1일차 - Chapter 1.

자 위 설명들은 모두 리액티브 프로그래밍을 위한 인트로에 불과하였다.

우리가 비즈니스 로직을 작성하고, 서버를 운영하게 되었을때 기존 Spring MVC 모델을 사용한다고 가정하면 스레드 모델 자체가 요청당 하나의 스레드가 작업이 이뤄지게 되는데

이때, 처리방식이 동기에 해당하기에 블로킹이 발생할 경우 쓰레드 풀을 이용하더라도 제한적인 쓰레드 갯수에 대한 문제는 해결 할 수 없음.

▼ 블로킹(Blocking)과 논블로킹(Non-blocking)

- **블로킹(Blocking)**: 어떤 작업이 끝날 때까지 **다른 작업을 못하고 기다리는 상태**. 예를 들어, 파일을 읽는 도중 CPU가 대기하면 다른 요청을 처리할 수 없음
- **논블로킹(Non-blocking)**: 작업이 완료되길 기다리지 않고 **다른 작업을 먼저 처리하는 방식**. 결과가 준비되면 콜백이나 이벤트로 통보받음

- 쓰레드의 생성 비용은 비싸다 > 요청 때마다 생성하면 응답속도가 느려진다.
- 쓰레드는 컨텍스트 스위칭 비용이 발생한다.
- 쓰레드 생성에 제한이 없으면 요청 수 만큼 계속 생성되고 메모리 허용범위가 넘어서면 서버가 죽는다.

즉, 이 문제를 해결하기 위해 논블로킹을 가진. 즉, 비동기 프로그래밍이 필요함.

이게 Spring WebFlux.

⚙️ Spring MVC와 Spring WebFlux의 차이점

항목	Spring MVC	Spring WebFlux
처리 방식	동기(블로킹)	비동기(논블로킹)

항목	Spring MVC	Spring WebFlux
스레드 모델	요청당 하나의 스레드	이벤트 루프 기반, 적은 수의 스레드로 다수 처리
기반 기술	서블릿 API	Netty, Undertow 등
적합한 상황	일반적인 웹 애플리케이션	높은 동시성, 실시간 데이터 처리 등

리액티브 시스템

반응을 잘하는 시스템

리액티브 시스템에서 반응을 잘한다는 것

⇒ 클라이언트의 요청에 즉각적으로 응답함으로써 지연 시간을 최소화

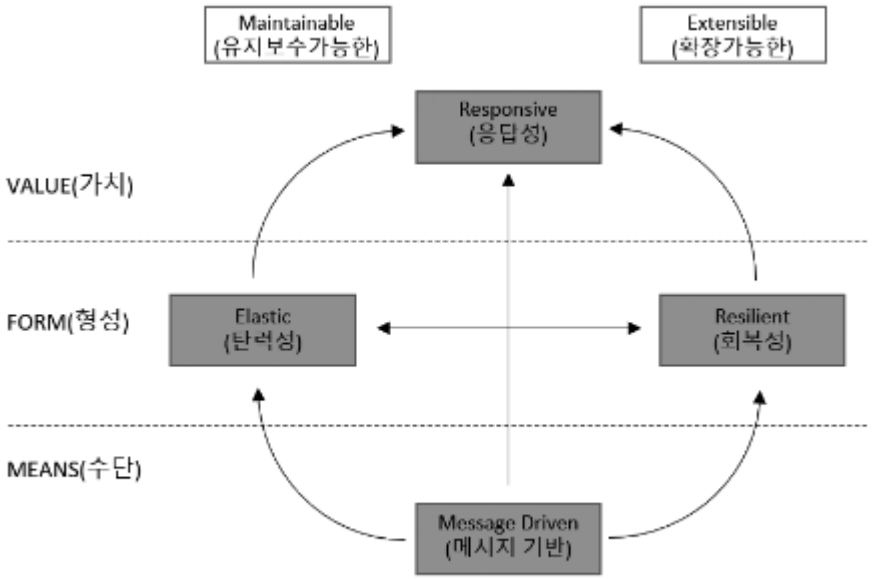

그림 1-1 리액티브 선언문 설계 원칙

- MEANS - 리액티브 시스템에서 주요 통신 수단은?

  비동기 메시지 기반의 통신

- `FORM` - 메시지 기반 통신을 통해 어떤 형태를 지니는 시스템인지?
  작업량이 변하더라도 일정한 응답을 지닌
  탄력성,
  시스템 장애가 발생하더라도 응답을 지닌
  회복성
- `VALUE` - 핵심가치는?

  빠른 응답성을 바탕으로 유지보수와 확장이 용이한 시스템

리액티브 프로그래밍이란?

리액티브 시스템을 구축하는 데 필요한 프로그래밍 모델

리액티브 프로그래밍의 특징

- 선언형 프로그래밍 방식. 그렇기에 실행할 동작을 구체적으로 명시하지 않고 목표만 선언
- 데이터 소스의 변경이 있을 때마다 데이터를 전파
- 리액티브 프로그래밍 코드는 코드의 간결함과 가독성에 유리한 메서드 체인의 형태로 표현
- 리액티브 프로그래밍 코드에서 파라미터를 가지는 메서드는 함수형 프로그래밍 방식의 코드 형태의 파라미터를 지님

▼ 명령형 프로그래밍과 선언형 프로그래밍

식당에 가서 음식을 주문하기 전에 물 한잔 마시는 상황을 떠올렸을 때.

내가 직접 식당 한쪽에 있는 정수기로 걸어가서 차가운 물을 컵에 따르는 것은 명령형,

“여기 차가운 물 한잔 주세요”라고 종업원에게 부탁하는 것은 선언형.

코드 1-1 명령형 프로그래밍 코드 예제(Example1_1.java)

```java
public class Example1_1 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 3, 21, 10, 8, 11);
        int sum = 0;
        for(int number : numbers){
            if(number > 6 && (number % 2 != 0)){
                sum += number;
            }
        }

        System.out.println("합계: " + sum);
    }
}
```

실행 결과

```
합계: 32
```

코드 1-2 선언형 프로그래밍 코드 예제(Example1_2.java)

```java
public class Example1_2 {
    public static void main(String[] args) {
        List<Integer> numbers = Arrays.asList(1, 3, 21, 10, 8, 11);
        int sum = numbers.stream()
            .filter(number -> number > 6 && (number % 2 != 0))
            .mapToInt(number -> number)
            .sum();

        System.out.println("합계: " + sum);

    }
}
```

실행 결과

```
합계: 32
```

해당 코드에서 선언형 코드를 보면 여러 가지 코드를 초기화, 할당, 조건, 반복 등으로 나누지 않고 `stream()`, `filter()`, `mapToInt()`, `sum()`으로 메서드 체인을 형성하여 한 문장으로 스트림 내부에서 대신 처리하는 것을 볼 수 있음.

선언형 프로그래밍 방식도 사실은 함수형으로 구성되어 있기에
filter 메서드의 파라미터 부분이 함수형 프로그래밍 방식이라고도 볼 수 있음.

리액티브 프로그래밍 코드 구성

- **Publisher(생산자)**
  - 입력으로 들어오는 데이터를 Subscriber에 제공하는 역할을 함.
- **Subscriber(소비자)**
  - Publisher로부터 전달받은 데이터를 사용하는 역할을 함.
- **Data Source**
  - Publisher의 입력으로 전달되는 데이터를 의미함.
- **Operator**
  - Publisher와 Subscriber 중간에서 데이터를 가공하는 역할을 함.

리액티브 프로그래밍은 “Operator로 시작하여 Operator로 끝난다” 는 말이 있을 정도로 데이터 필터링, 변환 등 수많은 Operator가 존재.