**Giới thiệu**

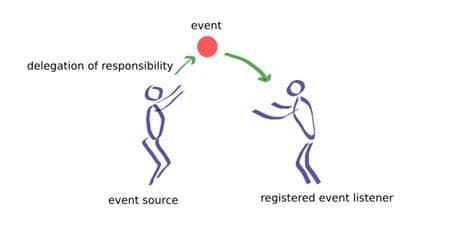
Hẳn trong **Java** không ít thì nhiều các bạn đã từng sử dụng qua Listener Pattern rồi, nên tôi nghĩ sẽ không giới thiệu về nó nữa.

Trong bài viết này, chúng ta tập trung vào cách làm sao để thực hiện việc tạo ra Event và xử lý Event đó một cách gọn gàng với **Spring Boot**.

Bài này yêu cầu kiến thức cơ bản:

1. [「Spring Boot #0」 Series làm chủ Spring Boot, Zero to Hero](https://loda.me/spring-boot-0-series-lam-chu-spring-boot-zero-to-hero-loda1558963914472)

**Cơ bản về Event & Listener**



Cơ bản là khi chương trình của bạn đang vận hành, và có một công việc gì đó, bạn không muốn xử lý trực tiếp tại Class hiện hành hoặc muốn thông báo cho các Đối tượng khác biết bạn vừa làm gì.

Thì bạn sẽ bắn ra một object gọi là Event (sự kiện), có hoặc không thông tin đi kèm, và nhiệm vụ của các thằng khác là đón lấy hay lắng nghe sự kiện đó để xử lý nghiệp vụ của riêng nó, thằng xử lý gọi là Listener.

Thằng gây ra sự kiện gọi là Source.

Còn thằng cầm sự kiện đó ném cho Listener gọi là Pushlisher

**Áp vào thực tế**

Giả sử bạn có một cái chuông cửa, khi có người tới bấm cái chuông này. Chuông sẽ phát ra tiếng kêu.

Ở trong nhà có chó, nó nghe thấy tiếng kêu, nó sẽ sủa lên.

Thì:

* Source: Người bấm chuông cửa, là người gây ra sự kiện.
* Event: sự kiện bấm chuông cửa
* Pushlisher: Cái chuông phát ra âm thanh (sự kiện) để thông báo.
* Listener: Con chó lắng nghe và xử lý sự kiện

**Event**

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

/\*

DoorBellEvent phải kế thừa lớp ApplicationEvent của Spring

Như vậy nó mới được coi là một sự kiện hợp lệ.

\*/

public class DoorBellEvent extends ApplicationEvent {

/\*

Mọi Class kế thừa ApplicationEvent sẽ

phải gọi Constructor tới lớp cha.

\*/

public DoorBellEvent(Object source, String guestName) {

// Object source là object tham chiếu tới

// nơi đã phát ra event này!

super(source);

}

}

**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.

@Component

public class MyHouse {

@Autowired

ApplicationEventPublisher applicationEventPublisher;

/\*\*

\* Hành động bấm chuông cửa

\*/

public void rangDoorbellBy(String guestName) {

// Phát ra một sự kiện DoorBellEvent

// source (Nguồn phát ra) chính là class này

applicationEventPublisher.publishEvent(new DoorBellEvent(this, guestName));

}

}

**Event Listener**

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

// Tạo ra Bean

@Component

public class MyDog {

/\*

@EventListener sẽ lắng nghe mọi sự kiện xảy ra

Nếu có một sự kiện DoorBellEvent được bắn ra, nó sẽ đón lấy

và đưa vào hàm để xử lý

\*/

@EventListener

public void doorBellEventListener(DoorBellEvent doorBellEvent) throws InterruptedException {

// Giả sử con chó đang ngủ, 1 giây sau mới dậy

Thread.sleep(1000);

// Sự kiện DoorBellEvent được lắng nghe và xử lý tại đây

System.out.println(Thread.currentThread().getName() + ": Chó ngủ dậy!!!");

System.out.println(String.format("%s: Go go!! Có người tên là %s gõ cửa!!!", Thread.currentThread().getName(), doorBellEvent.getGuestName()));

}

}

@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.

**Chạy thử 1**

@SpringBootApplication

public class App {

@Autowired

MyHouse myHouse;

public static void main(String[] args) {

SpringApplication.run(App.class, args);

}

@Bean

CommandLineRunner run() {

return args -> {

System.out.println(Thread.currentThread().getName() + ": Loda đi tới cửa nhà !!!");

System.out.println(Thread.currentThread().getName() + ": => Loda bấm chuông và khai báo họ tên!");

// gõ cửa

myHouse.rangDoorbellBy("Loda");

System.out.println(Thread.currentThread().getName() +": Loda quay lưng bỏ đi");

};

}

}

OUTPUT:

restartedMain: Loda đi tới cửa nhà !!!

restartedMain: => Loda bấm chuông và khai báo họ tên!

restartedMain: Chó ngủ dậy!!!

restartedMain: Go go!! Có người tên là Loda gõ cửa!!!

restartedMain: Loda quay lưng bỏ đi

Bạn sẽ thấy quá trình xử lý ở đây xảy ra một cách **tuần tự** và đồng bộ (Synchronous).

Vậy là chúng ta có thể hiểu:

nếu không cấu hình gì thêm, ApplicationEvent và @EventListener là Synchronous.

Chương trình sẽ phải chờ sự kiện xử lý xong thì mới được chạy tiếp.

**@Async**

Đa phần, xử lý Synchronous không phải điều mà chúng ta mong đợi, chúng ta muốn việc xử lý sự kiện có thể hoạt động riêng và không ảnh hưởng tới luồng làm việc chính.

Nói cách khác, chúng ta muốn sự kiện được xử lý ở một Thread khác, đây gọi là **bất đồng bộ (Asynchronous)**

Để làm được điều này, chúng ta cần kích hoạt chức năng xử lý bất đồng bộ của **Spring Boot**, bằng cách bổ sung annotation @EnableAsync.

@Configuration

@EnableAsync

public class ListenerConfiguration {

/\*\*

\* Tạo ra Executor cho Async

\* @return

\*/

@Bean

TaskExecutor taskExecutor() {

return new SimpleAsyncTaskExecutor();

}

}

**Spring Boot** khi thấy Annotation này, sẽ kích hoạt cho phép xử lý sự kiện dưới dạng Async

các Event sẽ được gửi vào một Executor (đơn giản nhất là SimpleAsyncTaskExecutor) và chờ được xử lý.

Nếu chưa biết Executor là gì, bạn có thể đọc 2 bài viết sau:

1. [Khái niệm ThreadPool và Executor trong Java](https://loda.me/khai-niem-thread-pool-va-executor-trong-java-loda1554800053212)
2. [ThreadPoolExecutor và nguyên tắc quản lý pool size](https://loda.me/thread-pool-executor-va-nguyen-tac-quan-ly-pool-size-loda1554816034602)

**@Async**

Sau khi kích hoạt tính năng Async, bất kỳ sự kiện nào bạn muốn nó xử lý Async thì hãy đánh dấu nó bởi @Async.

@Component

public class MyDog {

/\*

@EventListener sẽ lắng nghe mọi sự kiện xảy ra

Nếu có một sự kiện DoorBellEvent được bắn ra, nó sẽ đón lấy

và đưa vào hàm để xử lý

\*/

/\*

@Async là cách lắng nghe sự kiện ở một Thread khác, không ảnh hưởng tới

luồng chính

\*/

@Async

@EventListener

public void doorBellEventListener(DoorBellEvent doorBellEvent) throws InterruptedException {

// Giả sử con chó đang ngủ, 1 giây sau mới dậy

Thread.sleep(1000);

// Sự kiện DoorBellEvent được lắng nghe và xử lý tại đây

System.out.println("Chó ngủ dậy!!!");

System.out.println(String.format("Go go!! Có người tên là %s gõ cửa!!!", doorBellEvent.getGuestName()));

}

}

**Chạy thử lần 2**

OUTPUT:

restartedMain: Loda đi tới cửa nhà !!!

restartedMain: => Loda bấm chuông và khai báo họ tên!

restartedMain: Loda quay lưng bỏ đi

SimpleAsyncTaskExecutor-1: Chó ngủ dậy!!!

SimpleAsyncTaskExecutor-1: Go go!! Có người tên là Loda gõ cửa!!!

Lần này quá trình xử lý đã diễn ra đúng như chúng ta mong đợi, người bấm cứ bấm, mà chó kêu cứ kêu, mỗi người một việc, chả ai ảnh hưởng tới ai, chỉ cần biết có sự kiện xảy ra thì phản ứng là được.

**Kết**