01

01-3. Coroutine Builder와 Job

Coroutine

runBlocking()

- runBlocking() 은 새로운 코루틴을 만들어 실행시키는 코루틴 빌더
- 코루틴이 모두 종료될 때까지 코루틴을 실행시킨 스레드는 대기상태
- main 함수 혹은 단위 테스트를 위한 함수에서 suspend 함수 혹은 코루틴 빌더를 사용하는 목적으로 만 제한적으로 사용을 권장

```
runBlocking {
    repeat(2){
        delay(200)
        println("coroutine $it")
    }
}
```

launch()

- 코루틴 빌더
- launch() 에 의해 만들어진 코루틴은 Non-Blocking 으로 실행
- launch() 함수는 Job 을 리턴

launch()

- launch() 함수의 매개변수에 CoroutineStart 를 지정
- 코루틴을 바로 실행 시켜야 하는지 아니면 필요한 순간에 실행시켜야 하는지를 제어

  - CoroutineStart.DEFAULT : 기본 값, 바로 시작, 시작되기 전에 취소하면 취소됨
  - CoroutineStart.ATOMIC : 바로 시작, 시작되기 전에 취소해도 시작됨.
  - CoroutineStart.LAZY : 어디선가 Job 의 start() 함수가 호출될 때 시작
  - CoroutineStart.UNDISPATCHED : 바로 시작, 시작되기 전에 취소해도 시작됨. 현재 스레드에서 시작되지만 일시중단 되었다가 다시 실행될 때 CoroutineContext 에 지정된 스레드에서 실행됨.

```
val job2 = launch(start = CoroutineStart.ATOMIC) {

}
```

Job

- Job 은 코루틴에 의해 실행되는 업무를 지칭하는 객체
- launch() 함수의 리턴값으로 획득하는 방법과 Job() 함수로 획득하는 방법
- Job() 함수로 획득되는 객체는 CompletableJob 타입이며 Job 의 서브타입

```
val job2: CompletableJob = Job()
```

Job

- Job 객체의 함수를 이용해 코루틴에의해 실행되는 업무를 취소시키거나 종료될 때 까지 대기

  - start() : 코루틴 시작
  - join() : 코루틴이 종료될 때까지 대기
  - cancel() : 코루틴 취소
  - cancelAndJoin() : 코루틴 취소, 취소될 때까지 대기
  - cancelChildren() : 하위 코루틴 취소

CompletableJob

- CompletableJob 은 Job 의 서브 타입
- complete() 와 completeExceptionally() 함수 두개가 추가
- complete() 함수를 이용해 Job 이 완료되었음을 선언
- completeExceptionally() 함수를 이용해 Job 에 예외를 발생시키고 완료되었음으로 선언

```
var isCompleted = job3.complete()
```

CompletableJob

- join() 함수 사용할 때 주의할 점
- CompletableJob 은 join() 함수를 사용할 수는 있지만 어디선가 cancel() 혹은 complete() 함수가 호출되지 않는 한 코루틴의 작업이 완료되지 않기 때문에 join() 아랫줄이 실행되지 않을 수 있다.

CompletableJob

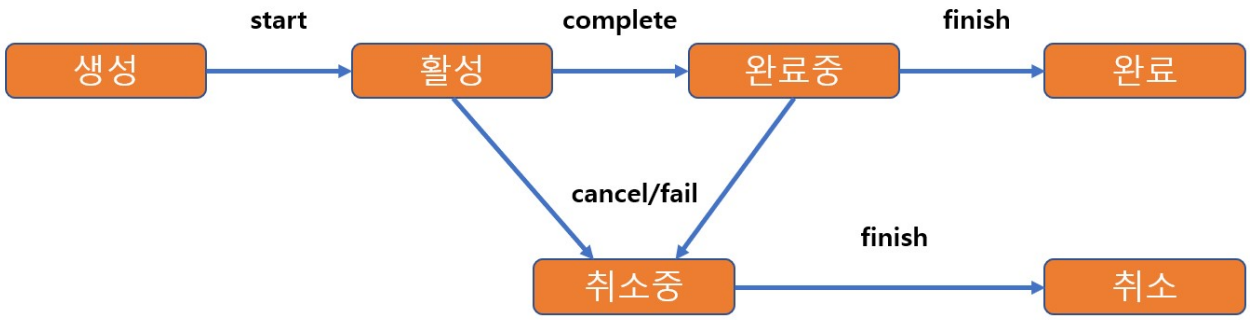
- CompletableJob 을 사용하는 이유는 코루틴 외부에서 코루틴이 완료되었음을 명시적으로 선언
- complete() 함수로 실행중인 코루틴을 완료시킬 수 있지만 한번 완료된 CompletableJob 을 이용해 다른 코루틴을 구동시킬 수는 없다.
- complete()는 리턴 값이 Boolean 형인데 true 가 리턴되면 정상적으로 코루틴 실행이 완료된 것이며 이미 완료된 Job 이면 false 가 리턴

Job Lifecycle

- Job 이 가지는 상태는 생성상태, 활성상태, 완료중인 상태, 완료상태, 취소중인 상태, 취소상태

- 생성상태는 코루틴이 만들어졌지만 아직 실행되지 않은 상태
- 활성상태는 코루틴이 동작하고 있는 상태
- 취소중, 취소 상태는 활성상태의 코루틴을 어디선가 취소 시켰거나 코루틴에서 예외가 발생해서 종료되는 상태
- 완료중, 완료 상태는 코루틴의 업무가 정상적으로 처리되어 종료되는 상태

Job Lifecycle



Job Lifecycle

- 코루틴 내부 혹은 외부에서 코루틴이 어떤 상태에 있는지를 판단할 필요가 있는데 이를 위해 isActive, isCompleted, isCancelled 프로퍼티를 제공

상태	isActive	isCompleted	isCancelled
New	false	false	false
Active	true	false	false
Completing	true	false	false
Cancelling	false	false	true
Cancelled	false	true	true
Completed	false	true	false

Job cancel

- 실행중인 코루틴을 취소시키려면 Job 의 cancel(), cancelAndJoin() 등의 함수를 이용
- 함수가 호출되었다고 코루틴이 취소되지는 않는다.

```
job.cancelAndJoin()
```

Job cancel

- 코루틴 내부에서 자신이 cancel 된 것인지를 판단해서 cancel 되었다면 예외를 발생시켜 더 이상 실행되지 않게 해주어야 취소 됨.
- delay() 같은 함수에서 코루틴이 취소 된 것인지를 판단해 예외를 발생시켜 주었기 때문에 cancel() 에 의해 코루틴이 취소되었던 것.
- kotlinx.coroutines 에서 정의한 suspend 함수는 delay() 처럼 코루틴이 취소된 것인지를 판단해 예외를 발생

- 코루틴 내부에서 취소 명령이 내려진 것인지를 판단하면 되고 이를 위해 Job 에는 isActive 라는 프로퍼티를 제공

Job cancel

- isActive 로 판단해 예외를 발생시켜 코루틴이 취소될 때 마지막으로 처리해야할 코드
- try – finally 구문을 이용
- finally 부분에서 또다시 cancel() 에 의해 예외가 발생되는 코드가 작성된 경우 finally 부분이 정상적으로 실행되지 않는 문제가 발생할 수 있음

Job cancel

- 코루틴이 취소되어 예외가 발생 하더라고 정상적으로 실행되어야 하는 부분이 있다면

  withContext(NonCancellable) { } 을 이용

```
withContext(NonCancellable) {
    ……..
}
```

Job 예외처리

- Job 이 실행이 완료되거나 예외가 발생했을 경우 마지막으로 실행되어야 할 로직
- Job 의 invokeOnCompletion() 으로 함수를 등록하여 자동으로 실행

```
job.invokeOnCompletion { e ->
    ……
}
```

async 와 Deferred

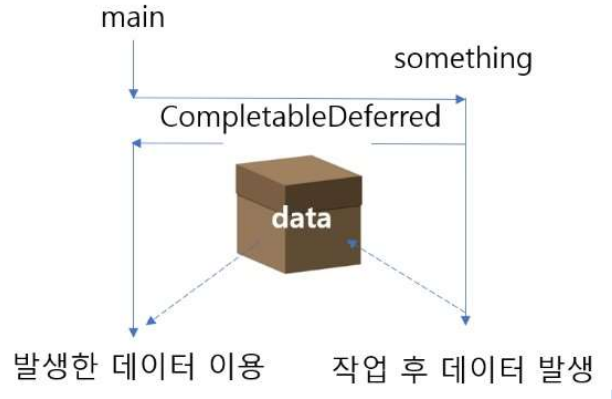
- launch() 에 의해 실행된 값을 코루틴 외부에서 획득하는 방법을 제공하지 않음
- 코루틴을 실행시킨 곳에서 데이터를 획득해 이용해야 하는 경우 async() 이용

- async() 는 코루틴 빌더이며 async() 함수의 리턴 타입은 Deferred
- Deferred 는 Job 의 서브 타입임으로 Job 에서 제공하는 함수이외에 코루틴의 결과 데이터를 획득할 수 있는 await() 함수를 제공

CompletableDeferred

- Deferred 서브 타입으로 CompletableDeferred 가 제공

- Deferred 는 미래에 발생하는 데이터를 표현하는 타입이며 이를 통해 데이터를 주고 받기 위해 이용
- CompletableDeferred 는 async() 로 코루틴을 이용하지 않는 경우에도 미래에 발생하는 데이터를 표현하고, 그 데이터가 발생했을 때 데이터를 획득하기 위해 만든 타입



Deferred 예외처리

- launch() 와 다르게 async() 는 코루틴이 실행되고 예외가 발생한다고 해서 등록한 CoroutineExceptionHandler 가 실행되지 않음.

- 실행시킨 곳에서 Deferred를 이용해 결과를 대기하지 않는 한 예외 발생 상태를 감지할 필요가 없다고 판단하여 자동으로 예외를 전파시키지 않음.

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare