Asynchronous Programming

짚고 넘어가야 할 중요 키워드

- Sync / Async, Blocking / Non-Blocking
- Parallelism / Concurrency
- Asynchronous programming

Sync / Async, Blocking / Non-Blocking

Synchronous / Asynchronous

Syn (together) + chronous (time)

동기/비동기를 이야기할때는 반드시 **2가지 이상** 언급해야하며 **시간 개념**이 있어야한다.

어떤 시간을 맞춘다 -> **동기**
어떤 시간을 안맞춘다 -> **비동기**

A와 B의 시작시간 또는 종료시간이 일치하면 동기
메소드 리턴시간과 결과를 전달받는 시간이 일치하면 동기
A가 끝나는 시간과 B가 시작하는 시간이 같으면 동기

Blocking / Non-Blocking

- 동기/비동기와 관점이 다름
- 내가 직접 제어할 수 없는 대상을 상대하는 방법
- ex) IO

Blocking / Non-Blocking

"내가 제어하지 못하는 제 3의 존재, 대상을 내가 어떤식으로 상대할 것인가"

외부 입출력과 프로그램 사이의 작업을 어떻게 할 것인가

Blocking I/O

프로그램이 외부와 데이터를 주고 받을때 외부에서 데이터를 처리하는 동안 가만히 기다리는 것.

Non-Blocking I/O

외부에서 필요한 작업을 하는 동안 프로그램은 다른 일을 할 수 있음.
-> **자원을 효율적으로 사용가능**
-> 현재 맥락과 상관없이 외부 작업과 데이터를 공유할 방법이 필요. 이를 위한 추가 구현이 필요.

Non-Blokcing I/O 에서 결과를 가져오는 방식에 따라 2 분류로 나누기도함.

1. **주도적**으로 데이터를 가져오는 방식(Pull Base) - Non-Blocking I/O
2. **수동적**으로 데이터를 받아오는 방식(Push Base) - Asynchronous I/O

일반적으로 "수동적으로 받아오는 방식"이 자원을 훨씬 효율적으로 사용.

여러가지 일을 다루는 2가지 방식

Parallelism, Concurrency

1. 병렬성

- 작업을 처리하는 일꾼을 물리적으로 여러개 둬 같은 작업을 동시에 수행.

2. 동시성

- 일꾼의 수와는 상관없이 일꾼이 일하는 방식과 연관있음.
- 일꾼의 작업을 차례대로 끝내가며 수행 X
- 맡은 일이 끝나지 않았음에도 여러 일을 **번갈아 가며 진행**.

둘의 의존성은 전혀 없다!

동시성 - **자원을 효율적**으로 사용하는 것이 목적
병렬성 - 많은 자원을 투자해서 **일의 처리량을 늘리는 것**이 목적.

어떤 문제이냐에 따라 둘 중에 하나를 적용하기도 하고 둘 다 적용하기도 한다.

동시성과 Non-Blokcing I/O 를 같이 사용하면 시너지 효과 발생!
But, **둘 사이에 의존성은 전혀 없다!**

Non-Blocking I/O - 외부 I/O 와 프로그램 사이의 작업을 어떻게 할 것인가
동시성 - 일꾼이 일하는 방식을 대표하는 개념

비동기 프로그래밍(Asynchronous Programming)

비동기 프로그래밍?

- 실행한 코드의 결과를 기다리지 않고 별도의 채널에 결과 처리를 맡긴 후 다음 작업을 바로 진행하는 방식의 프로그래밍.
- 자원을 더 효율적으로 사용할 수 있는 여지가 생김.

비동기 프로그래밍에서 결과를 처리하는 방식

1. 함수 전달을 통해 처리(Callback)
2. 언어에서 지원하는 방식(Future, Promise 등)

함수 전달을 통해 처리(Callback)

함수를 값 처럼 사용하는 기법 (First-class Function).
비동기 프로그래밍 방식으로 진행할 코드를 실행할 때 실행하는 시점에 결과를 처리해 줄 함수를 일꾼에게 같이 넘기는 기법.
일꾼은 작업 후 실제 결과를 실행 시점에 받은 함수를 통해 처리.

언어에서 지원하는 방식

비동기로 실행 한 코드의 결과를 언어에서 지원하는 방식으로 되돌려 받아 처리하는 방식.
코드를 비동기로 실행할 때 해당 코드의 실제 결과가 아닌 언어에서 지정한 방식으로 결과를 즉시 되돌려 받는다.

- Future, Promise 같은 객체 형태
- Python 코루틴, Golang 고루틴 과 같은 언어의 문법을 이용하는 형태
- 결과를 주고받을 별도의 채널을 사용하는 형태

즉시 되돌려 받은 결과를 이용하여 각 방식에 알맞게 추후에 비동기로 실행 한 코드의 실제 결과를 꺼내어 사용하는 형태.

- 객체 형태
  실행 흐름을 프로그램으로 구현하여 높은 자유도를 누릴 수 있으나 이를 위한 코드를 더 구현해야함.
- 언어의 문법을 이용한 형태
  언어가 제공하는 제어문을 통해 비동기 프로그래밍의 실행 흐름을 언어의 실행흐름으로 가져와 코드를 쉽게 작성 가능.

Asynchronous Wait (Await)

비동기 프로그래밍으로 실행했던 작업의 결과를 다시 최초의 실행 흐름으로 되가져와 사용해야하는 경우 이 때 Await 는기존의 실행 흐름에서 필요한 결과를 비동기 프로그래밍으로 실행한 작업에서 만들어낼 때 까지 기다려 두 작업 간 타이밍을 맞추어 주는 역할을 함.

비동기 프로그래밍으로 실행 한 작업과 원래 실행 흐름을 묶어주는 중요한 기법

Asynchronous Programming - Asynchronous Wait 를 한 쌍으로 둠

절차 중심 언어에서는 Await 를 통해 실행 흐름을 제어하는 것이 일반적.

Reference

- 비동기 프로그래밍과 웹 프레임워크
- Blocking-NonBlocking-Synchronous-Asynchronous
- 멈추지 않고 기다리기(Non-blocking)와 비동기(Asynchronous) 그리고 동시성(Concurrency)
- 스프링캠프 2017 [Day1 A2] : Async & Spring
- blocking, non-blocking and async