01-b. Flow

Coroutine

Flow

- Flow 는 연속적으로 발생하는 데이터의 흐름

- Flow 를 만드는 방법은 여러가지가 있는데 가장 기본적인 방법이 flow { } 으로 만드는 것
- flow { } 내에서 데이터 발행은 emit() 함수를 이용하며 이렇게 발행한 데이터를 구독하는 것은 Flow 의 collect() 함수

```
fun something(): Flow<Int> = flow{
    emit(0)
}
```

Flow

- Flow 는 Cold Stream
- 어디선가 collect() 등의 함수로 데이터를 구독해야 Flow 는 실행
- collect() 는 Blocking 으로 실행
- 만약 데이터 구독을 Non-Blocking 으로 이용하고 싶다면 launch() 등으로 코루틴을 만들어 코루틴 내에서 Flow 를 구독

```
myFlow.collect {
    println("collect 1 : $it")
}
```

Flow Cancel

- Flow는 실행을 취소할 방법을 제공하지 않는다.
- collect() 함수 등으로 이 데이터를 구독 해야 실행되며 더 이상 그 데이터를 구독 하는 곳이 없으면 자동으로 멈추게 된다.
- 특정 flow 의 데이터 발행을 취소시키려면 collect() 를 코루틴으로 돌리면서 그 코루틴을 취소

Flow Builder

- Flow Builder 는 Flow 를 만드는 함수
- flow() 이외에 flowOf(), asFlow(), channelFlow() 등이 제공

Flow Builder – flowOf()

- flowOf() 함수의 매개변수는 가변인수(vararg) 로 선언
- flowOf() 는 매개변수로 지정된 값들을 순차적으로 발행하는 Flow 를 만드는 함수

```
flowOf(1, "hello", true)
```

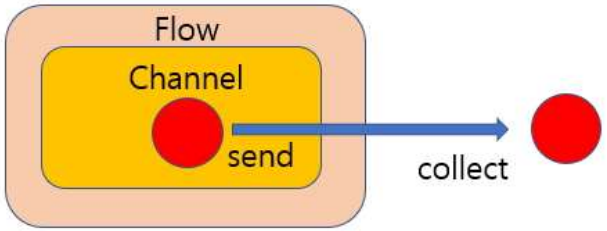
Flow Builder – asFlow()

- asFlow() 는 Array, Iterator 혹은 Sequence 의 데이터를 Flow 로 만드는 함수

```
listOf(1,2,).asFlow()
```

Flow Builder - channelFlow()

- channelFlow() 는 Channel을 이용하는 Flow 를 만드는 함수
- channelFlow { } 에 지정되는 코드는 ProducerScope 에서 실행됨으로 { } 내에서 Channel의 send() 함수를 이용해 데이터를 발행 가능



- channelFlow() 이 사용되는 주 목적은 코루틴에서 발생되는 데이터를Flow 로 처리하기 위해서 사용

MutableStateFlow

- Flow 의 하위 타입으로 MutableStateFlow 와 MutableSharedFlow 를 제공
- Flow 는 Cold Stream 이며 MutableStateFlow 와 MutableSharedFlow 은 Hot Stream

- Hot Stream 은 Cold Stream 의 반대 개념으로 어디선가 실행시키지 않아도 데이터 발행이 가능
- 여러곳에서 데이터를 구독한다고 하더라도 처음부터 다시 실행되어 모든 데이터가 발행되는 것이 아니라 구독한 순간부터 발행되는 데이터를 이용하게 하는 스트림

MutableStateFlow

- MutableStateFlow 는 StateFlow 의 서브 타입이며 MutableStateFlow() 함수에 의해 만듬.
- MutableStateFlow 은 상태를 표현(발행)하기 위해 사용
- 초기 상태 값을 위해 MutableStateFlow() 함수의 매개변수로 값을 지정
- MutableStateFlow 의 값 발행은 emit() 함수를 이용하거나 value 프로퍼티 값을 변경
- 값 획득은 collect() 함수를 이용

```
val stateFlow = MutableStateFlow(0)
```

MutableSharedFlow

- MutableSharedFlow 는 MutableSharedFlow() 함수에 의해 만들어 지는 hot stream
- 상태 표현이 아니라 이벤트 상황의 전파를 위해 설계
- replay, extraBufferCapacity, onBufferOverflow 매개변수를 이용해 어떻게 값이 발행되어야 하는지를 설정 가능.

- replay 설정은 collect 하는 곳에서 구독할 수 있는 이전 데이터 개수
- 기본 값이 0 이며 0으로 설정 되어 있다면 이전 데이터를 구독하지 않는다는 의미

```
val sharedFlow = MutableSharedFlow<Int>(replay = 2)
```

MutableSharedFlow - extraBufferCapacity

- extraBufferCapacity 는 collect 하는 데이터 구독자가 있는 경우에만 의미가 있으며 replay 값이 1 이상인 경우에만 적용
- extraBufferCapacity 는 replay 에 의해 확보되는 버퍼 이외에 추가적으로 확보해야 하는 버퍼사이즈를 의미

MutableSharedFlow - onBufferOverflow

- onBufferOverflow 설정은 확보한 버퍼 사이즈에 데이터가 모두 담긴 상태에서 새로운 데이터를 발행해야 할 때 어떻게 할 것인지에 대한 설정
- 기본값은 SUSPEND 이며 이는 버퍼의 데이터가 구독자에 의해 구독될 때까지 emit() 은 대기상태
- DROP_OLDEST 로 설정하면 버퍼에 담긴 데이터 중 가장 오래된 데이터가 제거되면서 새로운 데이터를 발행
- DROP_LATEST 이면 최신데이터가 제거

MutableSharedFlow vs MutableStateFlow

- 초기값 유무
- 다양한 설정

- 동일한 값이 연속으로 발행되어야 하는 상황에서 다르게 동작

감사합니다

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

윌리엄 셰익스피어
William Shakespeare