

SPRING BEAN CONTROLLER THYMELEAF

GV: Nguyễn Huy Cường
Email: nh.cuong@hutech.edu.vn

Nội dung

1. Spring bean

DI

IoC Containers

Tạo bean trong Spring

Phạm vi và vòng đời

2. Controller

3. Thymeleaf

SPRING BEAN

@Component, @Controller, @Service, @Repository

@ComponentScan

@Bean @Configuration

@Autowired @Primary @Qualifier

Vấn đề

- Có thể quản lý được đối tượng (thực thể) đã tạo ra ?
Theo singleton, Request, Session, Prototype, Application... ?

Ví dụ:

Có 1 đối tượng “gold9999” thuộc lớp **Gold**(name, price). Viết Controller có 2 Action:
(1) lấy price hiện tại (2) tăng price lên 10%.

Yêu cầu: Quản lý được đối tượng tạo ra

- ☐ Quản lý đối tượng theo thực thể duy nhất
- ☐ Quản lý đối tượng theo từng yêu cầu request
- ☐ Quản lý đối tượng theo session

....

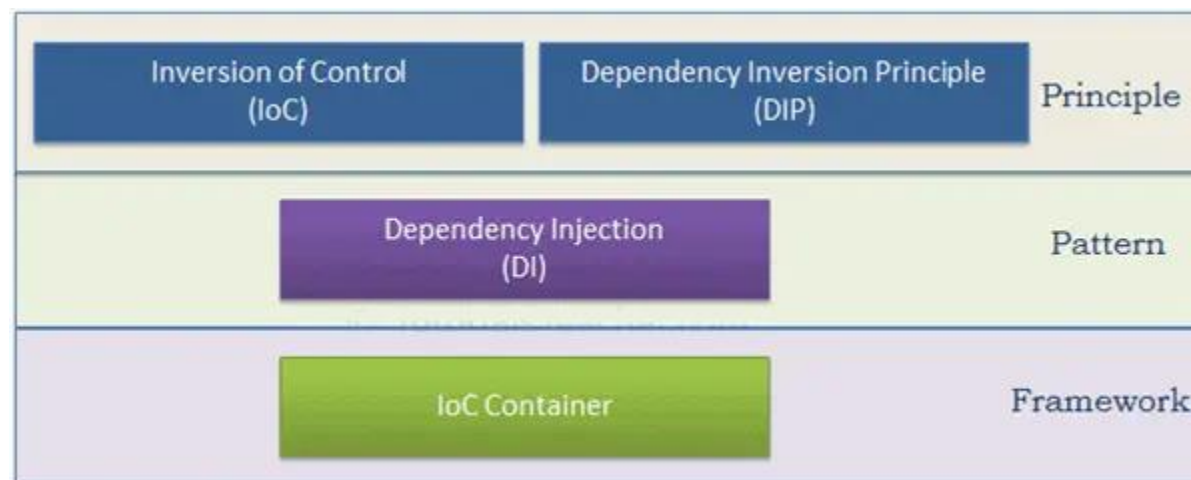
```
@Getter @Setter  
@AllArgsConstructor  
public class Gold {  
    public String name;  
    public Integer price;  
}
```

Vấn đề

- Có thể quản lý được đối tượng (thực thể) đã tạo ra ? Và có thể chia sẻ được trong toàn ứng dụng tuân thủ nguyên lý *Dependency Inversion Principle*?

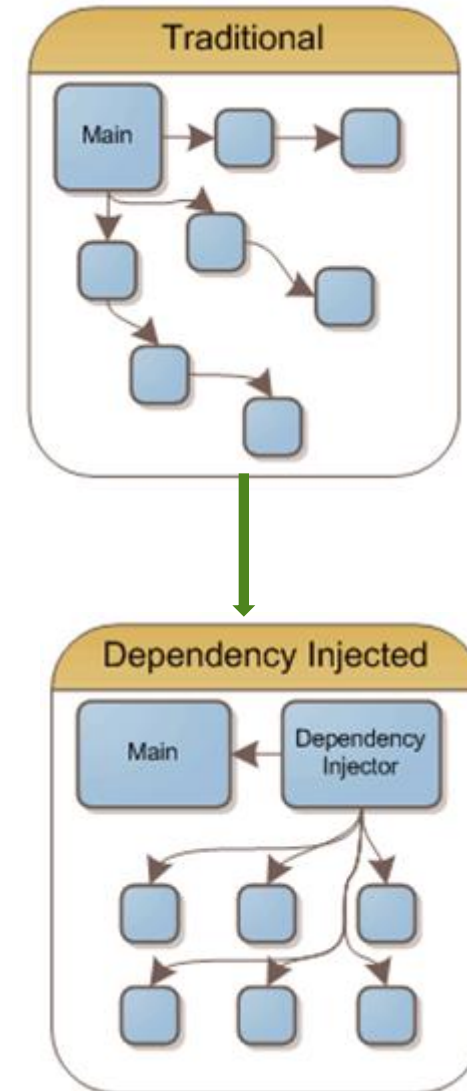
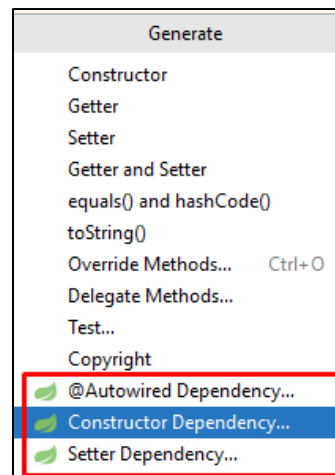
=> Sử dụng Spring Bean => được lưu trữ trong iOC Container => Phải đánh dấu là Spring Bean

- ❑ **Inversion of Control:** là một design pattern được tạo ra để code tuân thủ nguyên lý Dependency Inversion (thiết kế và viết code). Có nhiều cách hiện thực pattern này: ServiceLocator, Delegate, Event, Dependency Injection (DI),...



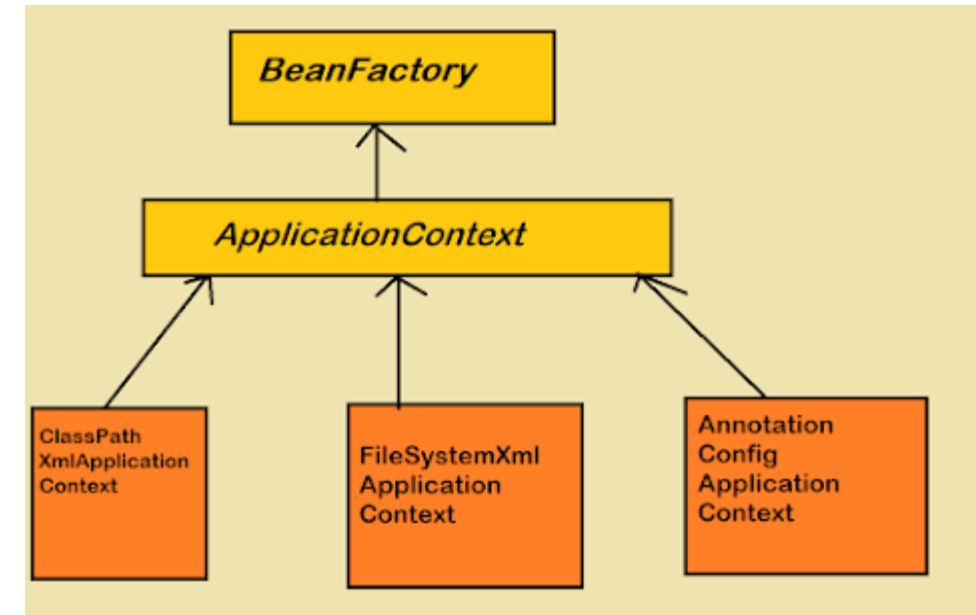
DI

- **Dependency Injection (DI)** là 1 mẫu thiết kế giúp loại bỏ việc phụ thuộc trực tiếp của class với đối tượng, được sử dụng để thực hiện Inversion of Control(IoC). Các module phụ thuộc (dependency) sẽ được inject vào module cấp cao
 - ❑ Quản lý sự phụ thuộc (dependencies) giữa các đối tượng.
 - ❑ Cung cấp các phụ thuộc (inject) được yêu cầu cho đối tượng (được truyền từ bên ngoài đối tượng).
- **Có 3 cách để thực hiện DI:**
 - ❑ Field
 - ❑ Constructor
 - ❑ Setter



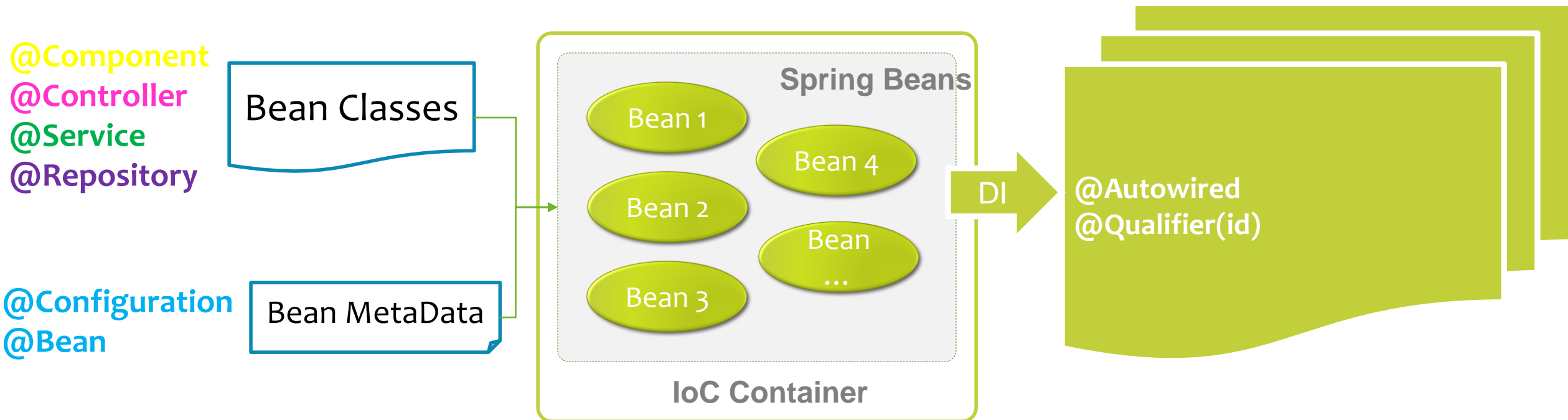
IoC Containers

- IoC Container sẽ tạo ra các đối tượng (được gọi là **Spring Bean**), nối chúng lại với nhau, cấu hình, và quản lý vòng đời của chúng **từ khi tạo ra đến khi bị hủy**. IoC Container sử dụng DI ([Dependency Injection](#)) để quản lý các thành phần tạo nên một ứng dụng.
- Spring Framework cung cấp 2 loại IoC Containers :
 - ❑ Bean Factory
 - ❑ Application Context



Spring bean là gì?

- Spring Beans chính là những Java Object mà từ đó tạo nên khung sườn của một ứng dụng Spring và được cài đặt, quản lý bởi inversion of Control (**IoC**).
 - ❑ Các bean có thể **phụ thuộc lẫn nhau**. Sự phụ thuộc này được mô tả cho **IoC** biết nhờ cơ chế **Dependency injection (DI)**.



Tạo bean trong Spring

- Để IoC Container nhận ra một class là **Spring Bean**, thì cần đánh dấu lớp đó trong tệp cấu hình XML hoặc bằng cách sử dụng các chú thích (annotations).

- ☐ C1: Tạo bean từ XML

-> khó phát hiện lỗi khi biên dịch, bảo trì -> nên sử dụng java config

- ☐ C2: Tạo bean từ Java-base configuration:

Tạo bean trên phương thức **@Bean** trong lớp **@Configuration**

- ☐ C3: Tạo bean từ annotation: **@Component**, **@Controller**, **@Service**, **@Resitory**

- Sử dụng **Spring Bean**

@Autowired được sử dụng để inject một phụ thuộc trong spring

Nếu có nhiều bean có cùng kiểu dữ liệu:

- ☐ **@Primary** được sử dụng kèm với **@Bean** để báo cho IoC biết đây là bean chính. Nếu có các bean cùng kiểu thì Spring sẽ ưu tiên liên kết với bean này

- ☐ **@Qualifier("id")**: Đặt id cho bean và sử dụng **@Qualifier(id)** bên cạnh **@Autowired**

Cấu hình tạo bean

C 1 - Từ XML Config

Beans.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean id="9999-bean" class="laptrinhungdungjava.demo.Gold">
    <property name="name" value="9999" />
    <property name="price" value="7890" />
  </bean>
</beans>
```

TẠI SAO BIẾN G KHÔNG NULL?

Spring sẽ tìm kiếm đối tượng (bean) trong môi trường của nó **có kiểu tương ứng với biến và liên kết biến với đối tượng tìm thấy** hoặc báo lỗi nếu không tìm thấy.

@Configuration

@ImportResource({"classpath*:beans.xml"})

public class GoldFactory {

@RestController

@RequestMapping("/gold")

public class GoldController {

@Autowired

private Gold g;

@GetMapping("")

@ResponseBody

public int Index()

{

return g.getPrice();

}

@GetMapping("/update")

public int Update()

{

g.setPrice((int) (g.getPrice() * 1.1));

return g.getPrice();

}

}

C2 - Từ phương thức **@Bean** ở lớp **@Configuration**

- Cấu hình dựa trên Java bằng cách sử dụng **@Bean** bên trong 1 lớp **@Configuration**.
 - ❑ Mỗi **@Bean** được khai báo cho một method để tạo ra 1 Spring Bean.
 - ❑ **@Configuration** – một class chứa các cấu hình Spring bean.

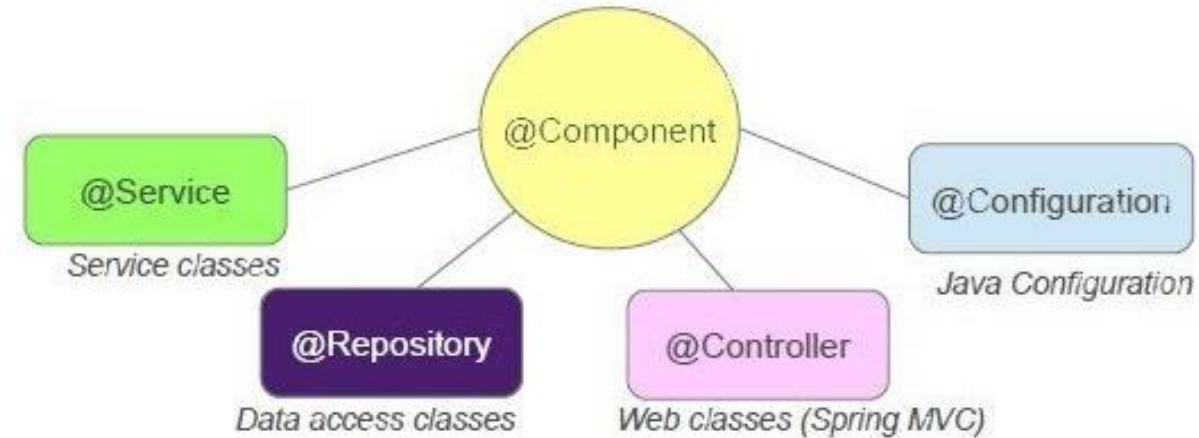
@Configuration

```
public class GoldFactory {  
    @Bean  
    public Gold gold()  
    {  
        return new Gold("9999", 7890);  
    }  
}
```

```
@RestController  
@RequestMapping("/gold")  
public class GoldController {  
    @Autowired  
    private Gold g;  
    @GetMapping("")  
    @ResponseBody  
    public int Index()  
    {  
        return g.getPrice();  
    }  
    @GetMapping("/update")  
    public int Update()  
    {  
        g.setPrice((int) (g.getPrice() * 1.1));  
        return g.getPrice();  
    }  
}
```

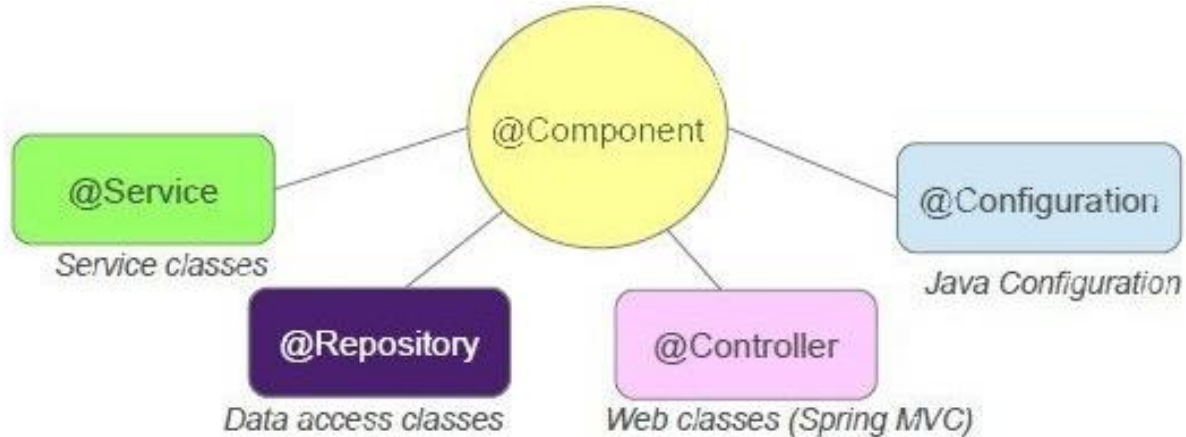
C3- Từ annotation: @Component, @controller, @Service, @Repository

- Chú thích cho **class** để đánh dấu là 1 bean
- Spring quét và đăng ký với **ApplicationContext**
- ❑ **@Repository**: để chỉ áp dụng trên các DAO (Data Access Object) class dùng để thao tác với database
- ❑ **@Service**: để chỉ thuộc về tầng service nắm giữ code xử lý business
- ❑ **@Controller**: là nơi tiếp nhận request và trả về response cho client
- @ComponentScan: mặc định sẽ quét tất cả các class ở cùng package ở main class và tất cả các sub-package. Trong @ComponentScan chúng ta có thể sử dụng thuộc tính basePackages, nhận vào danh sách các package Spring sẽ quét



Cấu hình tạo bean

Cách 3: Từ annotation – Ví dụ



@Getter @Setter

@AllArgsConstructor

@Component

```
public class Gold {
    public String name;
    public Integer price;
    public Gold()
    {
        name = "9999";
        price = 7890;
    }
}
```

```
@RestController
@RequestMapping("/gold")
public class GoldController {
    @Autowired
    private Gold g;
    @GetMapping("")
    @ResponseBody
    public int Index()
    {
        return g.getPrice();
    }
    @GetMapping("/update")
    public int Update()
    {
        g.setPrice((int) (g.getPrice() * 1.1));
        return g.getPrice();
    }
}
```

Phạm vi của bean trong spring

- Phạm vi của bean xác định thời gian tồn tại, có thể khai báo phạm vi của bean, mặc định của bean là **singleton**.

☐ Singleton

☐ Request

`@Scope(value = "request", proxyMode = ScopedProxyMode.TARGET_CLASS)`

☐ Session

`@Scope(value = "session", proxyMode = ScopedProxyMode.TARGET_CLASS)`

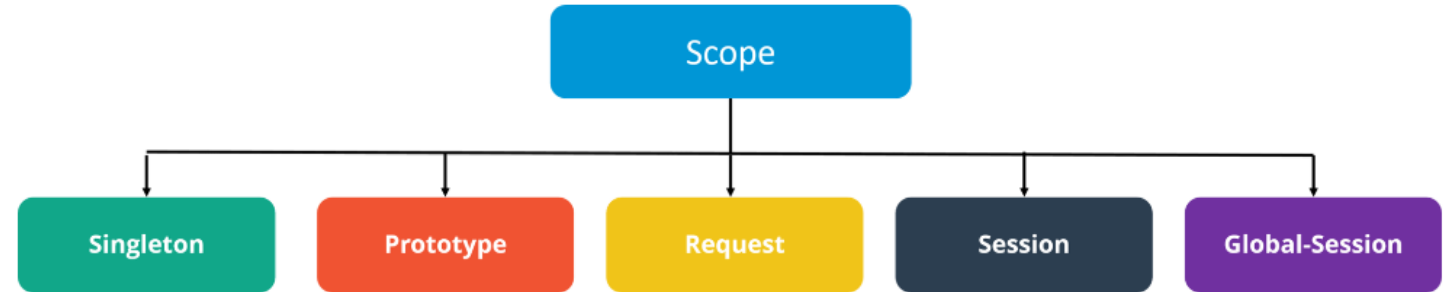
☐ Prototype

`@Scope(value = "prototype", proxyMode = ScopedProxyMode.TARGET_CLASS)`

☐ Application

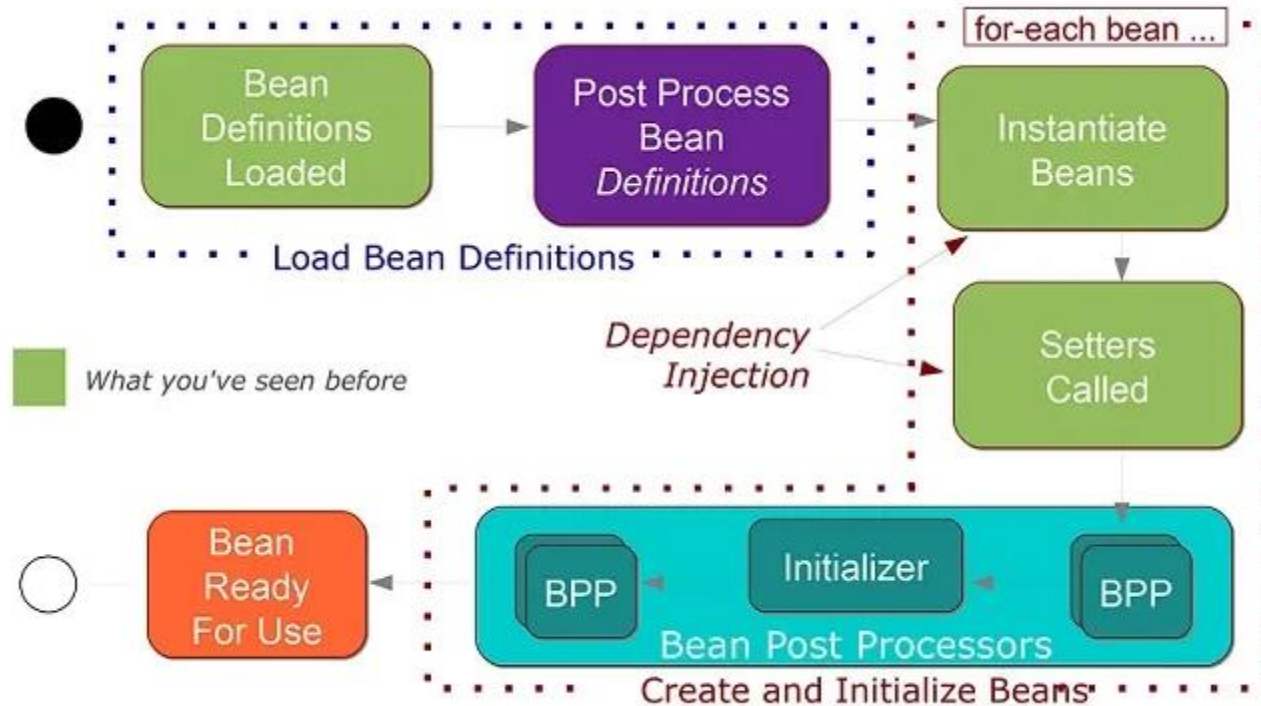
`@Scope(value = "application", proxyMode = ScopedProxyMode.TARGET_CLASS)`

=> Thực hiện lại quản lý đối tượng gold để hiểu rõ Singleton/ **Request** / **Session**...



Vòng đời của bean

- Phạm vi của bean xác định thời gian tồn tại.



CONTROLLER

@Controller

@RequestMapping @GetMapping @PostMapping @PutMapping @DeleteMapping

@RequestParam

Controller

- Định nghĩa controller bằng: **@Controller**
- Controller có thể chứa **action** khác route với nhau.

@RequestMapping: chấp nhận tất cả các loại (get, post, put, delete ...)

ví dụ: `@RequestMapping(value = "/home/demo1", method = RequestMethod.GET)`

Hoặc từng loại action qui định riêng:

@GetMapping,

@PostMapping

@PutMapping

@DeleteMapping

...

Định tuyến controller

- Có thể đặt tất cả actions trong controller bằng cách đặt mapping ở lớp controller

Ví dụ:

-> Để gọi GetAll() thì cần route: “/books”

```
@Controller
@RequestMapping("/books")
public class BookController {
    no usages
    @GetMapping()
    public String GetAll(Model model)
    {
        return "index";
    }
}
```

```
no usages
@GetMapping("/home/index")
//@RequestMapping(value = "/home/index")
public String index()
{
    return "index";
    // return "index.html";
}
no usages
@ResponseBody
@GetMapping("/test")
public String test()
{
    return "Lập trình Java";
}
```

Tên view

Dữ liệu trực tiếp

@RequestParam và @PathVariable

- **@RequestParam:** Lấy giá trị của parameters trên URL

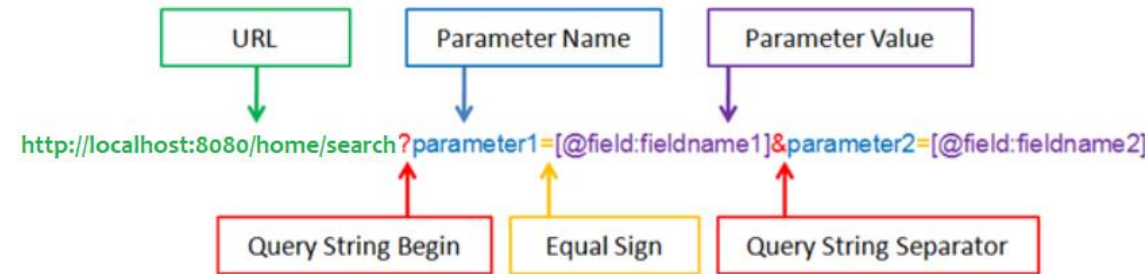
- ❑ Có thể sử dụng default value
- ❑ Sử dụng cho cả Post, Get

```
@GetMapping("/search")
public String Search(@RequestParam String parameter1, @RequestParam String parameter2)
{
    return "search";
}
```

- **@PathVariable:** để lấy giá trị trên URI.

- ❑ Các tham số được chú thích với @PathVariable mặc định phải khác NULL

```
@ResponseBody
@GetMapping("/book/{bookid}")
public String index(@PathVariable int bookid)
{
    return "Mã sách: " + bookid;
}
```



Redirect và forward

- Redirect: *Chuyển hướng* sang url khác (other request)
- Forward: Chuyển tiếp sang url khác (same request)

no usages

```
@GetMapping("/home/index")
```

```
public String index()
```

```
{
```

```
    return "index";
```

```
}
```

no usages

```
@RequestMapping(value = "/redirect1")
```

```
public String redirect1()
```

```
{
```

```
    return "redirect:/home/index"; // home/index
```

```
}
```

no usages

```
@RequestMapping(value = "/redirect2")
```

```
public String redirect2()
```

```
{
```

```
    return "redirect:https://vnexpress.net";
```

```
}
```

no usages

```
@RequestMapping(value = "/forward")
```

```
public String forward()
```

```
{
```

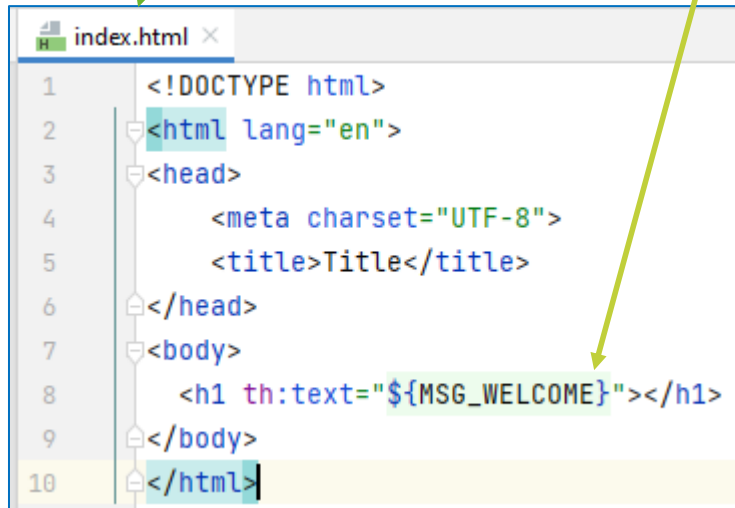
```
    return "forward:/home/index";
```

```
}
```

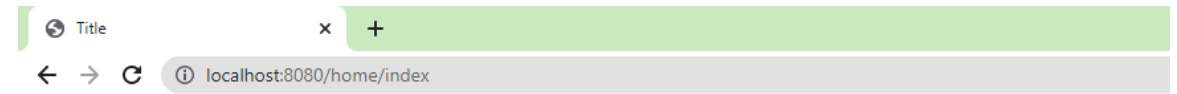
Truyền attribute sang view

- Sử dụng **Model** model qua phương thức **addAttribute**

```
@GetMapping("/index")
public String index(Model model)
{
    model.addAttribute(attributeName: "MSG_WELCOME", attributeValue: "Hi, Nguyen Huy Cuong!");
    return "index";
}
```



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8     <h1 th:text="${MSG_WELCOME}"></h1>
9 </body>
10 </html>
```



Hi, Nguyen Huy Cuong!

Form objects

- Các **name** của thẻ input phải cùng với **tên thuộc tính trong object**
- Spring tự động fill objects với form data

```
@PostMapping("/create")
@ResponseBody
public String create(int id, String title, String author, long price)
{
    return "Created Book title: %s, author: %s ".formatted(title, author);
}
```

```
@PostMapping("/create")
@ResponseBody
public String create(@ModelAttribute Book book)
{
    return "Created Book title: %s, author: %s ".formatted(book.getTitle(), book.getAuthor());
}
```

```
<body>
<form method="post" th:action="@{/home/create}">
    <label for="id">id:</label>
    <input type="number" id="id" name="id" required>

    <label for="title">Title:</label>
    <input type="text" id="title" name="title" required>

    <label for="author">Author:</label>
    <input type="text" id="author" name="author" required>

    <label for="price">Price:</label>
    <input type="number" id="price" name="price" required>

    <input type="submit" value="Create Book">
</form>
</body>
```

THYMELEAF

Thymeleaf

- Thymeleaf là 1 **Java template engine**. Có nhiệm vụ tạo ra các file HTML, XML, JS, CSS...
- Để sử dụng thymeleaf:

❑ Import “**Thymeleaf**” từ Spring Initializer
hoặc thêm từ dependency trong **pom.xml**

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

- Các file thymeleaf template là file .html có
<html xmlns:th="http://www.thymeleaf.org">



Sử dụng controller và thymeleaf

- Tạo **HomeController** là controller: **@Controller**
 - ❑ phương thức **index** sử dụng **@GetMapping** trả về “index”
- View ở templates
 - ❑ Thêm **index.html**

```
@Controller
public class HomeController {
    no usages
    @GetMapping("/")
    public String index()
    {
        return "index";
    }
}
```

The screenshot shows an IDE with a project named 'thymeleaf' located at 'D:\MonHoc\Java-Web\BaiGiang2023\thymeleaf'. The project structure is as follows:

- thymeleaf
 - .idea
 - .mvn
 - src
 - main
 - java
 - laptrinhungdungjava.thymeleaf
 - controller
 - HomeController
 - ThymeleafApplication
 - resources
 - static
 - templates
 - index.html
 - test
 - target
 - .gitignore

The 'index.html' file is open in the editor, showing the following HTML code:

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <title>Title</title>
6 </head>
7 <body>
8     <h1>Lập trình ứng dụng Java</h1>
9 </body>
10 </html>
```

A red arrow points from the 'index.html' file in the project structure to the corresponding line in the code editor.

Sử dụng bootstrap – Javascript

- Lấy mẫu sử dụng: <https://getbootstrap.com/>
- Có 2 cách để thêm bootstrap CSS/ JS vào Thymeleaf template:

❑ c1: Sử dụng CDN links

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css" rel="stylesheet" crossorigin="anonymous">
```

C2: Download về local, thêm vào project ở **src/main/resources/static** folder và thêm link vào phần header vào Thymeleaf template

```
<link th:href = "@{/css/bootstrap.min.css}" rel="stylesheet">
```

```
<script type="text/javascript" th:src="@{/js/popper.min.js}"></script>
```

Thymeleaf Engine

- Thymeleaf Engine sẽ phân tích teampalte và thay thế các vị trí được đánh dấu trên Thymeleaf Template để tạo một html mới.
- View ở templates

← → ↻ localhost:8080/products

Home Product

#	Name	Image	Price
1	Sản phẩm 1	1.jpg	29312
2	Sản phẩm 2	2.jpg	124246

```
<head>
  <meta charset="UTF-8">
  <title>List product</title>
</head>
<body>
  <div layout:fragment="content" class="container body-content">
    <table class="table table-striped">
      <thead>
        <tr>
          <th scope="col">#</th>
          <th scope="col">Name</th>
          <th scope="col">Image</th>
          <th scope="col">Price</th>
        </tr>
      </thead>
      <tbody>
        <tr th:each="product : ${listproduct}" >
          <th scope="row" th:text="${product.id}"></th>
          <td th:text="${product.name}"></td>
          <td th:text="${product.image}"></td>
          <td th:text="${product.price}"></td>
        </tr>
      </tbody>
    </table>
  </div>
</body>
</html>
```

Cú pháp

- Các thymeleaf tag và thuộc tính bắt đầu bằng **th:**

Ví dụ: `<p th:text = “Lập trình ứng dụng Java- Nguyễn Huy Cường” >`
`<th:block> </th:block>`

- **Biến**

`${...}`: Giá trị của một biến

`*{...}`: Giá trị của một biến được chỉ định

`#{...}`: Giá trị message

`@{...}`: Lấy đường dẫn URL dựa theo context của server

- Thuộc tính:

`th:text / th:utext` → Lấy chuỗi thuần / Lấy chuỗi có thẻ HTML, CSS

`th:each` → Vòng lặp

`th:if, th:unless / th:switch, th:case` → Câu lệnh điều kiện

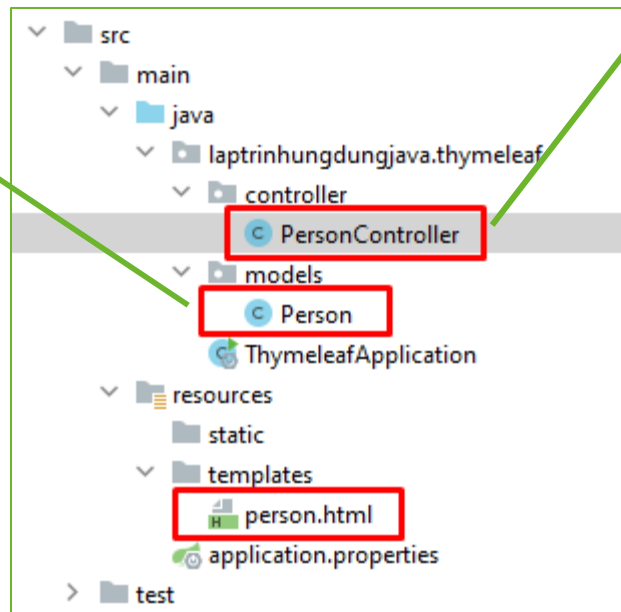
`th:object / th:field` → Chỉ định đối tượng / Dữ liệu của đối tượng

DEMO

Sử dụng controller + thymeleaf

```
public class Person {  
    private String name;  
    private Integer age;  
    private Integer sex;  
    private boolean isMarried;  
    private LocalDate createTime;  
    private List<String> language;
```

//Viết constructor, Getter ,
Setter
hoặc sử dụng Lombok

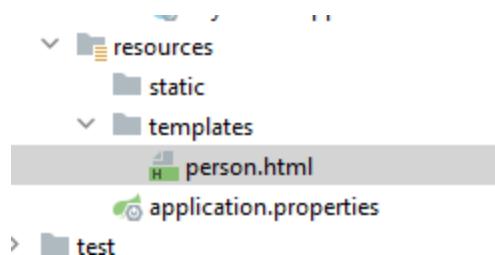


@Controller

```
public class PersonController {  
    @GetMapping("/person")  
    public String index(Model model)  
    {  
        model.addAttribute("info", "<b>Thông tin </b>  
</br>");
```

```
        Person p = new Person();  
        p.setName("Thymeleaf Spring Boot");  
        p.setAge(36);  
        p.setSex(1);  
        p.setMarried(true);  
        p.setCreateTime(LocalDate.now());
```

```
        p.setLanguage(Arrays.asList("Java","C#","Python")  
        );  
        model.addAttribute("person", p);  
        return "person.html";  
    }  
}
```



1- Simple attribute (1)

- **Controller:** Thêm `model.addAttribute`
`model.addAttribute("attributeName", "attributeValue");`

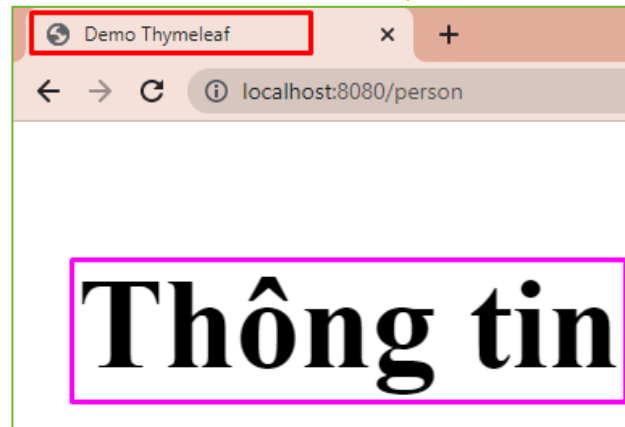
@Controller

```
public class PersonController {  
    @GetMapping("/person")  
    public String index(Model model)  
    {  
        model.addAttribute(  
            "title",  
            "Demo Thymeleaf");  
        model.addAttribute(  
            "info",  
            "<b>Thông tin </b> </br>");  
        return "person";  
    }  
}
```

- **Thymeleaf:** Nhận giá trị bằng `th:text` / `th:utext`

❑ `th:text = "${attributeName}"`

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="UTF-8">  
    <title th:text="${title}">Title</title>  
</head>  
<body>  
    <p><span th:utext="${info}"></span></p>  
</body>  
</html>
```

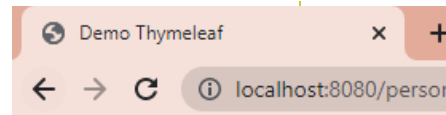


Sử dụng thymleaf:

1- Simple attribute (2)

- **Controller:** Thêm `model.addAttribute`
`model.addAttribute("attributeName", object);`

```
@GetMapping("/person")
public String index(Model model)
{
    model.addAttribute("title", "Demo Thymeleaf");
    model.addAttribute("info", "<b>Thông tin</b><br>");
    Person p = new Person();
    p.setName("Thymeleaf Spring Boot");
    p.setAge(36);
    p.setSex(1);
    p.setMarried(true);
    p.setCreateTime(LocalDate.now());
    p.setLanguage(Arrays.asList("Java", "C#", "Python"));
    model.addAttribute("person", p);
    return "person";
}
```



Thông tin

Thymeleaf Spring Boot

36

1

true

2023-04-30

[Java, C#, Python]

- **C1 Thymeleaf:** `${attributeName.fields}`

```
<body>
  <p><span th:utext="${info}"></span></p>
  <p th:text="${person.name}"></p>
  <p th:text="${person.age}"></p>
  <p th:text="${person.sex}"></p>
  <p th:text="${person.isMarried}"></p>
  <p th:text="${person.createTime}"></p>
  <p th:text="${person.language}"></p>
</body>
</html>
```

- **C2 Thymeleaf:** `*{fields}`

```
<div th:object="${person}">
  <p th:text="*{name}"></p>
  <p th:text="*{age}"></p>
  <p th:text="*{sex}"></p>
  <p th:text="*{isMarried}"></p>
  <p th:text="*{createTime}"></p>
  <p th:text="*{language}"></p>
</div>
```

2- Collections attribute

- Thymeleaf: Sử dụng **th:each**

```
<ul>  
  <li th:each="language:${person. language}" th:text="${language}"></li>  
</ul>
```

- Java
- C#
- Python

Sử dụng thymeleaf:

3- th:if và th:unless

- **Controller:** Thêm model.addAttribute

model.addAttribute("attributeName", object);

```
@GetMapping("/person")
public String index(Model model)
{
    model.addAttribute("title", "Demo Thymeleaf");
    model.addAttribute("info", "<b>Thông tin
</b></br>");
    Person p = new Person();
    p.setName("Thymeleaf Spring Boot");
    p.setAge(36);
    p.setSex(1);
    p.setMarried(true);
    p.setCreateTime(LocalDate.now());

    p.setLanguage(Arrays.asList("Java", "C#", "Python"));
    model.addAttribute("person", p);
    return "person";
}
```

- **C1 Thymeleaf:** \${attributeName.fields}

```
<body>
    <p th:if="${person.isMarried}">Đã kết hôn</p>
    <p th:unless="*{person.isMarried}">Chưa kết hôn</p>
</body>
</html>
```

```
Thymeleaf Spring Boot
36
1
true
Đã kết hôn
2023-04-30
[Java, C#, Python]
```

- **C2 Thymeleaf:** *{fields}

```
<div th:object="${person}">
    <p th:if="*{isMarried}">Đã kết hôn</p>
    <p th:unless="*{isMarried}">Chưa kết hôn</p>
</div>
```

Sử dụng thymleaf:

4- th:switch and th:case

- **Controller:** Thêm model.addAttribute

model.addAttribute("attributeName", object);

```
@GetMapping("/person")
public String index(Model model)
{
    model.addAttribute("title", "Demo Thymeleaf");
    model.addAttribute("info", "<b>Thông tin
</b></br>");
    Person p = new Person();
    p.setName("Thymeleaf Spring Boot");
    p.setAge(36);
    p.setSex(1);
    p.setMarried(true);
    p.setCreateTime(LocalDate.now());

    p.setLanguage(Arrays.asList("Java", "C#", "Python"));
    model.addAttribute("person", p);
    return "person";
}
```

- **C1 Thymeleaf:** \${attributeName.fields}

```
<div th:switch="${person.sex}">
    <p th:case="1">Nam</p>
    <p th:case="2">Nữ</p>
    <p th:case="*">Chưa xác định</p>
</div>
```

- **C2 Thymeleaf:** *{fields}

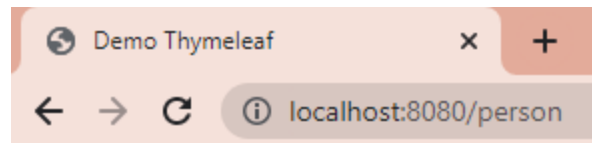
```
<div th:switch="${sex}">
    <p th:case="1">Nam</p>
    <p th:case="2">Nữ</p>
    <p th:case="*">Chưa xác định</p>
</div>
```

Sử dụng thymleaf:

5- th:href

- **Thymeleaf:**

```
<a th:href="@{https://https://www.thymeleaf.org}">Thymeleaf</a>
```



Thông tin

Thymeleaf

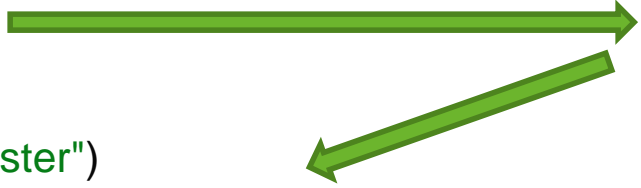
Thymeleaf Spring Boot

6- Forms: th:action, th:method

- Controller:

```
@GetMapping("/person/register")
public String register()
{
    return "register";
}

@PostMapping("/person/register")
public String register(Model model, @ModelAttribute Person person)
{
    model.addAttribute("person", person);
    return "thankyou";
}
```



- Thymeleaf:

```
<form th:action = "@{/person/register}" th:method="post">
    <input type="text" name ="name">
    <input type="text" name ="age">
    <input type="submit">
</form>
```

6- Forms: th:object, th:field

- Controller:

```
@GetMapping("/person/edit")
public String edit(Model model)
{
    Person p = new Person();
    p.setName("Thymeleaf Spring Boot");
    p.setAge(36);
    model.addAttribute("person", p);
    return "edit";
}

@PostMapping("/person/edit")
public String edit(Model model, @ModelAttribute Person person)
{
    model.addAttribute("person", person);
    return "thankyou";
}
```

- Thymeleaf:

```
<form th:action = "@{/person/edit}" th:method="post"
th:object="${person}">
    <input type="text" th:field = "${name}">
    <input type="text" th:field = "${age}">
    <input type="submit">
</form>
```

Layout page – content Page

- Sử dụng **thymeleaf-layout-dialect**

```
<dependency>  
<groupId>nz.net.ultraq.thymeleaf</groupId>  
<artifactId>thymeleaf-layout-dialect</artifactId>  
</dependency>
```

- Để sử dụng từ những template khác (như các footer, các header, các menu...)

-> Thymeleaf cần chúng ta định ra các phần đó, được gọi là các “**fragment**” -> sử dụng thuộc tính **th:fragment**.

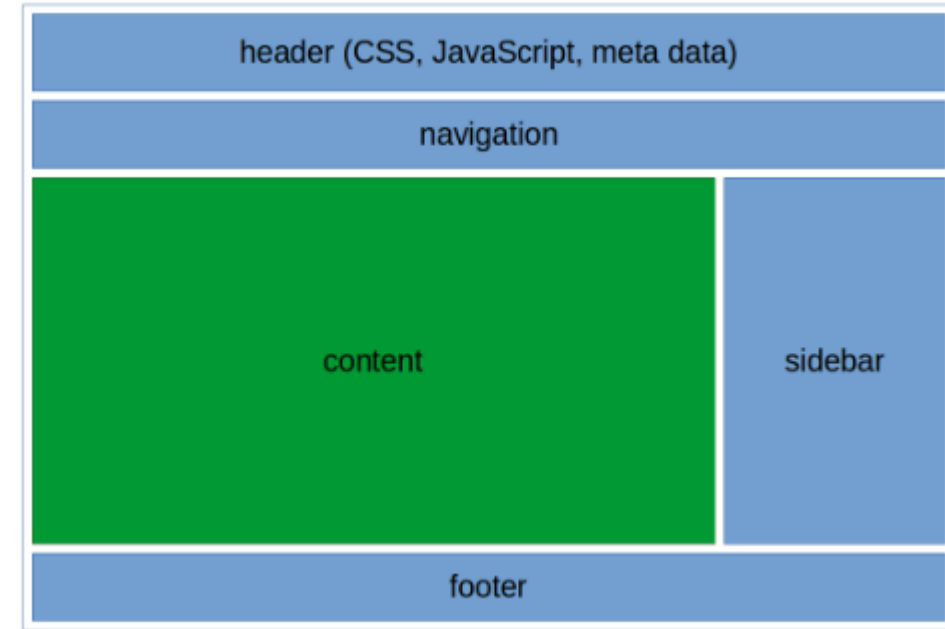
- Sử dụng scripts trong Layout-Content Page

- layout page

```
<th:block layout:fragment="script"></th:block>
```

- content page

```
<th:block layout:fragment="script"> <script th:src="@{/js/accounts.js}"></script> </th:block>
```



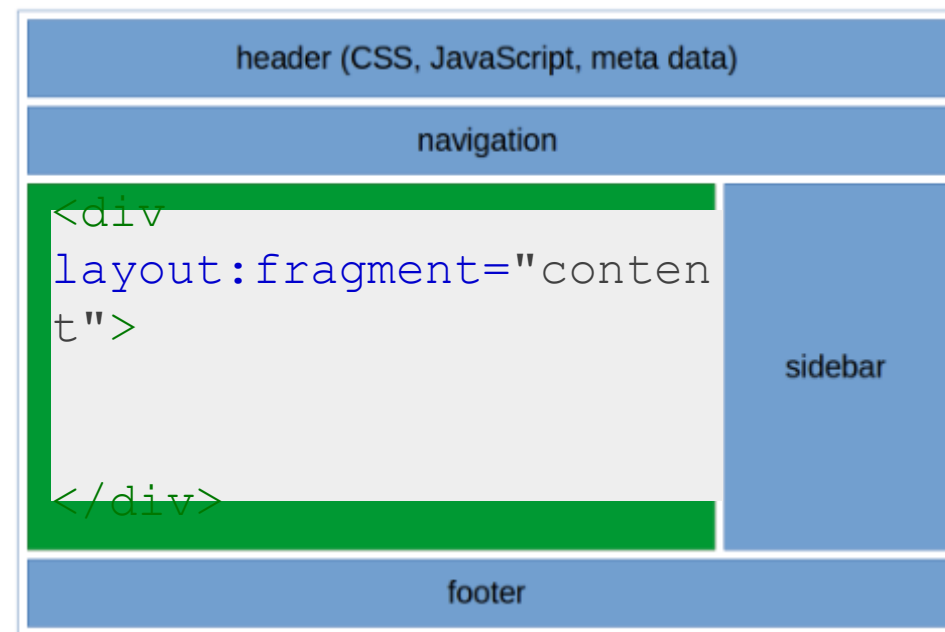
Layout page

- Thêm thuộc tính `xmlns:layout` ở HTML layout page.

```
<html lang="en"
```

```
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
```

- Sử dụng div với `layout:fragment`: chỉ định phần của trang content cho phần nội dung



Content page (fragment)

- Thêm thuộc tính `xmlns:layout` và `layout:decorate`

```
<html lang="en"
```

```
xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
```

```
layout:decorate="_layout">
```

Trong đó: `_layout`: là tên Layout page

- Sử dụng `div` với thuộc tính `layout:fragment`: Nội dung phần content

```
<div layout:fragment="content">
```

```
    <!--Content of the page-->
```

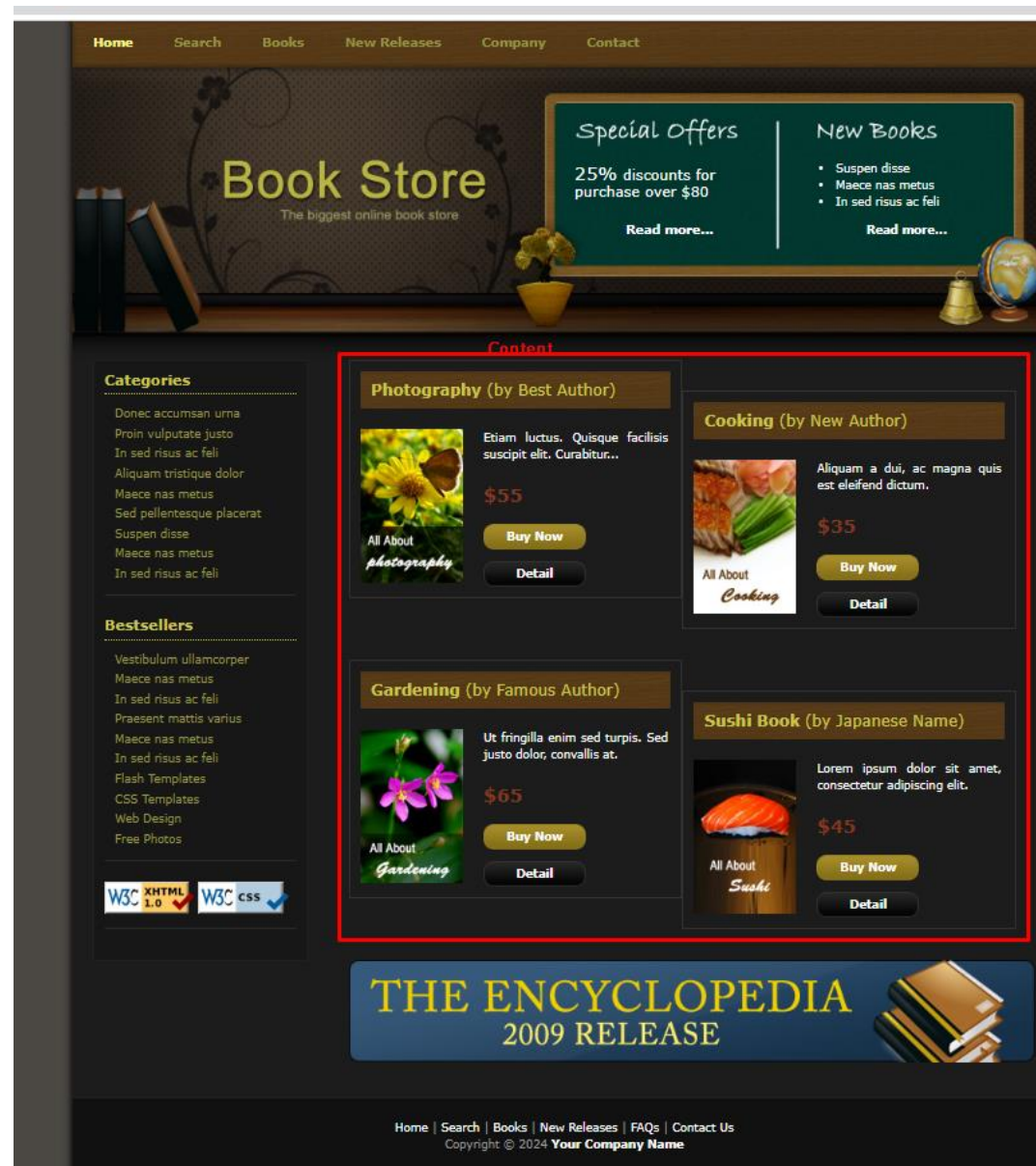
```
</div>
```

Ngoài ra có 2 cách để include content từ fragment :

- ❑ **insert** – chèn 1 fragment đã có vào tag hiện tại
- ❑ **replace** – thay thế cho thẻ hiện tại

Tham khảo: <https://www.baeldung.com/spring-thymeleaf-fragments>

Sử dụng controller + layout + thymeleaf



ASM3: Đăng ký khóa học

- Viết chương trình

1. Đăng ký khóa học
2. Các khóa học sắp tới
(có thời gian bắt đầu > thời gian hiện tại)
3. Điều chỉnh lại layout cho khóa học

Mỗi khóa học **Course**

```
public class Course {  
    3 usages  
    private int id;  
    3 usages  
    private String lectureName;  
    3 usages  
    private String place;  
    3 usages  
    private LocalDate startDate;  
}
```

Home Add Course

Place

Start Date:

Lecture:

Add

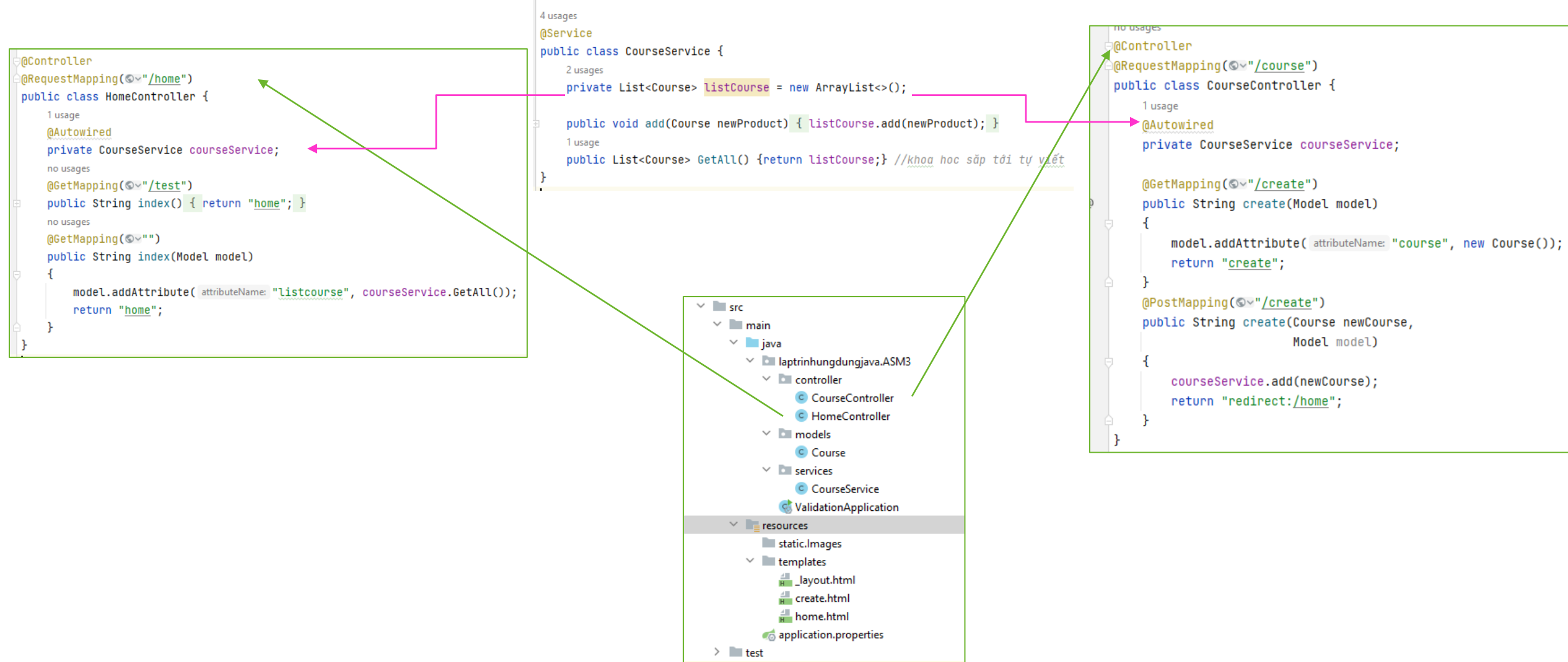
Lập trình ứng dụng Java - Nguyễn Huy Cường- @2023

Home Add Course

2023-06-02 - NGUYỄN HUY CƯỜNG, Hồ Chí Minh
2023-07-06 - NGUYỄN HUY CƯỜNG, BÌNH DƯƠNG

Lập trình ứng dụng Java - Nguyễn Huy Cường- @2023

Hướng dẫn ASM3 – xây dựng controller



Hướng dẫn ASM3 – xây dựng layout

```
create.html
1 <!DOCTYPE html>
2 <html lang="en"
3   xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
4   layout:decorate="_layout">
5 </html>
6 <head>
7   <meta charset="UTF-8">
8   <title>Create product</title>
9 </head>
10 <body>
11   <div layout:fragment="content" class="container body-content">
12     <form th:action="@{/course/create}" th:object="${course}" method="post" class="form">
13       <div class="form-group">
14         <label for="name">Place</label>
15         <input class="form-control" type="text" th:field="*{place}" id="name" placeholder="Place">
16       </div>
17       <div class="form-group">
18         <label for="image">Start Date</label>
19         <input class="form-control" type="date" th:field="*{startDate}" id="image" placeholder="image">
20       </div>
21       <div class="form-group">
22         <label for="lectureName">Lecture</label>
23         <input class="form-control" type="text" th:field="*{lectureName}" id="lectureName" placeholder="LectureName">
24       </div>
25       <input type="submit" class="btn btn-success" value="Add">
26     </form>
27   </div>
28 </body>
29 </html>
```

```
_layout.html
1 <!DOCTYPE html>
2 <html lang="en"
3   xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout">
4 </html>
5 <head>
6   <meta charset="UTF-8">
7   <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha3/dist/css/bootstrap.min.css"
8     rel="stylesheet"
9     integrity="sha384-KK94CHFLLe+nY2dmCWGMq91rCGa5gtU4mk92HdvYe+M/SXH301p5ILy+dN9+nJ0Z"
10    crossorigin="anonymous">
11 </head>
12 <body>
13   <nav class="navbar bg-dark" data-bs-theme="dark">
14     <div>
15       <a class="navbar-brand" th:href="@{/home}">Home</a>
16       <a class="navbar-brand" th:href="@{/course/create}">Add Course</a>
17     </div>
18   </nav>
19   <div layout:fragment="content" class="container body-content">
20     <div class="card-footer">
21       <hr />
22       Lập trình ứng dụng Java - Nguyễn Huy Cường- @2023
23     </div>
24   </div>
25 </body>
26 </html>
```

```
home.html
1 <!DOCTYPE html>
2 <html lang="en"
3   xmlns:layout="http://www.ultraq.net.nz/thymeleaf/layout"
4   layout:decorate="_layout">
5 </html>
6 <head>
7   <meta charset="UTF-8">
8   <title>Home</title>
9 </head>
10 <body>
11   <div layout:fragment="content" class="row">
12     <ul>
13       <li th:each="course : ${listcourse}" th:text="${course.getStartDate()} + ' - '
14         + ${course.getLectureName()} + ', '
15         + ${course.getPlace()}></li>
16     </ul>
17   </div>
18 </body>
19 </html>
```

Điều chỉnh lại giao diện cho trang home

