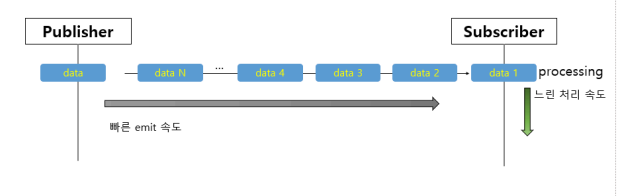
3일차 - Chapter 8~9

Chapter 8 Backpressure

배압 또는 역압

Publisher가 끊임없이 emit하는 무수히 많은 데이터를 적절하게 제어하여 데이터 처리에 과부하가 걸리지 않도록 제어하는 것.



https://velog.io/@zini9188/Spring-WebFlux-Project-Reactor

Publisher가 빠른 속도로 emit 할 경우 Subscriber의 처리속도가 느려서 처리하기도 전에 emit 됨.

이렇게 될 경우 오버플로가 발생하거나 최악의 경우 시스템이 다운되는 문제가 발생

⇒ 이 문제를 해결하기 위한 수단이 Backpressure

Reactor에서의 Backpressure 처리방식

1. 데이터 개수 제어

Subscriber가 적절히 처리할 수 있는 수준의 데이터 개수를 request()를 통해 Publisher에게 요청하는 것.

```
@Slf4j
public class Example8_1 {
    public static void main(String[] args) {
        Flux.range(1, 5)
            .doOnRequest(data -> log.info("# doOnRequest: {}", data))
            .subscribe(new BaseSubscriber<Integer>() {
                @Override
                // onSubscribe를 대신하여 구독 시점에 request()를 호출해 최초 데이터 요청 개수 제어
                protected void hookOnSubscribe(Subscription subscription) {
                    request(1);
                }

                @SneakyThrows
                @Override
                // onNext를 대신해 Publisher가 emit한 데이터를 전달받아 처리한 후 데이터 재요청할때
                protected void hookOnNext(Integer value) {
                    Thread.sleep(2000L);
                    log.info("# hookOnNext: {}", value);
                    request(1);
                }
            });
```

```
    }
}
```

c2. Backpressure 전략 사용

종류	설명
IGNORE 전략	Backpressure를 적용하지 않는다.
ERROR 전략	Downstream으로 전달할 데이터가 버퍼에 가득 찰 경우, Exception을 발생시킨다.
DROP 전략	Downstream으로 전달할 데이터가 버퍼에 가득 찰 경우, 버퍼 밖에서 대기하는 먼저 emit된 데이터부터 Drop시킨다.
LATEST 전략	Downstream으로 전달할 데이터가 버퍼에 가득 찰 경우, 버퍼 밖에서 대기하는 가장 최근에(나중에) emit된 데이터부터 버퍼에 채운다.
BUFFER 전략	Downstream으로 전달할 데이터가 버퍼에 가득 찰 경우, 버퍼 안에 있는 데이터부터 Drop시킨다. - DROP_LATEST - DROP_OLDEST

Chapter 9 - Sinks

Processor 인터페이스의 기능을 개선한 것.

> 💡 Processor란?
> Publisher와 Subscriber의 기능을 모두 지니기 있는 구성요소

Flux 또는 Mono가 onNext 같은 signal을 내부적으로 전송하는 방식이었는데, Sinks를 사용하면 프로그래밍 코드를 통해 명시적으로 Signal을 전송 가능

Operator 사용방식 vs Sinks 사용방식

일반적으로 generate()나 create() Operator는 싱글스레드 기반에서 Signal을 전송하는데 사용하지만, Sinks는 멀티스레드 방식으로 Signal을 전송해도 스레드 안전성을 보장하기 때문에 예기치 않은 동작으로 이어지는 것을 방지

```
13:54:33.391 [boundedElastic-1] INFO com.example.demo.DemoApplication -- # create() : Task 1 result
13:54:33.393 [boundedElastic-1] INFO com.example.demo.DemoApplication -- # create() : Task 2 result
13:54:33.393 [boundedElastic-1] INFO com.example.demo.DemoApplication -- # create() : Task 3 result
13:54:33.394 [boundedElastic-1] INFO com.example.demo.DemoApplication -- # create() : Task 4 result
13:54:33.394 [boundedElastic-1] INFO com.example.demo.DemoApplication -- # create() : Task 5 result
13:54:33.394 [parallel-2] INFO com.example.demo.DemoApplication -- # map() : Task 1 resultsuccess!!
13:54:33.394 [parallel-2] INFO com.example.demo.DemoApplication -- # map() : Task 2 resultsuccess!!
13:54:33.394 [parallel-2] INFO com.example.demo.DemoApplication -- # map() : Task 3 resultsuccess!!
13:54:33.394 [parallel-2] INFO com.example.demo.DemoApplication -- # map() : Task 4 resultsuccess!!
13:54:33.394 [parallel-1] INFO com.example.demo.DemoApplication -- # onNext() : Task 1 resultsuccess!!
13:54:33.394 [parallel-2] INFO com.example.demo.DemoApplication -- # map() : Task 5 resultsuccess!!
13:54:33.394 [parallel-1] INFO com.example.demo.DemoApplication -- # onNext() : Task 2 resultsuccess!!
13:54:33.394 [parallel-1] INFO com.example.demo.DemoApplication -- # onNext() : Task 3 resultsuccess!!
13:54:33.394 [parallel-1] INFO com.example.demo.DemoApplication -- # onNext() : Task 4 resultsuccess!!
13:54:33.394 [parallel-1] INFO com.example.demo.DemoApplication -- # onNext() : Task 5 resultsuccess!!
```

위 코드에서 작업을 처리한 후, 그 결과 값을 반환하는 doTaks() 메서드가 싱글스레드가 아닌 여러 개의 스레드에서 각각의 "전혀 다른 작업들"을 처리한 후, 처리 결과를 반환하는 상황이라면? 현재는 한 개의 boundedElastic-1 스레드에서 doTask() 메서드가 실행되고 있는 것을 볼 수 있습니다. 이 같은 상황에서 적절히 사용할 수 있는 방식이 Sinks.

Sinks는 프로그래밍 방식으로 Signal을 전송할 수 있으며, 멀티 스레드 환경에서 스레드 안정성을 보장받을 수 있는 장점이 있음.

Sinks 종류 및 특징

Sinks.One()

한 건의 데이터를 emit

호출 시 Sinks.One 객체로 데이터가 Emit 됨

아무리 많은 수의 데이터를 emit 하더라도 처음 emit한 데이터만 정상적으로 emit. 나머진 다 Drop

Sinks.Many()

여러 건의 데이터를 여러가지 방식으로 전송

호출 시 Sinks.Many가 아닌 ManySpec이라는 인터페이스 리턴

Manyspec 인터페이스

- unicast() - 단 하나의 Subscriber에게만 데이터를 emit
- multicast() - 하나 이상의 Subscriber에게 데이터를 emit
- replay() - emit된 데이터 중에서 특정 시점으로 되돌린 (replay) 데이터로부버 emit

FAIL_FAST
EmitFailureHandler 객체를 통해 emit 도중 에러가 발생했을 때 재시도를 하지 않고 즉시 실패 처리를 함.
⇒ 빠르게 실패처리함으로써 교착 상태 등을 미연에 방지하여, 스레드 안전성 보장