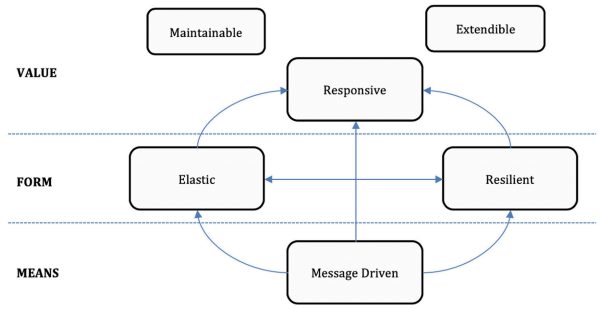
리액티브 프로그래밍 개요와 구성

1. 리액티브 선언문

리액티브 시스템의 설계 원칙



출처: https://velog.io/@rolroralra/Chapter01-리액티브-시스템과-리액티브-프로그래밍

- MEANS : 비동기 메시지 기반의 통신

  → 구성요소들 간의 느슨한 결합, 격리성, 위치 투명성을 보장
- FORM : 비동기 메시지 통신 기반 하에 탄력성과 회복성을 가지는 시스템
  - 탄력성 - 시스템의 작업량이 변화하더라도 일정한 응답을 유지하는 것을 의미
  - 회복성 - 장애가 발생하더라도 응답성을 유지하는 것을 의미

    → 일정한 응답을 유지하기 위해서 입력을 처리하기 위한 시스템 자원을 적절하게 추가하거나 감소시켜 작업량의 변화에 대응(ex. backpressure)
- VALUE : 시스템의 처리량을 자동으로 확장하고 축소하는 탄력성을 확보. 즉각적으로 응답 가능한 시스템을 구축할 수 있음을 의미

목표: 빠른 응답성을 바탕으로 유지보수와 확장이 용이한 시스템을 구축

2. 선언형 프로그래밍 vs 명령형 프로그래밍

프로그래밍 방식

- 명령형 프로그래밍 방식: 어떤 작업을 처리하기 위해 실행할 동작을 코드에 구체적으로 명시하는 방식
- 선언형 프로그래밍 방식: 실행할 동작을 구체적으로 명시하지 않고 목표만 선언하는 방식

참고자료

- Java 함수형 프로그래밍

Java 함수형 프로그래밍 - 풀코스(3시간27분)

#java #functional #풀코스

[설명]

https://www.youtube.com/watch?v=9Q5IyeGfTaQ

- 관련도서 - 자바 인 액션

product.kyobobook.co.kr

https://product.kyobobook.co.kr/detail/S000001057597

3. 리액티브 프로그래밍 구성

참고자료

- 리액티브 프로그래밍 튜토리얼

Introduction to Reactive Programming - Reactive Programming with Reactor 3

Explore this playground and try new concepts right into your browser

https://tech.io/playgrounds/929/reactive-programming-with-reactor-3/Intro

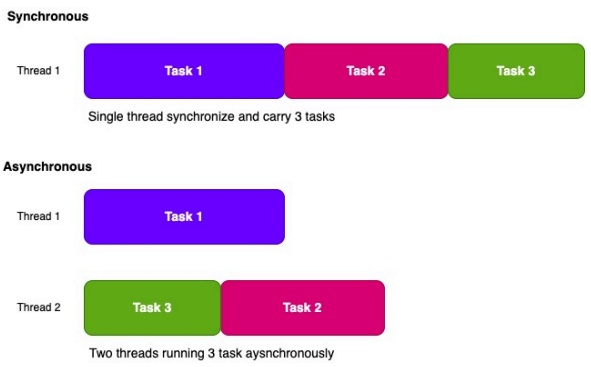
4. Blocking I/O , Non-Blocking I/O

운영체제 I/O : 컴퓨터 시스템이 외부의 입출력 장치들과 데이터를 주고 받는 것을 의미

디스크에 저장된 프로그램 실행파일을 읽어 들여 메모리에 올리는 것

외에도 파일 I/O, DB I/O 등이 존재

Blocking I/O



Blocking I/O 방식에서는 `Task 1` 이라는 작업 속에서 I/O 작업이 있다고 가정하면 작업을 마무리하기 전까지 스레드가 차단되어 대기하게 된다.

이 방식을 보완하기 위해서 멀티스레딩 기법으로 추가 스레드를 할당하여 그 시간을 효율적으로 사용하는 방법을 채택해볼 수 있다.

→ 컨텍스트 스위칭으로 인한 스레드 전환 비용이 발생. (멀티 프로세스, 멀티 스레드 방법)

단점

- 과다한 메모리 사용으로 오버헤드가 발생

  일반적으로 서블릿 컨테이너 기반의 Java 웹 애플리케이션은 요청당 하나의 스레드를 할당. 운영 환경에서 각각의 스레드 내부에서 도 다른 작업 처리를 위해 스레드를 추가 할당할 경우

  메모리 사용량이 증가해 문제가 발생할 가능성이 있다.

- 스레드 풀의 응답 지연 발생

  대량의 요청이 발생하게 되어 스레드 풀에 사용 가능한 유휴 스레드가 없을 경우, 사용 가능한 스레드가 확보되기 전까지 응답 지연 발생

  ※ 스레드 풀: 일정 개수의 스레드를 미리 생성해서 풀에 저장해두고 사용자 요청이 들어올 경우, 아직 사용되지 않고 있는 스레드가 있다면 풀에서 꺼내어 사용할 수 있도록 스레드 저장소

Non-Blocking I/O

작업 스레드의 종료 여부와 관계없이 요청한 스레드는 차단되지 않는다. 추가적으로 Blocking I/O 방식보다 적은 수의 스레드를 사용하므로 멀티스레딩 기법을 사용할 때

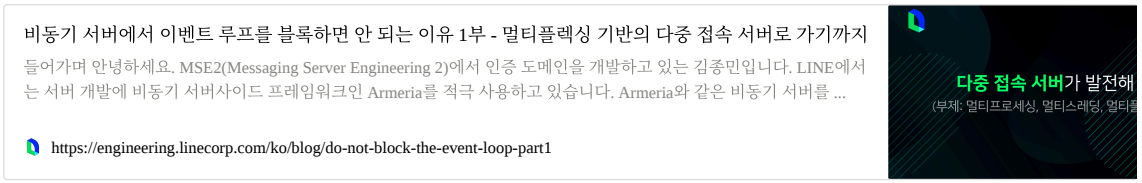
발생한 문제점들이 생기지 않고 CPU 대기 시간과 사용량에 있어서도 효율적이다.

단점

- 스레드 내부에서 CPU를 많이 사용하는 작업이 포함된 경우에는 성능에 악영향
- 사용자의 요청에서 응답까지 전체 과정에 Blocking I/O 요소가 포함된 경우에는 Non-Blocking의 장점을 발휘하기 어려움

참고자료

- 비동기 서버 Non-Blocking



5. Cold / Hot Sequence

Sequence는 Publisher가 emit하는 데이터의 연속적인 흐름을 정의해놓은 것. 코드로 표현하면 Operator 체인 형태로 정의

Cold Sequence

Subscriber가 구독할 때마다 데이터 흐름이 처음부터 시작되는 Sequence.

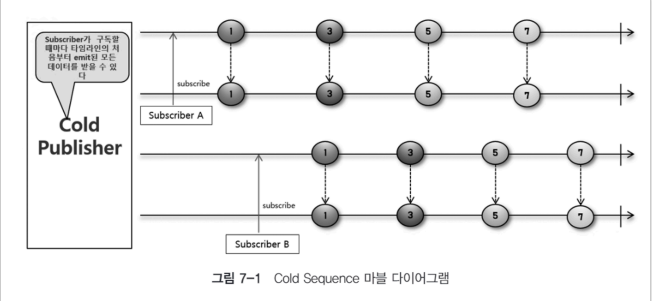


그림 7-1 Cold Sequence 마블 다이어그램

SubScribe A와 Subscribe B의 구독 시점이 달라도 모두 동일한 데이터를 전달받는 것.

→ Subscriber의 구독 시점이 달라도 구독을 할 때마다 Publisher가 데이터를 emit하는 과정을 처음부터 다시 시작하는 데이터 흐름을 의미

Hot Sequence

Subscriber가 구독할 때에 데이터 흐름이 구독 시점 이후에 emit된 데이터만 전달받는 Sequence

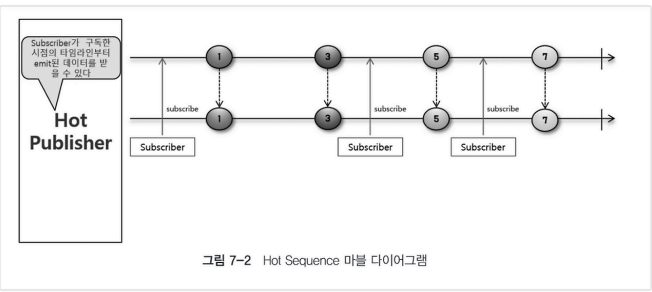


그림 7-2 Hot Sequence 마블 다이어그램

Subscribe A와 Subscribe B의 구독 시점이 다를 경우 구독 시점 이후의 emit된 데이터만 전달받는 것.

구독 횟수와는 별개로 Sequence 타임라인이 하나만 생기게 됨.

→ 구독이 발생한 시점 이전에 Publisher로부터 emit된 데이터는 Subscriber가 전달받지 못하고 구독이 발생한 시점 이후에 emit된 데이터만 전달

코드로 살펴보는 Sequence

```
public static void main(String[] args) throws Exception {

    Flux<String> conert = Flux
                                .fromArray(new String[]{"A", "B"}) // Flux: 원본 Flux
                                .delayElements(Duration.ofSeconds(1)) // Operator 체인 형태로 리턴되는 Flux는 모두 다
                                .share(); // 여러 Subscriber가 하나의 원본 Flux를 공유

    concert.subscribe(singer -> log.info("{}'s song", singer);

    Thread.sleep(2500);

    concert.subscribe(singer -> log.info("{}'s song", singer); // 구독 시점 이후에 emit된 데이터를 처리
```

Reactor에서는 Hot이라는 키워드에 대해 다음의 두 가지를 의미를 가지고 있다.

최초 구독이 발생하기 전까지는 데이터의 emit이 발생하지 않는 것

구독 여부와 상관없이 데이터가 emit되는 것

6. Backpressure

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

상대적으로 Subscriber가 데이터를 처리하는 속도보다 Publisher가 emit 하는 것이 빠른 경우가 많다.
이에 데이터를 적절하게 제어하여 처리에 과부하가 걸리지 않도록 제어하는 것이 Backpressure의 역할이라고 볼 수 있다.
만약 아주 빠른 속도로 데이터를 끊임없이 emit하게 된다면?

→ 대기 중인 데이터가 지속적으로 쌓이게 되면 오버플로가 발생하거나 최악의 경우에는 시스템이 다운되는 문제가 발생.