01-4. Scope Function

Coroutine

스코프 함수

- Coroutine scope function(혹은 scoping function)
- 코루틴에의해 실행되는 특정 영역을 만들기 위해서 선언
- 코루틴 내에서 선언되어야 하며 non-blocking suspension 으로 실행된다.

- 코루틴 내에서 특정 목적을 가지는 영역 선언이 주 목적
- 에러처리 혹은 context 교체등이 주 목적

- coroutineScope(), supervisorScope(), withContext(), withTimeout()

coroutineScope()

- 대표적인 스코프 함수가 coroutineScope()
- coroutineScope() 로 만든 스코프는 외부(그 부분을 호출했던 코루틴) 스코프의 coroutineContext 를 상속, context 를 지정할 수 없다.
- coroutineScope() 내의 모든 작업이 완료될 때까지 기다리는 범위를 위해 이용
- 자식 코루틴에서 예외가 발생하면 전 범위의 코루틴이 취소
- 관련된 여러 작업을 코루틴으로 병렬 실행하고 모든 실행이 끝날때까지 기다릴 때 유용

coroutineScope()

```
suspend fun loadDashboardData() = coroutineScope {
    try {
        val userData = async { userRepository.getUserProfile() }
        val notifications = async { notificationService.getLatest() }
        val analytics = async { analyticsService.getDailyStats() }

        Dashboard(
            user = userData.await(),
            notifications = notifications.await(),
            stats = analytics.await()
        )
    } catch (e: Exception) {
        Log.e("Dashboard", "데이터 로드 실패", e)
    }
}
```

supervisorScope()

- 스코프에서 여러 개의 코루틴이 실행될 수 있으며 하나의 코루틴이 예외가 발생하게 되면 같은 스코프에서 동작하는 모든 코루틴이 종료
- 동일 스코프의 코루틴들이라고 하더라도 하나의 코루틴이 예외가 발생한 경우 나머지 코루틴이 정상적으로 실행되게 만들고 싶을 때 supervisorScope() 이용

supervisorScope()

```
suspend fun loadMultipleResources() = supervisorScope {
    val users = launch {
        try {
            // 이 작업이 실패해도 다른 작업은 계속 실행됨
            loadUsers()
        } catch (e: Exception) {
            Log.e("Resource", "사용자 로드 실패: ${e.message}")
        }
    }

    val products = launch {
        try {
            loadProducts()
        } catch (e: Exception) {
            Log.e("Resource", "상품 로드 실패: ${e.message}")
        }
    }
}
```

withContext()

- withContext() 는 스코프를 선언하면서 별도의 CoroutineContext 를 선언 가능.
- CoroutineContext 선언하여 외부 CoroutineContext 의 설정 중 일부를 withContext() 에서 교체
- 모든 코루틴 종료될 때까지 대기, 하나의 코루틴 취소되면 전체 취소

```
withContext(Dispatchers.IO){

}
```

withContext()

특성	coroutineScope()	supervisorScope()	withContext()
주요 목적	구조화된 동시성 보장	독립적인 작업 관리	컨텍스트 전환
컨텍스트 지정	불가능 (부모 상속)	불가능 (부모 상속)	가능 (파라미터로 전달)
예외 처리	자식 실패 시 모든 자식 취소	자식 실패해도 다른 자식 계속 실행	자식 실패 시 모든 자식 취소
부모 취소 시	모든 자식 취소	모든 자식 취소	모든 자식 취소
완료 시점	모든 자식 완료 시	모든 자식 완료 시	모든 자식 완료 시
새 코루틴 생성	필요 (launch/async)	필요 (launch/async)	필요 없음 (컨텍스트만 변경)
반환 값	마지막 표현식	마지막 표현식	마지막 표현식
대표적 사용 사례	관련된 작업들의 원자적 실행	독립적인 작업들의 병렬 실행	특정 디스패처로 작업 전환
실패 시 동작	전체 실패 (all-or-nothing)	부분 성공 허용	전체 실패

NonCancellable

- NonCancellable 로 지정한 withContext() 영역은 어디선가 취소 명령이 발생했다고 하더라도 끝까지 정상 실행

```
withContext(NonCancellable){

}
```

withTimeout()

- 코루틴에 의해 실행되는 영역 중 timeout 을 걸고 싶을 때 사용하는 스코프 함수

```
withTimeout(1300){

}
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare