Sign In' button.



กรณี user และ password ถูกต้อง



กรณี user และ password ไม่ถูกต้อง



Template Engine



- ❖ Template Engine คือ คลาสที่ทำหน้าที่ render หน้าเว็บที่มีตัวแปรหรือ object ที่ประมวลผลเสร็จแล้วจาก Controller ให้อยู่ในรูปแบบการส่ง response กลับไปยังผู้ใช้ เช่น HTML, XML
- ❖ Template Engine ที่ Spring Boot สนับสนุน auto-configuration ได้แก่ FreeMarker, Groovy, Thymeleaf, Velocity
- ❖ Template Engine แต่ละตัวมี syntax ที่แตกต่างกัน โดยส่วนใหญ่จะใช้รูปแบบ HTML เดิม แต่แทรกด้วย attribute หรือสัญลักษณ์ใหม่เข้าไป
- ❖ ไฟล์ที่สร้างสำหรับใช้ในการ Render จะเก็บในโฟลเดอร์
`src/main/resources/templates`
- ❖ Spring Boot ไม่แนะนำให้ใช้ JSP ในการแสดงผล



Thymeleaf



- ❖ Thymeleaf คือ Java Template Engine หนึ่งที่มีความเป็น Natural Templates สูง
- ❖ Natural Templates คือ Template ที่สามารถแสดงผลได้อย่างถูกต้อง แม้ว่าจะถูกแทรกคำสั่งต่างๆ ไปแล้ว ก็ไม่ทำให้การแสดงผลผิดเพี้ยน
- ❖ การแสดงผลถูกต้อง หมายถึง สามารถเปิดบน browser แบบ offline โดยไม่ต้องผ่านการ render ก็สามารถทำงานได้

```
<table>
  <caption>Product list</caption>
  <thead>
    <tr>
      <th>Description</th>
      <th>Price</th>
      <th>Available from</th>
    </tr>
  </thead>
  <tbody>
    <%
      List<Product> products = ProductDAO.getAllProducts();
      for (Product product : products) {
    %>
    <tr>
      <td><%= product.getDescription() %></td>
      <td><%= df.format(product.getPrice()) %></td>
      <td><%= sdf.format(product.getAvailableFrom()) %></td>
    </tr>
    <%
      }
    %>
  </tbody>
</table>
```

JSP

```
<table>
  <caption>Product list</caption>
  <thead>
    <tr>
      <th>Description</th>
      <th>Price</th>
      <th>Available from</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="product : ${products}">
      <td th:text="${product.description}">Chair</td>
      <td th:text="${#numbers.formatDecimal(product.price, 0, 2)}">30.00</td>
      <td th:text="${#dates.format(product.availableFrom, 'dd/MM/yyyy')}">
        27/12/2012
      </td>
    </tr>
  </tbody>
</table>
```

Thymeleaf



โครงสร้าง Template ของ Thymeleaf

กำหนด Namespace ของ Thymeleaf

```
<html xmlns:th="http://thymeleaf.org">
```

```
<head>
```

```
<meta charset="UTF-8"/>
```

```
</head>
```

```
<body>
```

```
<!-- ใช้แท็ก HTML ร่วมกับ attribute ของ Thymeleaf -->
```

```
</body>
```

```
</html>
```




แท็ก HTML แบบปิดใน Thymeleaf



- ❖ แท็ก HTML ที่เป็น close tag เช่น `
`, ``, `<meta>` จะต้องใส่แท็กปิด หรือ กำหนดเครื่องหมาย `>` เสมอ เช่น

`
`

``

`<meta charset="UTF-8"/>`

`<input type="text" name="user"/>
`

`<input type="submit"/>`

- ❖ หากไม่ใส่จะแสดง Error บน Console ดังนี้

The element type "meta" must be terminated by the matching end-tag "</meta>"



Expression Syntax



- ❖ Variable Expressions: $\$\{...\}$
- ❖ Selection Variable Expressions: $*\{...\}$
- ❖ Message Expressions: $\#\{...\}$
- ❖ Link URL Expressions: $@\{...\}$
- ❖ String concatenation: $+$
- ❖ Binary operators: $+$, $-$, $*$, $/$, $\%$
- ❖ Binary operators: and , or
- ❖ Boolean negation (unary operator): $!$, not
- ❖ Comparators: $>$, $<$, $>=$, $<=$ (gt , lt , ge , le)
- ❖ Equality operators: $==$, $!=$ (eq , ne)

การแสดงค่าจากตัวแปร หรือ object

❖ เมื่อต้องการนำค่าจาก object ที่ถูกกำหนดไว้ใน model จาก controller จะใช้ attribute

th:text

แสดงค่าจากตัวแปรชื่อว่า beer

`Beer Name`

สามารถใส่ attribute อื่นในแท็กเพื่อ
จัดรูปแบบได้ เช่น class, style หรือใช้
แท็กอื่นๆแทน span ก็ได้ เช่น <p>, <div>

ข้อความในแท็กจะใช้สำหรับการออกแบบ
เท่านั้น จะแสดงเมื่อเปิดไฟล์ HTML ปกติ ที่
ไม่ผ่าน Template Engine

ผลการ Render

`Carlsberg`



สร้าง URL ที่แนบข้อมูลไปด้วย



❖ หากต้องการส่งข้อมูลจากตัวแปรไปพร้อมกับ URL มีรูปแบบ @{ } เช่น

```
<a th:href="@{editform(id=${pid})}">Edit</a>
```

ผลการ Render

```
<a href="editform?id=69">Edit</a>
```

```
<a th:href="@{editform/{id}(id=${pid})}">Edit</a>
```

ผลการ Render

```
<a href="editform/69">Edit</a>
```

การวนลูป

- ❖ ในบางครั้ง object ที่อยู่ในโมเดล อาจอยู่ในรูปแบบ Array ดังนั้นเราสามารถใช้คำสั่งเพื่อวนลูปแสดงค่าจากอาร์เรย์ได้

```
<div th:each="beer: ${brands}">
  <span th:text="${beer}"></span>
</div>
```

ผลการ Render

```
<div>
  <span>Carlsberg</span>
</div>
<div>
  <span>Heineken</span>
</div>
<div>
  <span>Tiger</span>
</div>
```




การตรวจสอบเงื่อนไข

❖ th:if

```
<span th:if="${product.price lt 100}">Special offer!</span>
```

❖ th:unless

```
<span th:unless="${product.notInStock}">Currently in stock!</span>
```



การกำหนดค่าใน application.properties



❖ เป็นการกำหนดส่วน Auto Configuration สำหรับ Thymeleaf

```
spring.thymeleaf.cache=true # set to false for hot refresh  
spring.thymeleaf.check-template-location=true  
spring.thymeleaf.prefix=classpath:/templates/
```

```
spring.thymeleaf.suffix=.html  
spring.thymeleaf.mode=HTML5  
spring.thymeleaf.encoding=UTF-8
```

```
# ;charset=<encoding> is added  
spring.thymeleaf.content-type=text/html
```

```
# comma-separated list of view names that should be excluded from resolution  
spring.thymeleaf.excluded-view-names=
```

```
# comma-separated list of view names that can be resolved  
spring.thymeleaf.view-names=
```



การ Integration Spring ส่วนต่างๆ

- ❖ Spring Data JPA ใช้ access ฐานข้อมูล
- ❖ Spring MVC สร้างส่วนประมวลผลตามคำร้องจาก client
- ❖ Thymeleaf ออกแบบส่วน view ของระบบ



ToDo App

- ❖ เว็บแอปพลิเคชัน ToDo ใช้สำหรับบันทึกงานที่ต้องทำในแต่ละวัน ซึ่งออกแบบซอฟต์แวร์ตามสถาปัตยกรรม MVC โดยใช้ Spring Framework การทำงานประกอบด้วย แสดงรายการงานที่ต้องทำ เพิ่ม แก้ไข ลบ และค้นหารายการที่ทำได้

ตัวอย่างหน้าจอ

The screenshot shows a web browser window with the title 'Theerayut'. The address bar shows 'localhost:8080/' and the page URL is 'localhost:8080/task'. The main content area has a light blue background and the title 'THING TO DO' in bold. Below the title, there is a list of tasks: 'ซื้อของใช้ แก้ไข ลบ', 'ฝากเงิน แก้ไข ลบ', 'วิ่ง แก้ไข ลบ', and 'โทรกลับบ้าน แก้ไข ลบ'. At the bottom, there are two input fields: the first is for adding a new task, followed by an 'add' button, and the second is for searching, followed by a 'search' button.



POM: Web + JPA + Thymeleaf



❖ สร้าง Maven Project ใหม่ และเปิดไฟล์ pom.xml เพิ่ม Library ดังนี้

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.5.2.RELEASE</version>
</parent>
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
  </dependency>
</dependencies>
```

สืบทอดค่าเริ่มต้นของ Spring Boot

Dependency สำหรับพัฒนาเว็บแอปพลิเคชันแบบ MVC

JDBC Driver สำหรับติดต่อฐานข้อมูล MySQL

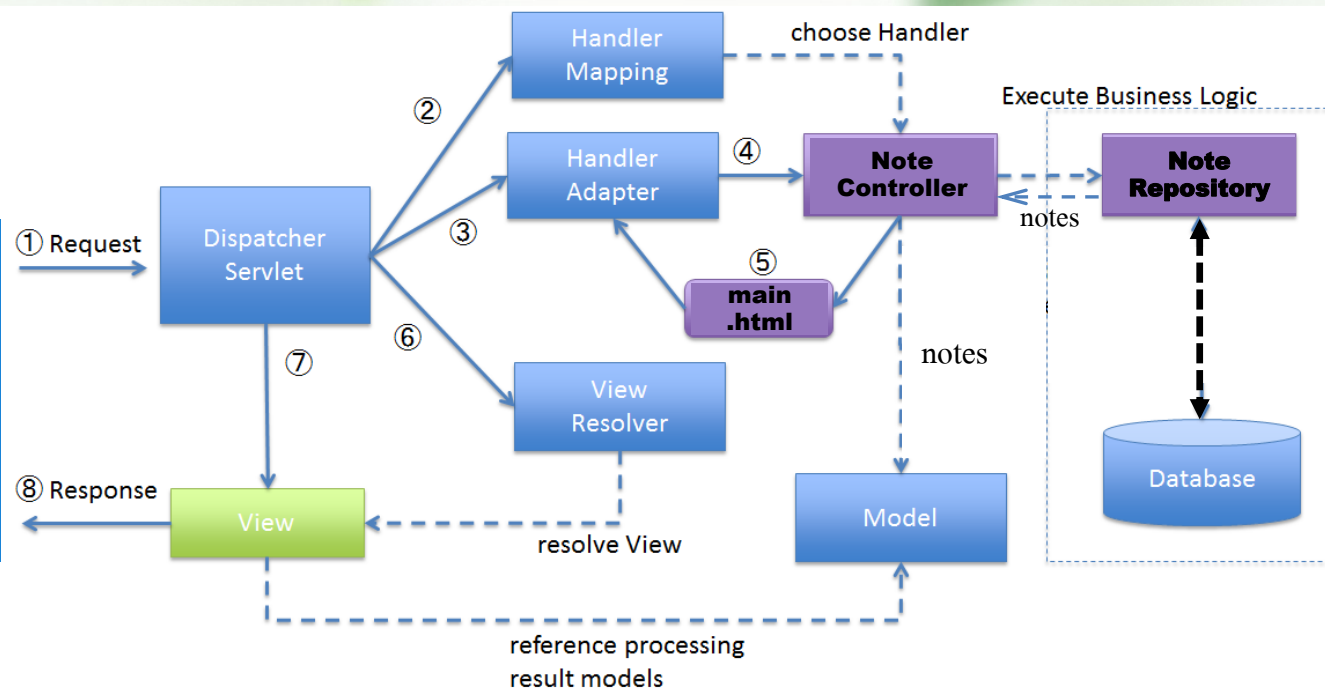
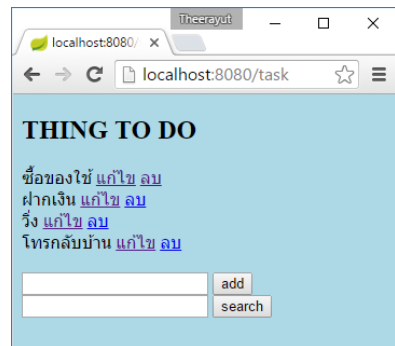
ใช้ JPA สำหรับจัดการฐานข้อมูล

ใช้ Thymeleaf สำหรับ render ส่วน View

ติดตามการแก้ไขโค้ด และ reload แบบอัตโนมัติ



การประมวลผลคำร้องใน Spring MVC



สร้างโดยนักพัฒนา

Spring สร้างให้แล้ว

บางส่วน Spring สร้างให้แล้ว บางส่วนต้อง implement เพิ่ม



application.properties



กำหนดค่าเกี่ยวกับการติดต่อกับฐาน

```
spring.datasource.url = jdbc:mysql://localhost/mydb?characterEncoding=utf-8  
spring.datasource.username = root  
spring.datasource.password =
```

กำหนดค่าให้แสดงคำสั่ง SQL บน console: true แสดง, false ไม่แสดง

```
spring.jpa.show-sql = true
```

กำหนดให้สร้างคำสั่ง DDL แบบอัตโนมัติ เมื่อเจอ Entity class

ค่าที่กำหนดได้ประกอบด้วย none, validate, update, create, create-drop

```
spring.jpa.hibernate.ddl-auto = update
```



Entity Class

```
import javax.persistence.*;
```

```
@Entity
```

```
public class Note implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue(strategy=GenerationType.AUTO)
```

```
    private Integer nid;
```

```
    private String task;
```

```
    public Integer getNid() { return nid; }
```

```
    public void setNid(Integer nid) { this.nid = nid; }
```

```
    public String getTask() { return task; }
```

```
    public void setTask(String task) { this.task = task; }
```

```
}
```

โครงสร้างตาราง Note

ชื่อฟิลด์	คำอธิบาย	ชนิดข้อมูล	หมายเหตุ
nid	รหัสโน้ตงาน	INT	คีย์หลัก แบบเพิ่มค่าอัตโนมัติ
task	ชื่องานที่ต้องทำ	VARCHAR(100)	



Repository Interface

```
public interface NoteRepository  
    extends CrudRepository<Note, Integer> {  
  
    List<Note> findByTaskContains(String tname);  
}
```



Controller

- ❖ สร้างคลาสใหม่ชื่อ NoteController และ ใส่ @Controller เหนือชื่อคลาส
- ❖ ประกาศ object ของ NoteRepo โดยกำหนด @Autowired

```
@Controller
public class NoteController {

    @Autowired
    private NoteRepository noteRepo;

    @GetMapping("/note")
    public String listAll(Model model) {
        Iterable<Note> list = noteRepo.findAll();
        model.addAttribute("noteList", list);
        return "result";
    }

    // ต่อหน้าต่อไป
```



Controller

```
@GetMapping("/searchnote")
public String search(@RequestParam("task") String task, Model model) {
    List<Note> list = noteRepo.findByTaskContains(task);
    model.addAttribute("noteList", list);
    return "result";
}

@PostMapping("/addnote")
public String add(@RequestParam("task") String task) {
    Note note = new Note();
    note.setTask(task);
    noteRepo.save(note);
    return "redirect:/note";
}

@GetMapping("/delnote/{id}")
public String delete(@PathVariable("id") Integer id) {
    noteRepo.delete(id);
    return "redirect:/note";
}

@GetMapping("/editform/{id}")
public String showEditForm(@PathVariable("id") Integer id, Model model) {
    Note note = noteRepo.findOne(id);
    model.addAttribute("note", note);
    return "editform";
}

@PostMapping("/editnote")
public String edit(@ModelAttribute Note note, Model model) {
    noteRepo.save(note);
    return "redirect:/note";
}
```




ส่วน View



```
<html xmlns:th="http://www.thymeleaf.org">
<head><meta charset="UTF-8"/></head>
<body bgcolor="lightblue">
  <h2>THING TO DO</h2>
  <div th:each="note : ${noteList}">
    <span th:text="${note.task}"></span>
    <a th:href="@{editform/{id}(id=${note.nid})}">แก้ไข</a>
    <a th:href="@{delnote/{id}(id=${note.nid})}">ลบ</a> <br/>
  </div><br/>

  <form action="addnote" method="post">
    <input type="text" name="task"/>
    <input type="submit" value="add"/>
  </form>

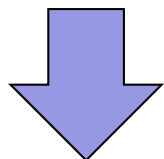
  <form action="searchnote">
    <input type="text" name="task"/>
    <input type="submit" value="search"/>
  </form>
</body>
</html>
```



แบบฟอร์มแก้ไขข้อมูล (editform.html)



```
<html xmlns:th="http://www.thymeleaf.org">
<head><meta charset="UTF-8"/></head>
<body bgcolor="lightblue">
  <form action="/editnote" th:object="${note}" method="post">
    <input type="hidden" th:field="*{nid}"/>
    <input type="text" th:field="*{task}"/>
    <input type="submit" value="update"/>
  </form>
</body></html>
```



เมื่อ Thymeleaf Engine ทำการ Render เสร็จแล้วจะได้ response กลับไปยังผู้ใช้ ดังนี้

```
<html>
<head><meta charset="UTF-8" /></head>
<body bgcolor="lightblue">
  <form action="/editnote" method="post">
    <input type="hidden" id="nid" name="nid" value="4" />
    <input type="text" id="task" name="task" value="ส่งพัสดุ" />
    <input type="submit" value="update" />
  </form>
</body></html>
```