스레드

01 스레드

개념

하드웨어적 스레드: 하나의 코어가 동시에 처리하는 명령어 단위

소프트웨어적 스레드: 어떠한 프로그램 내에서, 특히 프로세세스 내에서 실행되는 흐름의 단위

일반적으로 한 프로그램은 하나의 스레드를 가지고 있지만 프로그램 환경에 따라
둘 이상의 스레드를 동시에 실행 할 수 있다

멀티스레드

01 스레드

스레드와 프로세스

프로세스

실행파일로 존재하던 프로그램이 메모리에 적재되어 CPU에 의해 실행되는 것

스레드

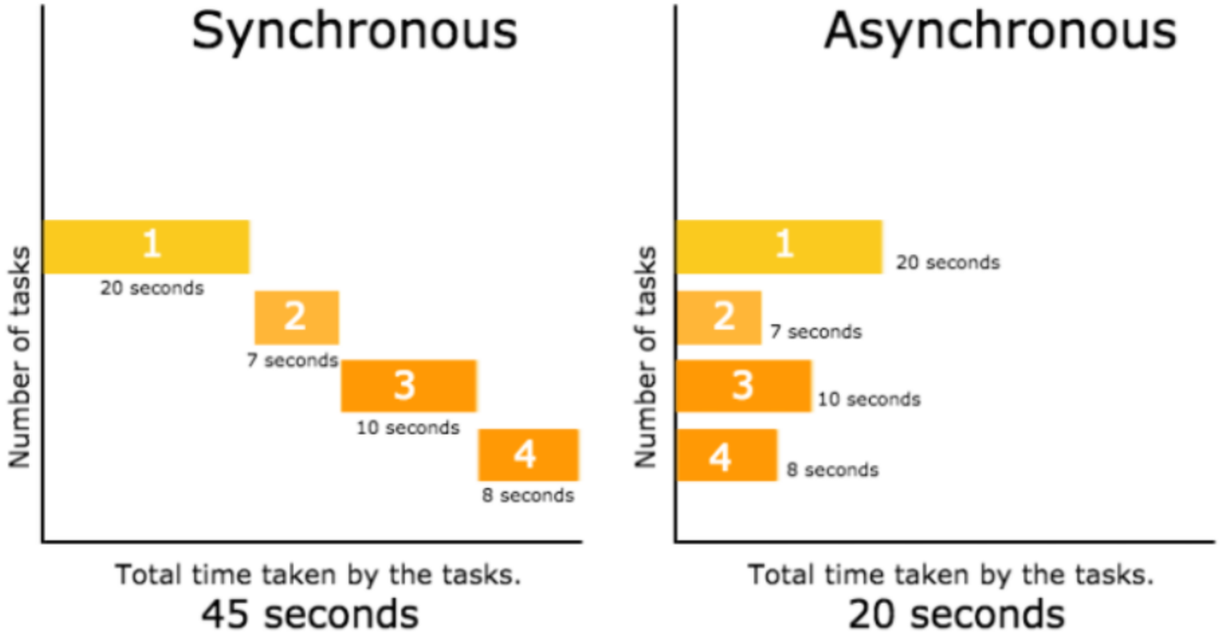
프로세스 내에서 실행되는 흐름의 단위

- 멀티 스레드와 멀티 프로세스는 양쪽 모두 여러 흐름이 동시에 진행된다.
- 멀티 프로세스에서 각 프로세스는 독립적으로 실행되며 각각 별개의 메모리로 차지하고 있다.
- 멀티 스레드는 프로세스 내의 메모리를 공유해 사용할 수 있다.
- 프로세스 간의 전환 속도보다 스레드 간의 전환 속도가 더 빠르다.

02 스레드풀

비동기란!

앞선 작업이 끝나기를 대기하며 수행되는게 아니라 앞선 작업의 종료까지 대기하지 않고 다음 작업을 수행하는 것



02 스레드풀

스레드풀: 작업 처리에 사용되는 스레드를 제한된 개수만큼 정해 놓고 작업 큐에 들어오는 작업들을 하나씩 스레드가 맡아서 처리하는 것

병렬처리 작업이 많아 질 수록 스레드 개수가 늘어나고 그에 따른 스레드 생성과 스케줄링으로 인해 CPU가 바빠져서 메모리 사용량이 증가한다.

이러한 문제를 해결하기 위해 **스레드풀**을 만들어 처리한다.

02 스레드풀

구성요소

스레드풀 크기

스레드풀이 생성할 수 있는 최대 스레드 수를 나타냄
이는 동시에 실행 가능한 작업의 수를 제한하는 데 사용

작업 큐

스레드풀에 제출된 작업을 저장하는 큐
작업 큐에는 스레드풀의 스레드가 처리할 작업들이 대기함

02 스레드풀

구성요소

작업 실행 정책

작업 큐가 가득 찼을 때 새로운 작업을 어떻게 처리할지를 정의함
일부 정책은 작업을 큐에 추가하지 않고 즉시 실행하거나 작업을 거부할 수도 있음

스레드 팩토리

스레드풀이 새로운 스레드를 생성할 때 사용되는 팩토리
기본적으로는 자바의 내장 스레드 팩토리를 사용하며, 사용자 지정 스레드 팩토리를 제공하여 스레드 생성 및 초기화를 조정 할 수 있음

02 스레드풀

스레드 풀 생성

CatchedThreadPool

작업이 들어올 때 마다 스레드를 생성하며 생성된 스레드가 60초 동안 아무일도 하지 않으면 즉, 유휴상태라면 해당 스레드를 제거한다.

FixedThreadPool

존재하는 스레드의 개수보다 작업량이 많으면 스레드를 생성하며 생성된 스레드는 유휴상태가 오래되었다고 하더라도 제거하지 않는다.

ThreadPoolExecutor

존재하는 스레드의 개수보다 작업량이 많으면 스레드를 생성하며 생성된 스레드는 유휴상태가 오래되었다고 하더라도 제거하지 않는다.

02 스레드풀

스레드풀 종료

shutdown()

현재 쓰레드가 처리중인 작업과 작업 큐에 대기하고 있는 작업을 모두 끝 마친 뒤 쓰레드 풀을 종료한다.

shutdownNow()

현재 작업중인 쓰레드를 강제로 작업 중지하고, 작업 큐 대기열에 남아있는 작업을 List<Runnable>로 반환한다.

awaitTermination()

shutdown() 을 먼저 호출하고 전달된 timeout 시간 내로 모든 작업이 완료 되면 ture를 완료하지 못하면 처리중인 스레드를 강제 종료하고 false를 반환한다.

02 스레드풀

작업 처리 단위

Runnable

작업 처리 후 반환값이 없다.

Callable

작업 처리 후 반환 값이 있다.

02 스레드풀

작업 처리 요청

execute()

Runnable 만을 처리하며 작업 처리의 결과를 반환 받을 수 없다.
스레드에서 작업처리 도중 예외가 발생하면 해당 예외를 제거하고 새로운 스레드를 생성한다.

submit()

Runnable 과 Callable 두가지를 처리하며 작업의 처리 결과를 Future 라는 인터페이스로 반환한다.
작업처리 도중 예외가 발생해도 스레드를 제거하지 않고 다음 작업에 재사용한다.

02 스레드풀

```java
package com.example.studytest;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Main {
    public static void main(String[] args) {
        // 스레드풀 생성
        ExecutorService executor = Executors.newFixedThreadPool( nThreads: 5); // 최대 5개의 스레드를 가진 스레드풀 생성

        // 작업 제출
        for (int i = 0; i < 10; i++) {
            Runnable worker = new WorkerThread( taskName: "Task " + (i+1));
            executor.execute(worker); // 작업을 스레드풀에 제출
        }

        // 작업이 모두 완료될 때까지 대기
        executor.shutdown(); // 스레드풀 종료

        while (!executor.isTerminated()) {
        }

        System.out.println("모든 작업이 완료됨");
    }
}
```

```java
// 작업을 수행할 Runnable 구현 클래스
1 usage
class WorkerThread implements Runnable {
    3 usages
    private String taskName;

    1 usage
    public WorkerThread(String taskName) {
        this.taskName = taskName;
    }

    @Override
    public void run() {
        System.out.println(Thread.currentThread().getName() + " 작업 시작: " + taskName);
        try {
            // 작업 시뮬레이션
            Thread.sleep( millis: 2000); // 2초간 대기
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(Thread.currentThread().getName() + " 작업 완료: " + taskName);
    }
}
```

02 스레드풀

```
> Task :Main.main()
pool-1-thread-4 작업 시작: Task 4
pool-1-thread-5 작업 시작: Task 5
pool-1-thread-2 작업 시작: Task 2
pool-1-thread-1 작업 시작: Task 1
pool-1-thread-3 작업 시작: Task 3
pool-1-thread-1 작업 완료: Task 1
pool-1-thread-1 작업 시작: Task 6
pool-1-thread-4 작업 완료: Task 4
pool-1-thread-5 작업 완료: Task 5
pool-1-thread-3 작업 완료: Task 3
pool-1-thread-3 작업 시작: Task 9
pool-1-thread-2 작업 완료: Task 2
pool-1-thread-4 작업 시작: Task 7
pool-1-thread-5 작업 시작: Task 8
pool-1-thread-2 작업 시작: Task 10
pool-1-thread-1 작업 완료: Task 6
pool-1-thread-2 작업 완료: Task 10
pool-1-thread-5 작업 완료: Task 8
pool-1-thread-4 작업 완료: Task 7
pool-1-thread-3 작업 완료: Task 9
모든 작업이 완료됨
```

03 출처

https://velog.io/@wijoonwu/%EB%A9%B4%EC%A0%91-%EC%A7%88%EB%AC%B8
https://akku-dev.tistory.com/89
https://dingdingmin-back-end-developer.tistory.com/entry/SpringBoot%EB%8A%94-%EC%8B%B1%EA%B8%80%ED%86%A4%EC%9D%B8%EB%8D%B0-%EC%96%B4%EB%96%BB%EA%B2%8C-%EB%8B%A4%EC%A4%91-%EC%9A%94%EC%B2%AD%EC%9D%84-%EC%B2%98%EB%A6%AC%ED%95%A0%EA%B9%8C
https://cheershennah.tistory.com/170
https://hudi.blog/java-thread-pool/