WaffleStudio 2023
Rookies 21.5 Seminar

Android Seminar 4
2023.11.10 (금)

Instructor : 양주현 (@JuTaK97) / TA : 송동엽 (@eastshine2741)

오늘 배울 것

- Jetpack Compose
- Kotlin Flow

Jetpack Compose (1)

기존의 XML 기반 레이아웃 작업은?

- 코틀린 코드 외에 xml 파일을 만들고 작업해야 한다.
- 화면이 어떻게 보이는지 일일이 지정해야 한다. (== 명령형)

  예: 데이터가 바뀌면 myText.text = “newValue” 를 직접 지정한다.

Jetpack Compose는?

- 오직 코틀린 코드만으로 UI를 작성하고, 쉽게 컴포넌트를 모듈화하고 재사용할 수 있다.
- 화면이 어떻게 보여야 하는지 지정한다. (== 선언형)

  데이터가 바뀌어 UI의 변경이 필요한 경우 프레임워크에서 자동으로 UI를 업데이트한다.

Jetpack Compose (2)

가장 기본이 되는 컴포넌트들

- Row / Column
- Box
- Surface
- Spacer
- Image, Text, TextField
- LazyRow, LazyColumn

Jetpack Compose (3)

https://developer.android.com/courses/jetpack-compose/course?hl=ko

하나씩 따라해 봅시다. (레고놀이 시작)

Jetpack Compose (4)

Compose에서 가장 중요한 개념 : State (상태)

- State/MutableState는 값을 보유하고, 그 값이 변경되면 재구성(recomposition)을 트리거한다.
- Compose는 데이터가 변경된 구성 요소들만 재구성하고, 나머지는 건너 뛰는 최적화가 아주 잘 되어 있다.
- 즉, Composable 함수가 State를 subscribe하는 구조

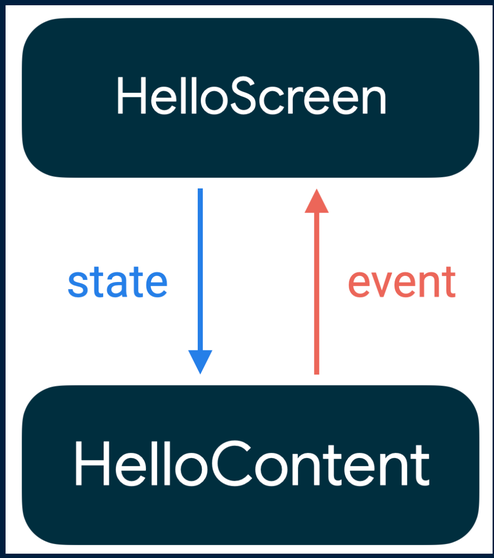
- 재구성 시 변하지 않는 값 만들기 : remember

  remember의 key란? = calculation 람다 블록을 다시 실행할 트리거를 지정

Jetpack Compose (5)

상태 호이스팅 (State Hoisting)

- 컴포저블 함수가 내부에 상태를 가지고 있으면 “Stateful”, 없으면 “Stateless”
- Stateful 하게 되면 재사용성이 낮아진다.



- 모든 State는 상위에서 주입받는다! == 상태 호이스팅
- 컴포저블 함수는 오직 상태에 따라서 UI를 그릴 뿐이고, 상태를 저장하고 변경하는 부분과 분리된다
- state는 위에서 아래로, event는 아래에서 위로 (단방향 데이터 흐름) == 이미 익숙한 개념!
- 상태를 읽는 곳은 최대한 아래, 상태를 쓰는(변경하는) 곳은 최대한 위

Jetpack Compose (6)

컴포즈 UI 예쁘게 만들기의 핵심 : Modifier

- 레이아웃
- 장식, 효과, 애니메이션
- 상호작용 (클릭, 드래그 등등)

이것저것 써 보면서 익히기

https://developer.android.com/jetpack/compose/modifiers-list?hl=ko

Jetpack Compose (7)

Jetpack Compose의 Navigation (탐색)

- 너무 쉬운 가이드

  https://developer.android.com/jetpack/compose/navigation?hl=ko

Jetpack Compose의 ViewModel

- viewModel (액티비티/프래그먼트 스코프) : 가이드
- hiltViewModel (위 Navigation의 NavBackStackEntry 스코프) : 가이드1 가이드2

Jetpack Compose (8)

Jetpack Compose에서 Coroutine 사용

- LaunchedEffect : remember + suspend block

  컴포저블 함수에서 최초 한번만 수행 (key를 지정하면 key에 의해서만 다시 수행)

- rememberCoroutineScope

  호출되는 Composition 지점에 bound된 CoroutineScope를 반환

  쉽게 말하면, rememberCoroutineScope를 부른 지점의 컴포저블이 dispose되면 해당 CoroutineScope로 launch한 Job들은 자동으로 모두 취소된다.

Jetpack Compose (9)

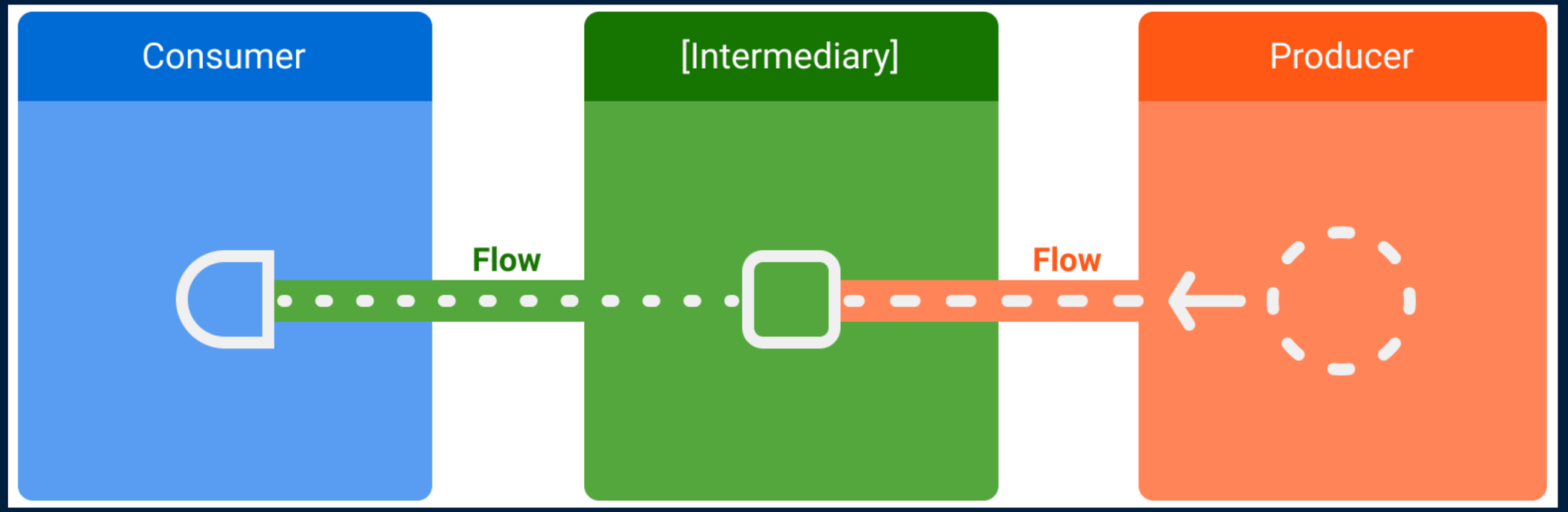
공식 홈페이지에 수많은 정보와 고급 활용 가이드가 있다.

https://developer.android.com/jetpack/compose/documentation?hl=ko

- 다음 주에 조금 더 다룰 예정

Kotlin Flow (1)

https://developer.android.com/kotlin/flow?hl=ko



Kotlin Flow (1)

기존의 suspend fun 은 단일 값만을 반환하는 반면..

Flow는 코루틴 내에서 여러 값을 순차적으로 내보낼 수 있는 데이터 스트림이다.

Iterator과의 차이점 : suspend fun을 사용해서 비동기적으로 값을 내보낼 수 있다.

```
flow {
    emit(1)
    delay(100L)
    emit(2)
    delay(100L)
}
```

Kotlin Flow (2)

한쪽에서는 flow 빌더와 emit, 혹은 iterable을 flow로 만들어 데이터 스트림을 produce

한쪽에서는 collect를 이용해 consume 한다.

```
// produce
val myFlowFlow = flow<Int> {
    emit(1)
    delay(100L)
    emit(2)
    delay(100L)
}
```

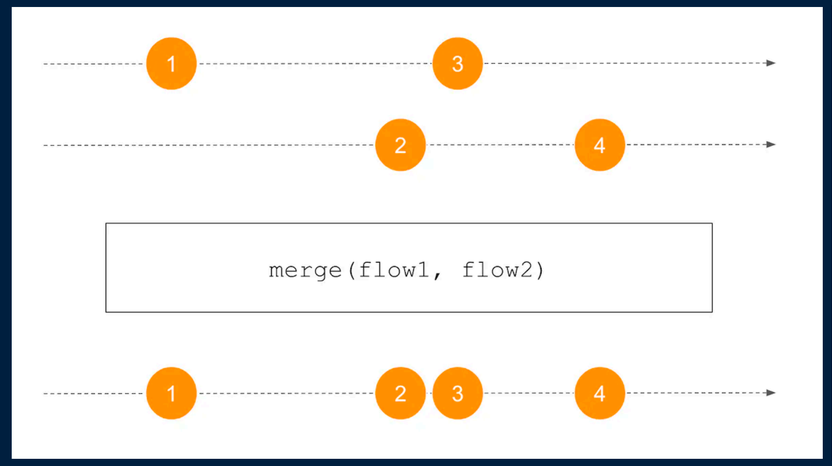
```
// consume
myFlowFlow
    .onEach {
        print(it)
    }
    .collect()

myFlowFlow.collectLatest {
    print(it)
}
```

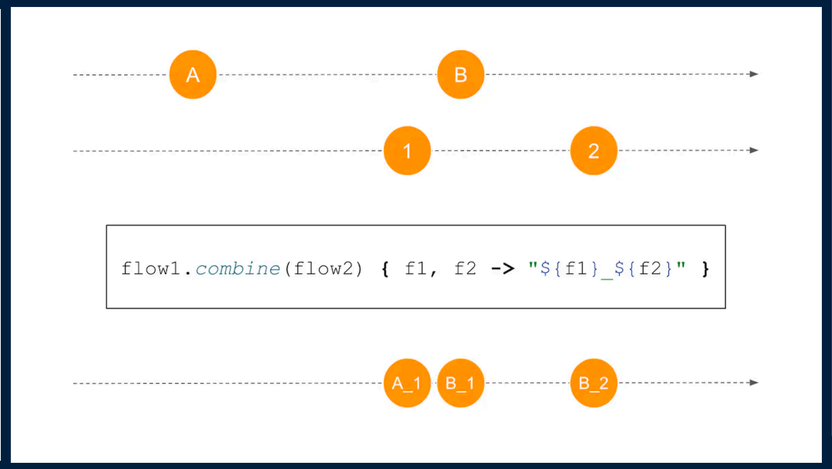
Kotlin Flow (3)

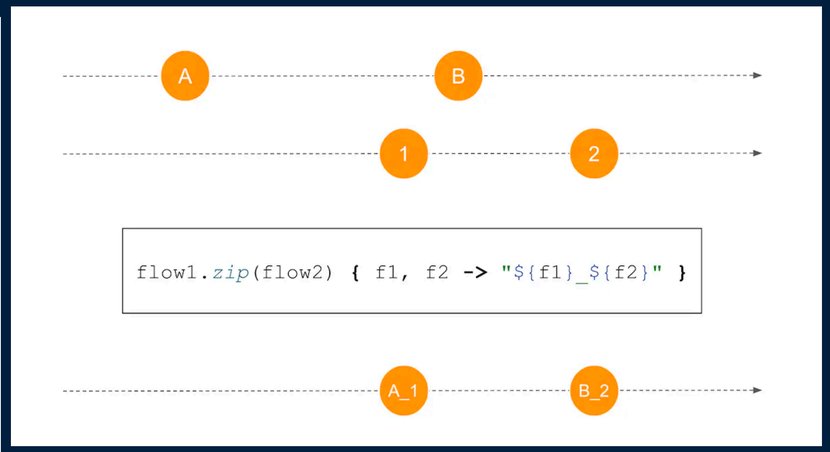
Intermediary : 가로채서 데이터를 변경한다!

map, filter 등 디양한 중간 연산자를 이용할 수 있다. (Collection의 그것들과 비슷)

combine, merge, zip 등을 이용해 여러 개의 flow를 합칠 수도 있다.







Kotlin Flow (4)

StateFlow와 SharedFlow

- Hot Stream vs Cold Stream

https://developer.android.com/kotlin/flow/stateflow-and-sharedflow?hl=ko

Kotlin Flow (5)

활용

- DataStore

  https://developer.android.com/topic/libraries/architecture/datastore?hl=ko

- Paging

  https://developer.android.com/topic/libraries/architecture/paging/v3-overview?hl=ko

- 이외에도 다양한 곳에서 Flow를 활용할 수 있습니다.