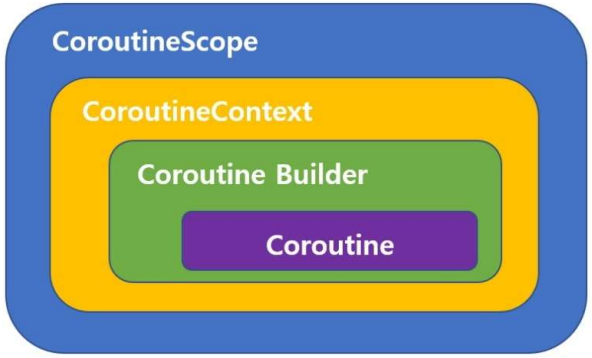
01

01-2. CoroutineScope

Coroutine

CoroutineScope

- 코루틴이 실행되는 영역
- 스코프를 선언하고 그 스코프에 CoroutineContext 를 지정
- 스코프에서 CoroutineContext 정보를 이용하여 코루틴이 만들어지고 실행



CoroutineScope

- 코루틴 빌더(launch, async등)와 스코핑 함수(scoping function – coroutineScope, withContext 등)

  는 모두 스코프가 지정되어야 하며 그 스코프내에서 실행

```
fun main() = runBlocking{ this: CoroutineScope
    launch { this: CoroutineScope
        println("hello coroutine..")
    }
}
```

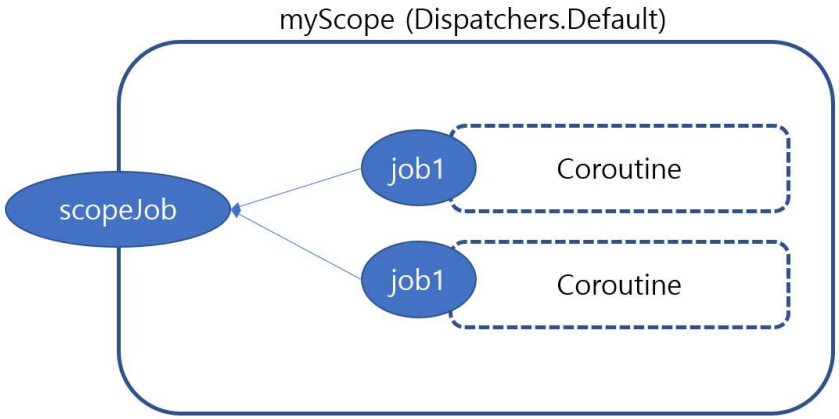
CoroutineScope

- 스코프는 CoroutineScope 인터페이스를 구현한 클래스의 객체

```
interface CoroutineScope {
    val coroutineContext: CoroutineContext
}
```

CoroutineScope

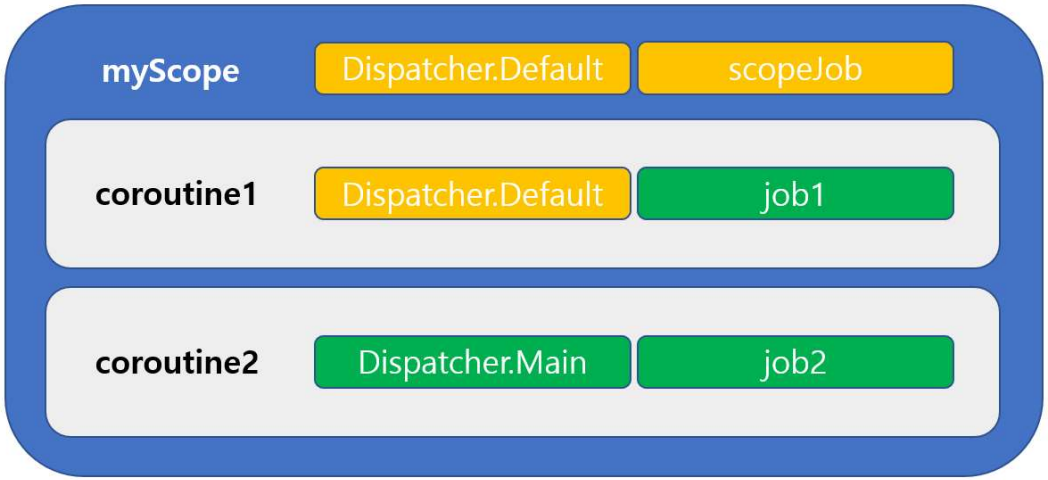
- 스코프는 코루틴을 구조화하기 위한 목적
- 스코프내에서 여러 코루틴을 실행시킴으로서 동일한 설정이 적용되게 할 수 있으며 한꺼번에 제어



CoroutineContext

- CoroutineContext 는 코루틴을 위한 설정 정보
- CoroutineContext 는 스코프에 선언하여 해당 스코프에서 실행되는 코루틴을 위한 공통의 설정도 가능하며 개별 코루틴에 설정하는 것도 가능
  - Job : 코루틴 생명주기 제어
  - CoroutineDispatcher : 코루틴이 실행될 스레드 지정
  - CoroutineName : 코루틴 이름
  - CoroutineExceptionHandler : 예외처리
- 여러가지 설정을 하고자 한다면 + 연산자를 이용

CoroutineContext



CoroutineDispatcher

- CoroutineContext 에 설정되는 정보로 코루틴이 어떤 스레드에서 동작해야 하는지를 명시하기 위해 사용
- CoroutineContext에 CoroutineDispatcher 가 지정되지 않았다면 기본으로 Dispatchers.Default 가 적용
- 다른 코루틴(부모 코루틴) 내에서 실행되는 코루틴이라면(자식 코루틴) 부모 코루틴의 CoroutineContext 정보를 이용하게 됨으로 CoroutineDispatcher 를 지정하지 않았다면 부모 코루틴에서 사용한 CoroutineDispatcher 가 적용

CoroutineDispatcher

newSingleThreadContext(), newFixedThreadPoolContext()

- 코루틴을 위해 단일 스레드를 만들고 그 스레드에서 실행되게 하고자 한다면

  newSingleThreadContext() 을 이용
- 새로운 스레드 풀을 만들고 코루틴이 그 스레드 풀에서 실행되게 하고 싶다면

  newFixedThreadPoolContext() 를 이용

Dispatchers

- Dispatchers 란 코루틴이 어느 스레드 혹은 스레드 풀에서 동작해야 하는지를 명시하기 위한 프로퍼티가 선언된 객체
- Dispatchers.Main, Dispatchers.Default, Dispatchers.IO, Dispatchers.Unconfined 중 하나로 CoroutineDispatcher 를 설정

- Dispatchers.Main 은 앱을 실행시킨 스레드를 지칭
- 안드로이드 앱처럼 화면이 있는 경우 화면 출력을 하는 Main 스레드를 지칭

Dispatchers

- Dispatchers.Default 는 지속적으로 CPU 를 점유해 빠른 연산이 필요한 작업을 위해 설계
- Dispatchers.Default 의 경우 JVM 의 공유 스레드 풀을 사용하며 최대 스레드 수는 CPU 코어 수와 동일하며 최소는 2개

- Dispatchers.IO 의 경우는 네트워크, 데이터베이스 연동, 파일 작업등 대기시간이 있는 작업을 위해 설계
- Dispatchers.IO 의 경우 필요에 따라 스레드를 더 만들거나 줄이게 되며 최대 64개

Dispatchers

- Dispatcher.Unconfined 는 특정 스레드를 국한하지 않을 때 사용
- Dispatcher.Unconfined 로 지정하면 코루틴을 시작시키는 스레드에 의해 실행되지만 일시 중단함수 이후에 코루틴이 실행될 때는 가능한 모든 스레드에서 재개

다양한 CoroutineScope - GlobalScope

- GlobalScope 는 싱글톤으로 유지되는 CoroutineScope 타입의 객체
- 애플리케이션과 동일한 생명 주기
- GlobalScope 는 애플리케이션과 생존주기가 동일한 일종의 데몬 스레드처럼 동작하는 코루틴을 위해서만 제한적으로 사용해 주기를 권장

```
GlobalScope.launch {
    repeat(5){
        println("step1...  : ${Thread.currentThread().name}")
    }
}
```

다양한 CoroutineScope – CoroutineScope()

- 매개변수에 지정되는 CoroutineContext 정보를 가지는 스코프를 만들어 주는 함수
- 함수로 간단하게 스코프를 정의하기 위해 사용

fun CoroutineScope(context: CoroutineContext): CoroutineScope

```
val scope = CoroutineScope(Dispatchers.Default)
scope.launch {
    println(".....")
}
```

다양한 CoroutineScope – CoroutineScope()

- CoroutineScope() 도 액티비티/프레그먼트등이 종료될때 이 스코프에서 실행된 코루틴이 자동 종료되지 않는다.
- 메모리 누수 가능성 있다.

특성	GlobalScope	CoroutineScope()
특성	싱글톤, 전역	인스턴스 생성 필요
생명주기	앱 전체	개발자 관리
컨텍스트	기본(제한적)	완전 커스터마이징
취소	개별 작업만	전체 스코프 가능
구조화된 동시성	지원 안 함	지원
안드로이드 권장	거의 사용 안 함	생명주기 인식 변형

다양한 CoroutineScope – MainScope()

- MainScope() 는 스코프를 만드는 함수
- CoroutineContext 은 SupervisorJob 과 Dispatchers.Main 로 고정
- 안드로이드 같은 UI 를 주 목적으로 하는 곳에서 간단하게 스코프를 정의해 사용하기 위한 목적으로 제공되는 함수

```
val scope1 = MainScope()
scope1.launch {
    //do something....
}
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare