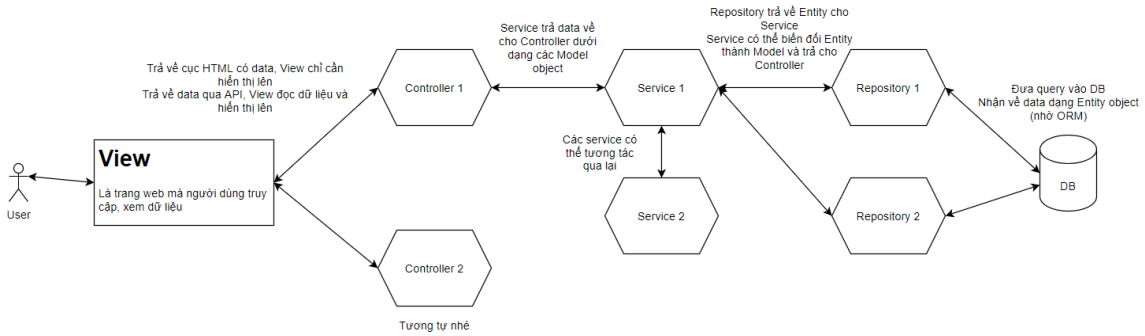
1.Luồng đi trong Spring Boot

Cấu trúc source code của Spring Boot được dựa trên hai mô hình là mô hình MVC và mô hình 3 lớp.

- Controller: trả về View (có chứa data sẵn, dạng trang HTML), hoặc Model thể hiện dưới dạng API cho View (View viết riêng bằng React, Vue, hoặc Angular).
- Service: chứa các code tính toán, xử lý. Khi Controller yêu cầu, thì Service tương ứng sẽ tiếp nhận và cho ra dữ liệu trả cho Controller (trả về Model). Controller sẽ gửi về View như trên.
- Repository: Service còn có thể tương tác với service khác, hoặc dùng Repository để gọi DB. Repository là thằng trực tiếp tương tác, đọc ghi dữ liệu trong DB và trả cho service.

Sơ đồ luồng đi:



2. Dependency injection

Module coupling

Coupling có thể hiểu là mối quan hệ giữa hai module, hai đối tượng với nhau, có sự phụ thuộc lẫn nhau.

- Tight coupling: hai module liên kết chặt chẽ, khó tách rời
- Loose coupling: liên kết yếu, rời rạc

Nguyên tắc về sự phụ thuộc

Để code dễ bảo trì và sửa đổi, thì nguyên tắc là phải giảm sự phụ thuộc giữa các module.

Nghĩa là biến mối quan hệ giữa chúng từ tight coupling thành loose coupling.

3. Nguyên lý Dependency inversion

Dependency inversion principle

DI principle có hai ý chính:

- Các module cấp cao không nên phụ thuộc (trực tiếp) vào module cấp thấp. Cả hai nên phụ thuộc vào abstraction (của OOP).
- Abstraction không nên phụ thuộc vào chi tiết, mà ngược lại.

4. Inversion of Control (IoC)

IoC nhằm mục đích đơn giản hóa quá trình tạo đối tượng và liên kết giữa chúng, bằng cách tuân theo nguyên tắc: Không tạo đối tượng, chỉ mô tả cách chúng sẽ được tạo ra.
IoC framework sẽ có các thành phần có sẵn làm nhiệm vụ tạo, quản lý các đối tượng trong chương trình. IoC sẽ quản lý, phân tích các mối phụ thuộc,

tạo các đối tượng theo thứ tự phù hợp nhất và liên kết chúng lại với nhau, theo cách lập trình viên mô tả.
với IoC framework, chúng ta chỉ cần đánh dấu (mark) trên các class. IoC framework sẽ dựa vào đó để tạo module đúng theo yêu cầu.

Mỗi class được đánh dấu `@Component` (cái này gọi là Annotation trong java) sẽ được IoC hiểu là một module:

- `@Component` là bảo IoC container tạo một object duy nhất (singleton)
- `@Autowired` là tìm module tương ứng (tạo từ trước) và inject vào đó.

5. Dependency injection

DI là một dạng thực hiện của IoC, bằng cách tiêm (inject) module vào một module khác cần nó. Mọi module trong IoC đều gọi là dependency, mặc dù có những module không bị phụ thuộc bởi module nào khác. Khi chương trình chạy, IoC sẽ quét tất cả class đánh dấu dependency, tạo một đối tượng duy nhất (singleton), và bỏ vào cái túi gọi là IoC container, lúc nào cần thì lấy ra sử dụng.
Nếu khi tạo module nào đó, mà module đó cần một module khác phụ thuộc, thì IoC sẽ tìm trong IoC container xem có không, nếu có thì inject vào, nếu chưa thì tạo mới, bỏ vào container và inject vào. Việc inject tự động các dependency (module) như thế được gọi là Dependency injection.

Các loại injection

- Constructor-based injection: Dùng inject các module bắt buộc. Các module được inject nằm trong constructor, và được gán lần lượt vào các field.
- Setter-based injection: Dùng inject các module tùy chọn. Mỗi module sẽ được inject thông qua setter, nằm ở tham số và cũng gán cho field nào đó.

6. Áp dụng vào Spring Boot

Spring là một framework được xây dựng dựa trên nguyên lý Dependency injection. Bản thân Spring có chứa IoC container, có nhiệm vụ tạo và quản lý các module:

- IoC container của Spring gọi là Application context
- Các module chứa trong IoC container được Spring gọi là các Bean

Spring Boot sử dụng các annotation dạng như `@Component` để đánh dấu lên class, chỉ ra rằng class đó cần tạo một module. Ngoài `@Component`, còn có các annotation khác như `@Repository`, `@Controlller`, `@Service`,... cũng được đánh dấu là module.

Khi ứng dụng Spring Boot chạy, thì IoC container sẽ thực hiện quá trình như sau:

- Quét tìm (scan) các class được đánh dấu là Bean, và tạo một object singleton, bỏ vào IoC container
- Khi có một Bean phụ thuộc vào Bean khác, thì IoC sẽ tìm trong container, nếu chưa có thì tạo, nếu đã có thì lấy ra và inject vào bean cần nó

7. Bean và ApplicationContext

Bean

Bean là những module chính của chương trình, được tạo ra và quản lý bởi Spring IoC container.

Các bean có thể phụ thuộc lẫn nhau. Sự phụ thuộc này được mô tả cho IoC biết nhờ cơ chế Dependency injection.

ApplicationContext

Là khái niệm Spring Boot dùng để chỉ Spring IoC container, tương tự như bean là đại diện cho các dependency.

Cách lấy bean ra từ Context

Method `SpringApplication.run()` sẽ return về một object `ApplicationContext` interface, đại diện cho IoC container.
Hoặc dùng dùng method `getBean()`

Kĩ thuật inject bean vào bean khác

Sử dụng `@Autowired`

Sử dụng annotation `@Autowired` để báo cho Spring biết tự động tìm và inject bean phù hợp vào vị trí đặt annotation.

Cách dùng `@Autowired` trên field là không được khuyến khích, do nó sử dụng Java reflection để inject. Chúng ta nên cân nhắc đổi qua dùng inject theo kiểu constructor hoặc setter.

Inject qua constructor hoặc setter

Code inject theo kiểu constructor-based nên dùng khi các module là bắt buộc. Khi đó Spring Boot khi tạo bean (cũng chỉ là tạo object, gọi constructor thôi) thì sẽ đưa các phụ thuộc vào constructor khi gọi.

Hoặc dùng kiểu setter-based. Cách dùng setter để inject thường dùng trong trường hợp phụ thuộc vòng, module A phụ thuộc vào B và ngược lại. Do đó, nếu cả hai đều sử dụng constructor based injection thì Spring Boot sẽ không biết nên tạo bean nào trước. Vì thế, giải pháp là một bean sẽ dùng constructor, một bean dùng setter như trên.

Khi Spring Boot không biết chọn bean nào

Có hai cách giải quyết vấn đề này. Thứ nhất là dùng `@Primary` đánh dấu lên một bean. Khi đó bean này sẽ được ưu tiên chọn hơn, trong trường hợp có nhiều bean phù hợp trong context.

Cách 2 là chỉ định rõ tên bean (tên class) cụ thể được inject bằng `@Qualifier`.
Đối với constructor hay setter based cũng tương tự, chỉ cần có `@Qualifier` trước tên field cần inject vào là được.

8. Vòng đời, cơ chế Component scan

Vòng đời của bean

`@PostConstructor` và `@PreDestroy`

- `@PostConstruct` dùng để thực hiện một số task khi khởi tạo bean
- `@PreDestroy` thực hiện các task để dọn dẹp bean sau khi dùng xong

Chúng ta dùng hai annotation trên đánh dấu lên method nào đó, method đó sẽ được tự động gọi khi sự kiện bean xảy ra.

Cách định nghĩa bean

Dùng `@Bean` bên trong `@Configuration`

Cách này dùng cho trường hợp bean cần thực hiện nhiều thao tác phức tạp để khởi tạo, hoặc có nhiều bean liên quan với nhau. Do đó, thay vì khởi tạo riêng rẽ từng class là từng bean, thì gom chung các bean cần khởi tạo lại bỏ vào class chứa là `@Configuration`.

Thường thì các class đánh dấu `@Configuration` có hậu tố là Config.

Khi Spring tìm thấy class `@Configuration`, nó sẽ tạo bean của class này trước (do `@Configuration` cũng là `@Component`). Trong khi tạo thì các logic khởi tạo cũng được thực thi, để chuẩn bị sẵn sàng tạo các `@Bean` bên trong.

Sau đó Spring Boot sẽ tìm các method được đánh dấu `@Bean` bên trong `@Configuration` để tạo bean. Thường thì các bean dạng này ngắn và return ngay object chứ không phải để Spring Boot tạo ra.

Các bean cũng được đưa vào ApplicationContext như bình thường.

Component scan

Cách component scan hoạt động

Khi ứng dụng Spring Boot bắt đầu chạy, thì nó sẽ tìm hết các class đánh dấu là bean trong chương trình và tạo bean. Quá trình tìm kiếm các bean này gọi là component scan.

Do đó, class đánh dấu `@SpringBootApplication` có chứa main method sẽ là nơi bắt đầu. Spring Boot sẽ tìm từ package này (package gốc) tìm xuống để tạo các bean.

9. Cấu trúc dự án Spring Boot thế nào cho chuẩn?

Cấu trúc của ứng dụng

Cấu trúc chung của ứng dụng

- Thư mục gốc chứa các file linh tinh như `pom.xml` (của Maven), `build.gradle` và các file khác như `.gitignore`,... dùng để cấu hình dự án.
- Thư mục `.mvn` hoặc `.gradle` là thư mục riêng của Maven và Gradle, đừng nên đụng tới hay exclude nó ra khỏi source code.
- Code được chứa trong thư mục `src`.
- Thư mục build ra chứa các file class, file JAR. Với Maven là `target` còn Gradle là `build`.

Chi tiết cấu trúc source code

- Tên package chính được đặt dạng ngược với tên miền. Ví dụ như `tonghoangvu.com` thì đặt thành `com.tonghoangvu`. Cộng thêm tên project nữa.
- Có các package con, mỗi package đại diện cho các class thuộc layer cụ thể (ví dụ như `service`, `controller`,...)
- Thư mục `resources` chứa các tài nguyên của ứng dụng như hình ảnh, static file, properties file,...

Ngoài ra còn có `src/test` dùng để chứa các test class, dùng cho unit test.

Tổ chức source code theo mô hình 3 lớp

Tương ứng với từng thành phần của mô hình 3 lớp, thì chúng ta sẽ có các thư mục và cách đặt tên tương ứng:

- Controller layer: đặt trong `controller`, các class là controller sẽ có hậu tố Controller (ví dụ `UserController`, `AuthController`,...)
- Service layer: đặt trong `service`, các class có hậu tố là Service và thường tương ứng với controller (ví dụ `UserService`,...)
- Data access layer: ca này khó, bởi vì layer này bao gồm repository (đặt trong `repository` và hậu tố tương tự), DTO, model, entity... chi tiết mình sẽ nói ở các bài sau.

Ngoài ra, với các loại khác thì:

- `util` package chứa các lớp util (xử lý linh tinh), ví dụ như convert end date, tính toán đơn giản,...
- `common` package chứa các class định nghĩa như enum, interface, class dùng chung và đơn giản
- `exception` package chứa các class có nhiệm vụ xử lý exception trong Spring Boot.

- `component` chứa các bean được định nghĩa còn lại nhưng không thuộc layer nào.

10. Entity, domain model và DTO

Các dạng data

- DTO (Data transfer object): là các class đóng gói data để chuyển giữa client - server hoặc giữa các service trong microservice. Mục đích tạo ra DTO là để giảm bớt lượng info không cần thiết phải chuyển đi, và cũng tăng cường độ bảo mật.
- Domain model: là các class đại diện cho các domain, hiểu là các đối tượng thuộc business như Client, Report, Department,... chẳng hạn. Trong ứng dụng thực, các class đại diện cho kết quả tính toán, các class làm tham số đầu vào cho service tính toán,... được coi là domain model.
- Entity: cũng là domain model nhưng tương ứng với table trong DB, có thể map vào DB được. Lưu ý chỉ có entity mới có thể đại diện cho data trong DB.

Áp dụng vào mô hình 3 lớp trong sơ đồ, thì chúng ta rút ra được nguyên tắc thiết kế chung:

- Web layer: chỉ nên xử lý DTO, đồng nghĩa với việc các Controller chỉ nên nhận và trả về dữ liệu là DTO.
- Service layer: nhận vào DTO (từ controller gửi qua) hoặc Domain model (từ các service nội bộ khác). Dữ liệu được xử lý (có thể tương tác với DB), cuối cùng được Service trả về Web layer dưới dạng DTO.
- Repository layer: chỉ thao tác trên Entity, vì đó là đối tượng thích hợp, có thể mapping vào DB.

Đối với các thành phần khác của Spring Boot mà không thuộc layer nào, thì:

- Custom Repository: đây là layer không thông qua repository mà thao tác trực tiếp với database. Do đó, lớp này được hành xử như Service.

11. Spring Boot Application Config và @Value

application.properties

Trong Spring Boot, các thông tin cấu hình mặc định được lấy từ file `resources/applications.properties`.

Ví dụ, bạn muốn Spring Boot chạy trên port 8081 thay vì 8080:

applicatoin.properties

```
server.port = 8081
```

Hoặc bạn muốn log của chương trình chi tiết hơn. Hãy chuyển nó sang dạng Debug bằng cách config như sau:

```
logging.level.root=DEBUG
```

Đây là cách chúng ta có thể can thiệp vào các cấu hình của ứng dụng từ bên ngoài. Cho phép thay đổi linh hoạt tùy môi trường.

@Value

Trong trường hợp, bạn muốn tự config những giá trị của riêng mình, thì Spring Boot hỗ trợ bạn với annotation `@Value`

Thông tin truyền vào annottaion `@Value` chính là tên của cấu hình đặt trong dấu `${name}`

12. Model & View Trong Spring Boot

`Model` là đối tượng lưu giữ thông tin và được sử dụng bởi Template Engine để generate ra webpage. Có thể hiểu nó là `Context` của Thymeleaf

`Model` lưu giữ thông tin dưới dạng key-value.

Trong template thymeleaf, để lấy các thông tin trong `Model`. bạn sẽ sử dụng `Thymeleaf Standard Expression`.

1. `${...}`: Giá trị của một biến.
2. `{...}`: Giá trị của một biến được chỉ định (sẽ giải thích ở dưới)

Ngoài ra, để lấy thông tin đặc biệt hơn:

1. `#{...}`: Lấy message
2. `@{...}`: Lấy đường dẫn URL dựa theo context của server

13. @RequestMapping + @PostMapping + @ModelAttribute + @RequestParam + Web To-Do với Thymeleaf

@PostMapping

`@PostMapping` có nhiệm vụ đánh dấu hàm xử lý POST request trong Controller.

@RequestMapping

Trong trường hợp bạn muốn tất cả các method đều dùng chung một cách xử lý thì có thể sử dụng Annotation `@RequestMapping`.

`@RequestMapping` là một annotation có ý nghĩa và mục đích sử dụng rộng hơn các loại `@GetMapping`, `@PostMapping`,v.v..

14. Spring Boot JPA + MySQL

Spring Boot JPA

Spring Boot JPA là một phần trong hệ sinh thái Spring Data, nó tạo ra một layer ở giữa tầng service và database, giúp chúng ta thao tác với database một cách dễ dàng hơn, tự động config và giảm thiểu code thừa thãi.

JpaRepository

Để sử dụng Spring JPA, bạn cần sử dụng interface `JpaRepository`.

Yêu cầu của interface này đó là bạn phải cung cấp 2 thông tin:

1. Entity (Đối tượng tương ứng với Table trong DB)
2. Kiểu dữ liệu của khóa chính (primary key)

`@Repository` đánh dấu `UserRepository` là một Bean và chịu trách nhiệm giao tiếp với DB.

Spring Boot sẽ tự tìm thấy và khởi tạo ra đối tượng `UserRepository` trong Context. Việc tạo ra `UserRepository` hoàn toàn tự động và tự config, vì chúng ta đã kế thừa `JpaRepository`.

15. Spring JPA Method + @Query

Query Creation

Trong Spring JPA, có một cơ chế giúp chúng ta tạo ra các câu Query mà không cần viết thêm code.

Cơ chế này xây dựng Query từ tên của method.

Thì Spring JPA sẽ tự định nghĩa câu Query cho method này, bằng cách xử lý tên method. Vậy là chúng ta đã có thể truy vấn dữ liệu mà chỉ mất thêm 1 dòng code.

Quy tắc đặt tên method trong Spring JPA

Trong Spring JPA, cơ chế xây dựng truy vấn thông qua tên của method được quy định bởi các tiền tố sau:

`find…By`, `read…By`, `query…By`, `count…By`, và `get…By`.

phần còn lại sẽ được parse theo tên của thuộc tính (viết hoa chữ cái đầu). Nếu chúng ta có nhiều điều kiện, thì các thuộc tính có thể kết hợp với nhau bằng chữ `And` hoặc `Or`.

@Query

Spring JPA còn hỗ trợ chúng ta một cách nguyên thủy khác.

Với cách sử dụng `@Query`, bạn sẽ có thể sử dụng câu truy vấn JPQL (Hibernate) hoặc raw SQL.

16. Restful API + @RestController + @PathVariable + @RequestBody

@RestController

Khác với @Controller là sẽ trả về một template.

@RestController trả về dữ liệu dưới dạng JSON.

Các đối tượng trả về dưới dạng Object sẽ được Spring Boot chuyển thành JSON.

Các đối tượng trả về rất đa dạng, bạn có thể trả về `List`, `Map`, v.v.. Spring Boot sẽ convert hết chúng thành JSON, mặc định sẽ dùng Jackson converter để làm điều đó.

Nếu bạn muốn API tùy biến được kiểu dữ liệu trả về, bạn có thể trả về đối tượng `ResponseEntity` của Spring cung cấp. Đây là đối tượng cha của mọi response và sẽ wrapper các object trả về. C

@RequestBody

Vì bây giờ chúng ta xây dựng API, nên các thông tin từ phía Client gửi lên Server sẽ nằm trong `Body`, và cũng dưới dạng `JSON` luôn.

Tất nhiên là Spring Boot sẽ làm giúp chúng ta các phần nặng nhọc, nó chuyển chuỗi JSON trong request thành một Object Java. bạn chỉ cần cho nó biết cần chuyển JSON thành Object nào bằng Annotation `@RequestBody`

@PathVariable

`RESTful API` là một tiêu chuẩn dùng trong việc thiết kế các thiết kế API cho các ứng dụng web để quản lý các resource.

Và với cách thống nhất này, thì sẽ có một phần thông tin quan trọng sẽ nằm ngay trong chính URL của api.

17. Exception Handling @ExceptionHandler + @RestControllerAdvice / @ControllerAdvice + @ResponseStatus

@RestControllerAdvice & @ControllerAdvice + @ExceptionHandler

`@RestControllerAdvice` là một Annotation gắn trên Class. Có tác dụng xen vào quá trình xử lý của các `@RestController`. Tương tự với `@ControllerAdvice`

`@RestControllerAdvice` thường được kết hợp với `@ExceptionHandler` để cắt ngang quá trình xử lý của Controller, và xử lý các ngoại lệ xảy ra.

Controller đang hoạt động bình thường, chẳng may có một Exception được ném ra, thì thay vì báo lỗi hệ thống, thì nó sẽ được thằng `@RestControllerAdvice` và `@ExceptionHandler` đón lấy và xử lý. Sau đó trả về cho người dùng thông tin hữu ích hơn.

@ResponseStatus

`@ResponseStatus` là một cách định nghĩa Http Status trả về cho người dùng.

Nếu không muốn sử dụng `ResponseEntity` thì có thể dùng `@ResponseStatus` đánh dấu trên `Object` trả về.

18. @ConfigurationPropertie

Chúng ta sử dụng `@Component` để Spring biết đây là một bean và khởi tạo nó.

Sử dụng `@PropertySource` để định nghĩa tên của file config. Nếu không có annotation này, Spring sẽ sử dụng file mặc định (*classpath:application.yml* trong thư mục *resources*)

Cuối cùng là `@ConfigurationProperties`, annotation này đánh dấu class bên dưới nó là properties, các thuộc tính sẽ được tự động nạp vào khi Spring khởi tạo.

Lưu ý: các thuộc tính này được xác định bởi `prefix=loda`. Cái này bạn xem file *application.yml* ở dưới sẽ hiểu.

Spring sẽ tự tìm các hàm setter để set giá trị cho các thuộc tính này, nên quan trọng là bạn phải tạo ra các setter method. (Ở đây tôi nhường việc đó cho lombok).

Ngoài ra, để chạy được tính năng này, bạn cần kích hoạt nó bằng cách gắn `@EnableConfigurationProperties` lên một configuration nào đó.

19. Tạo Bean có điều kiện với @Conditional

@ConditionalOnBean

`@ConditionalOnBean` sử dụng khi chúng ta muốn tạo ra một Bean, chỉ khi có một Bean khác đang tồn tại

@ConditionalOnProperty

Dùng `@ConditionalOnProperty` khi bạn muốn quyết định sự tồn tại Bean thông qua cấu hình property.

@ConditionalOnExpression

Trong trường hợp bạn muốn thỏa mãn nhiều điều kiện trong property, hãy sử dụng `@ConditionalOnExpression`

@ConditionalOnMissingBean

Nếu trong `Context` chưa tồn tại bất kỳ Bean nào tương tự, thì `@ConditionalOnMissingBean` sẽ thỏa mãn điều kiện và tạo ra một Bean như thế.

@ConditionalOnResource

`@ConditionalOnResource` thỏa mãn khi có một resources nào đó do bạn chỉ định tồn tại

@ConditionalOnClass

`@ConditionalOnClass` thỏa mãn khi trong classpath có tồn tại class mà bạn yêu cầu

@ConditionalOnMissingClass

`@ConditionalOnMissingClass` ngược lại với `@ConditionalOnClass`. thỏa mãn khi trong classpath không tồn tại class mà bạn yêu cầu

@ConditionalOnJava

`@ConditionalOnJava` thỏa mãn nếu môi trường chạy version Java đúng với điều kiện

20. Tự tạo custom @Conditional

Tự tạo @Conditional

Để tạo ra một điều kiện cho mình, bạn cần kế thừa lớp `Condition`, và implement lại function `matches`

`matches` là nơi bạn kiểm tra điều kiện xem có thoả mãn không.

Tự tạo Annotation @Conditional

Nếu việc viết `@Conditional(WindowRequired.class)` chưa làm bạn hài lòng, bạn có thể tự tạo ra một `Annotation` giống với Spring Boot.

Ví dụ như `@ConditionalOnClass`

Kết hợp nhiều điều kiện với OR

Bạn có thể kết hợp nhiều điều kiện với nhau bởi phép OR.

Spring Boot hỗ trợ điều này bằng cách kế thừa lớp `AnyNestedCondition`

Kết hợp nhiều điều kiện với AND

Bạn có thể kết hợp các điều kiện bằng phép AND bằng cách kế thừa lớp `AllNestedConditions`.

Cách kế thừa của nó giống với `AnyNestedCondition` nên tôi sẽ không cần đề cập thêm.

Ngoài ra, có một cách khác là sử dụng nhiều custom `@Conditional` cùng một lúc.

21. Xử lý sự kiện với @EventListener + @Async

Event

Một `Event` (sự kiện) muốn được Spring Boot hỗ trợ thì sẽ phải kế thừa lớp `ApplicationEvent`.

Event Publisher

Trong Spring Boot, để bắn ra một sự kiện chúng ta sử dụng đối tượng `ApplicationEventPublisher`. Đây là một `Bean` có sẵn trong `Context` do Spring cung cấp, bạn chỉ cần lôi ra sử dụng thôi.

Event Listener

Để lắng nghe các sự kiện do `ApplicationEventPublisher` bắn ra, chúng ta sử dụng `@EventListener`

`@EventListener` gắn trên một method, với tham số đầu vào chính là sự kiện mà bạn muốn lắng nghe.

Lưu ý: Class chịu trách nhiệm xử lý, có chứa method `@EventListener` cũng phải là Bean nhé.