**Tổng Quan**

**Reactor Core** là một thự viện Java 8 implement mô hình [**Reactive Programming**](https://loda.me/gioi-thieu-reactive-programming-voi-reactor-loda1556032486705). Nó được xây dựng dựa trên **Reactive Streams Specification** - một tiêu chuẩn để xây dựng ứng dụng Reactive.

Trong bài viết này, chúng ta sẽ đi từng bước nhỏ thông qua Reactor cho đến khi có cái nhìn toàn cảnh cũng như cách thực thi của **Reactor core**.

**Maven Dependencies**

Đây là thư viện của Reactor, chúng ta có thể lấy thư viện mới nhất tại [đây](https://mvnrepository.com/artifact/io.projectreactor/reactor-core)

<dependency><groupId>io.projectreactor</groupId>

<artifactId>reactor-core</artifactId>

<version>3.2.8.RELEASE</version>

</dependency>

**Tạo ra một luồng dữ liệu**

Để có một ứng dựng phản ứng (reactive), điều đầu tiên chúng ta cần phải làm là tạo ra một luồng dữ liệu. Không có dữ liệu này chúng ta sẽ không có bất cứ điều gì để phản ứng, đó là lý do tại sao đây là bước đầu tiên.

Reactor core cung cấp 2 loại dữ liệu cho phép chúng ta thực hiện điều này.

**Flux**

Cách đầu tiên đó là dùng Flux. Flux là một luồng có thể phát ra **0..n** phần tử. Ví dụ tạo đơn giản:

Flux<Integer> just = Flux.just(1,2,3,4);

**Mono**

Cách thứ hai đó là Mono. Mono là một luồng có thể phát ra **0..1** phần tử. Nó hoạt động gần giống hệ như Flux, chỉ là bị giới hạn không quá một phần tử. Ví dụ:

Mono<String> just = Mono.just("atomPtit");

Điều lưu ý rằng cả Flux và Mono đề được triển khai từ interface Publisher. Cả hai đều tuần thủ tiêu chuẩn Reactive, chúng ta có thể sử dụng interface như sau:

Publisher<String> just = Mono.just("foo");

**Subscribe()**

Hãy luôn ghi nhớ rằng: **Không có gì xảy ra cho đến khi subscribe()** .

Trong reactor, khi bạn viết một Publisher, dữ liệu không bắt đầu được bơm vào theo mặc định. Thay vào đó, bạn tạo một mô tả trừu tượng về quy định không đồng bộ của bạn(hỗ trợ tái sử dụng).

Để hiểu rõ luồng hoạt động hãy theo dõi qua ví dụ đơn giản sau.

<dependency>

<groupId>io.projectreactor</groupId>

<artifactId>reactor-core</artifactId>

<version>3.2.8.RELEASE</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.1.3</version>

</dependency>

Chúng ta thêm thư viện logback. Điều này sẽ giúp chúng ta ghi nhật ký đầu ra của quá trình hoạt động reactor từ đó hiểu rõ hơn về luồng dữ liệu.

public class ReactorCode {

public static void main(String[] args) {

List<Integer> elements = new ArrayList<>();

Flux.just(1, 2, 3, 4)

.log()

.subscribe(elements::add);

}

}

// OUTPUT:

/\*

23:02:16.996 [main] DEBUG reactor.util.Loggers$LoggerFactory - Using Slf4j logging framework

23:02:17.014 [main] INFO reactor.Flux.Array.1 - | onSubscribe([Synchronous Fuseable] FluxArray.ArraySubscription)

23:02:17.017 [main] INFO reactor.Flux.Array.1 - | request(unbounded)

23:02:17.018 [main] INFO reactor.Flux.Array.1 - | onNext(1)

23:02:17.018 [main] INFO reactor.Flux.Array.1 - | onNext(2)

23:02:17.018 [main] INFO reactor.Flux.Array.1 - | onNext(3)

23:02:17.018 [main] INFO reactor.Flux.Array.1 - | onNext(4)

23:02:17.019 [main] INFO reactor.Flux.Array.1 - | onComplete()

\*/

Hãy nhìn vào phần output, mọi thứ đều chạy trên main thread. Bây giờ chugn ta đi xem rõ từng dòng thực thi: 1. onSubscribe() - Điều này được gọi thi chúng ra đăng ký (subscriber()) luồng

1. request(unbounded) - Khi chúng ta gọi đăng ký, thì hàm này được chạy ngầm nhằm ý nghĩa tạo đăng ký. Trong trường hợp này chạy mặc định là unbounded (không giới hạn), nghĩa là nó yêu cầu mọi phần tử có sẵn.
2. onNext() - Hàm này được gọi cho mọi phần tử đơn.
3. onComplete() - Hàm này được gọi sau cùng sau khi nhận được phần tử cuối cùng. Trong thực có thể xảy ra các hàm khác như onError(), cái mà có thể được gọi khi xảy ra một exception.

**So sánh với Streams Java 8**

Có vẻ nhiều người vẫn đang nghĩ sự tương đồng với Stream trong Java 8:

List<Integer> collected = Stream.of(1, 2, 3, 4)

.collect(toList());

Sự khác biết cốt lõi là Reactive là một hình **push** (đẩy) , trong khi Stream Java 8 là mô hình **pull** (kéo)

Streams Java 8 là terminal - kéo tất cả dữ liệu và trả về một kết quả. Với Reactive, chúng ta có một luồng vô hạn đến từ một nguồi tài nguyên bên ngoài, với nhiều người subscribe(). Chúng ta cũng có thể làm những việc như kết hợp các luồng, tiều tiết luồng và backpressure.

**Backpressure**

Trong ví dụ trên, người đăng ký nói với Publisher đẩy từng phần tử một. Điều này có thể trở nên quá tải cho người đăng ký phải tiêu thụ hết tất cả tài nguyên của nó.

**Backpressure** đơn giản chỉ là bảo với Publisher gửi cho nó ít dữ liệu hơn để ngăn chặn nó bị quá tải.

Ví dụ dưới đây, chúng ta sẽ yêu cầu chỉ gửi 2 phần từ cùng một lúc bằng cách sử dụng request ():

Flux.just(1, 2, 3, 4)

.log()

.subscribe(new Subscriber<Integer>() {

private Subscription s;

int onNextAmount;

@Override

public void onSubscribe(Subscription s) {

this.s = s;

s.request(2);

}

@Override

public void onNext(Integer integer) {

elements.add(integer);

onNextAmount++;

if (onNextAmount % 2 == 0) {

s.request(2);

}

}

@Override

public void onError(Throwable t) {}

@Override

public void onComplete() {}

});

//OUTPUT

/\*

23:31:15.395 [main] INFO reactor.Flux.Array.1 - | onSubscribe([Synchronous Fuseable] FluxArray.ArraySubscription)

23:31:15.397 [main] INFO reactor.Flux.Array.1 - | request(2)

23:31:15.397 [main] INFO reactor.Flux.Array.1 - | onNext(1)

23:31:15.398 [main] INFO reactor.Flux.Array.1 - | onNext(2)

23:31:15.398 [main] INFO reactor.Flux.Array.1 - | request(2)

23:31:15.398 [main] INFO reactor.Flux.Array.1 - | onNext(3)

23:31:15.398 [main] INFO reactor.Flux.Array.1 - | onNext(4)

23:31:15.398 [main] INFO reactor.Flux.Array.1 - | request(2)

23:31:15.398 [main] INFO reactor.Flux.Array.1 - | onComplete()

\*/

Bây giờ chúng ta nhìn thấy hàm request() được gọi trước, tiếp theo đó là 2 hàm onNext() thực hiện, sau đó lại là request().

**Concurrency**

Tất cả các ví dụ trên chúng ta đều đang chạy trên một luồng chính. Tuy nhiên, chúng ta có thể kiểm soát luồng nào mà code của chúng ta chạy nếu chúng ta muốn. Các inteface Scheduler cung cấp một sự trừu tượng với asynchronous.

public class ReactorCode {

public static void main(String[] args) {

ExecutorService service = Executors.newFixedThreadPool(10);

Flux.just(1, 2, 3, 4)

.log()

.subscribeOn(Schedulers.fromExecutorService(service))

.subscribe();

Flux.just(5, 6, 7, 8)

.log()

.subscribeOn(Schedulers.fromExecutorService(service))

.subscribe();

}

}

//OUTPUT

/\*

23:48:02.972 [main] DEBUG reactor.util.Loggers$LoggerFactory - Using Slf4j logging framework

23:48:02.996 [pool-1-thread-2] INFO reactor.Flux.Array.2 - | onSubscribe([Synchronous Fuseable] FluxArray.ArraySubscription)

23:48:02.996 [pool-1-thread-1] INFO reactor.Flux.Array.1 - | onSubscribe([Synchronous Fuseable] FluxArray.ArraySubscription)

23:48:03.000 [pool-1-thread-2] INFO reactor.Flux.Array.2 - | request(unbounded)

23:48:03.000 [pool-1-thread-1] INFO reactor.Flux.Array.1 - | request(unbounded)

23:48:03.001 [pool-1-thread-1] INFO reactor.Flux.Array.1 - | onNext(1)

23:48:03.001 [pool-1-thread-2] INFO reactor.Flux.Array.2 - | onNext(5)

23:48:03.001 [pool-1-thread-1] INFO reactor.Flux.Array.1 - | onNext(2)

23:48:03.001 [pool-1-thread-2] INFO reactor.Flux.Array.2 - | onNext(6)

23:48:03.001 [pool-1-thread-1] INFO reactor.Flux.Array.1 - | onNext(3)

23:48:03.001 [pool-1-thread-2] INFO reactor.Flux.Array.2 - | onNext(7)

23:48:03.001 [pool-1-thread-1] INFO reactor.Flux.Array.1 - | onNext(4)

23:48:03.001 [pool-1-thread-2] INFO reactor.Flux.Array.2 - | onNext(8)

23:48:03.002 [pool-1-thread-1] INFO reactor.Flux.Array.1 - | onComplete()

23:48:03.002 [pool-1-thread-2] INFO reactor.Flux.Array.2 - | onComplete()

\*/

Ở đây chúng ta dùng [ExecutorService](https://loda.me/khai-niem-thread-pool-va-executor-trong-java-loda1554800053212), 2 luồng code thực hiện song song trên 2 thread khác nhau, điều mà đã chứng minh bằng output.

**Kết luận**

Sau bài viết này, chúng tôi đã có cái nhìn tổng quan về Reactor Core. Từ các tạo một Publisher , các đăng ký, backpressure cũng như xử lý không đồng bộ. Đây cũng là nền tảng để cho chúng tôi viết cái bài viết khác liên quan về Reactor Core.