WaffleStudio Android Seminar - 3

이승민 (안드로이드 세미나장)

2021.09.25.(토) 11:30 ~

What we will learn in Seminar 2

- Asynchronous Programming
  - LiveData
  - Kotlin Coroutines
- Network
  - Basic
  - Retrofit (OkHttp)
  - Moshi
- View Holder (View Type)

Asynchronous Programming

- Thread
  - 실행되는 코드의 흐름
  - 한 프로그램에서 한 흐름의 코드만 실행되고 있지 않음
  - Ex) 유튜브
    - 댓글 로딩
    - 동영상 로딩
    - 구독 좋아요 알림설정 클릭 시 서버와 통신
  - 다양한 Thread에서 코드를 병렬적으로 실행시키는 중!

Asynchronous Programming

- Android 에서 Thread
  - 다양한 코드의 흐름들(Thread)이 UI를 한꺼번에 조작하면 동기화 이슈 발생!
  - -> MainThread에서만 UI를 바꿀 수 있게 하자
  - MainThread가 아닌 곳에서 UI 조작 시도 시 Exception 발생

- 그렇다면 전부 MainThread에서 돌리면 되나?

Asynchronous Programming

- 무거운 Task가 실행될 때, 코드의 흐름은 멈춘다!
  - PS나 코테 연습할 때, 1억 개 sorting 같은 느낌적인 느낌
  - 우리야 기다리면 되지만…
- 유저 입장에서 UI와 interact 해야하는데 터치가 안된다?!
  - 최악의 UX가 되어버림
- UI 코드의 흐름을 멈추지 않고, 따로 동작하는 코드의 흐름이 필요
  - Asynchronous Programming이 해답!

Asynchronous Programming

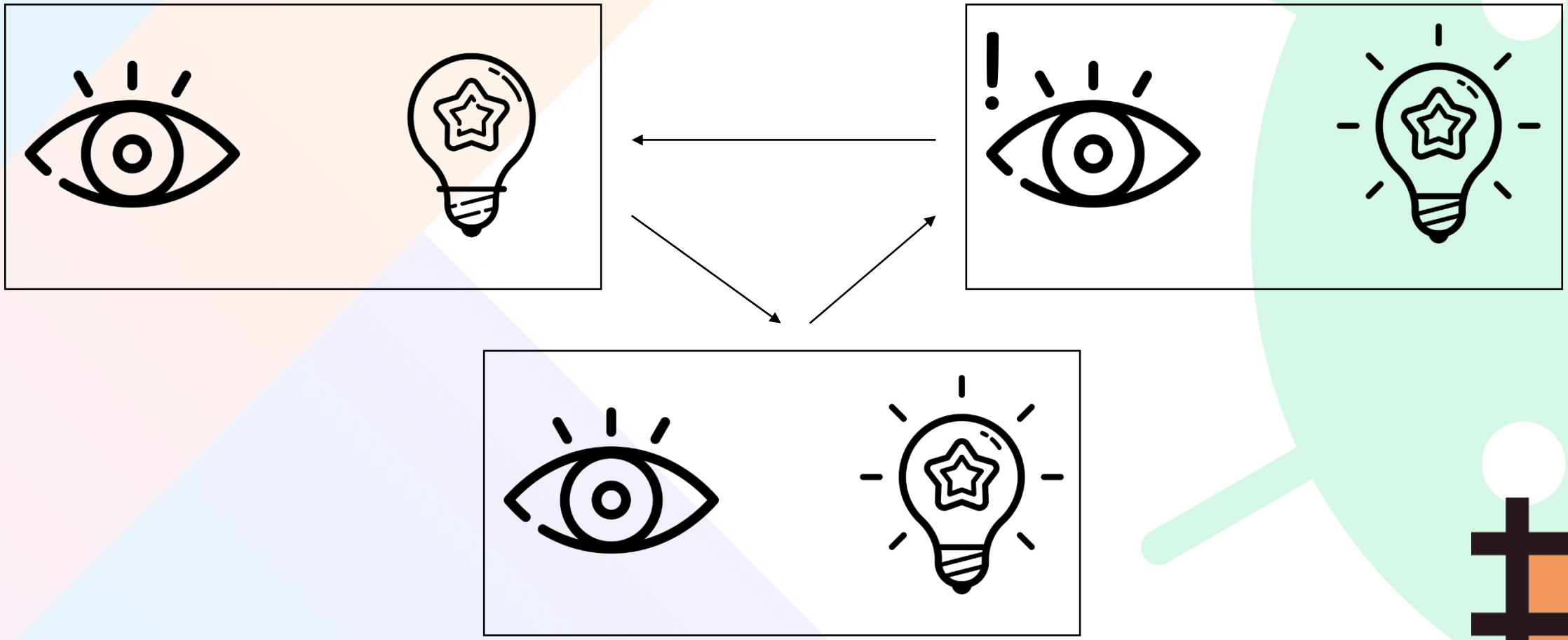
- Android OS에서 보통 무거운 Task
  - 네트워크 통신
  - Local DB 접근
  - ~~빅데이터 연산~~
  - ~~머신러닝~~

Asynchronous Programming

- ~~AsyncTask~~ (Deprecated)
- LiveData
- Kotlin Coroutines
- RxJava (Recommend Coroutines)

LiveData

- Reactive Programming 또한 Asynchronous Programming



LiveData

- 이제 다들 익숙하시죠?

```
private val _dataList: MutableLiveData<Array<Array<String>>> =
    MutableLiveData(Array( size: 3) { Array( size: 3) { MARK_BLANK } })
val dataList: LiveData<Array<Array<String>>> = _dataList
```

```
vm.dataList.observe( owner: this, {  it: Array<Array<String>>!
    for (i in 0 until 3) {
        for (j in 0 until 3) {
            buttonArray[i][j].text = it[i][j]
        }
    }
})
```

Kotlin Coroutines

- Asynchronous Programming을 사용하기 쉽게 만들어 줌
- LiveData 보다 더 많은 기능을 제공
- Observer Pattern, Data Flow 등
- vs LiveData
  - 더 쉽게 사용 : LiveData
  - 다양한 기능 : Coroutines
  - 둘 모두 적절히 사용

Kotlin Coroutines

- Flow
  - LiveData와 비슷
  - setValue(~) == emit(~)
  - observe() == collect()
  - But 더 다양한 기능 지원
    - map, combine, filter 등등

Kotlin Coroutines

- Scope
  - suspend fun 을 실행시킬 수 있는 공간
  - Activity에서 실행해야 함 -> lifecycleScope
  - ViewModel에서 실행해야 함 -> viewModelScope
  - ~~Scope.launch { suspend fun 실행 }
  - withContext를 통해 원하는 쓰레드에서 실행하도록 명령 가능

Network

- Server와 소통하기 위해서
- 클라이언트는…
  - Request를 보낸다.
  - 기다린다.
  - Response가 도착한다.
  - 정보를 읽는다.

HTTP Request

- 요청
  - GET, POST, PUT, DELETE
- URL
  - 어디에 접근할 것인가
- Query
  - 어떤 세팅으로 요청할 것인가
- Body
  - 어떤 정보를 담아서 보낼 것인가

HTTP Response

- Status Code
  - 성공 시 보통 2XX
    - 200 OK
    - 201 Created
  - 실패 시 보통 4XX, 5XX
    - 400 Bad Request
    - 403 Forbidden
    - 404 Not Found
    - 502 Bad Gateway
    - 503 Service Unavailable
- Body
  - Raw Data
    - image, json 등

OkHttp

- Http 통신을 쉽게 하도록 도와주는 library
- OkHttp가 없으면…
  - HttpUrlConnection 를 이용해서
  - Stream 열고… 정보 넣고… 등등 귀찮아짐
- OkHttp가 다 알아서 해준다!

Retrofit

- Http 통신 시 Request를 쉽게 만들어줌
- OkHttp를 내장하고 있음

Moshi

- 통신 시 Response는 json 형태로 들어옴 (단순한 txt파일)
- 이걸 Kotlin 코드로 사용할 수 있게 변환해줘야 함
  - Moshi가 해준다

Moshi

- 통신 시 Response는 json 형태로 들어옴 (단순한 txt파일)
- 이걸 Kotlin 코드로 사용할 수 있게 변환해줘야 함
  - Moshi가 해준다

How to make?

- Retrofit 객체 생성

```
private val retrofit by lazy {
    Retrofit.Builder()
        .client(httpClient)
        .baseUrl( baseUrl: "https://jsonplaceholder.typicode.com/")
        .addConverterFactory(MoshiConverterFactory.create(moshi))
        .build()
}
```

OkHttp Client

연결할 링크

사용할 Converter

How to make?

- 그 전에…
  - httpClient 생성
  - Moshi 객체 생성

```
private val moshi by lazy {
    Moshi.Builder()
        .add(KotlinJsonAdapterFactory())
        .build()
}

private val httpClient by lazy {
    OkHttpClient.Builder()
        .addInterceptor(
            HttpLoggingInterceptor().apply { this: HttpLoggingInterceptor
                level =
                    if (BuildConfig.DEBUG) HttpLoggingInterceptor.Level.BODY
                    else HttpLoggingInterceptor.Level.NONE
            }
        )
        .build()
}
```

How to make?

- Retrofit Service 생성
  - Room DB의 Dao와 비슷함

```
interface PostService {
    @GET( value: "/posts")
    suspend fun getAllPost(): List<Post>
}
```

```
private val postService by lazy {
    retrofit.create(PostService::class.java)
}
```

How to make?

- Retrofit Service 생성
  - Room DB의 Dao와 비슷함

```
interface PostService {
    @GET( value: "/posts")
    suspend fun getAllPost(): List<Post>
}
```

```
private val postService by lazy {
    retrofit.create(PostService::class.java)
}
```

ViewHolder (View Type)

- RecyclerView + α
- View Type을 이용해서 하나의 RecyclerView에 다양한 디자인을 적용 가능

```
override fun getItemViewType(position: Int): Int {
    return if (posts[position].id % 3 == 0) VIEW_TYPE_PURPLE else VIEW_TYPE_TEAL
}
```

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): RecyclerView.ViewHolder {
    return when (viewType) {
        VIEW_TYPE_TEAL -> {
            val binding = ItemPostTealBinding.inflate(LayoutInflater.from(parent.context), parent, attachToParent: false)
            PostTealViewHolder(binding)
        }
        VIEW_TYPE_PURPLE -> {
            val binding = ItemPostPurpleBinding.inflate(LayoutInflater.from(parent.context), parent, attachToParent: false)
            PostPurpleViewHolder(binding)
        }
        else -> throw IllegalStateException("viewType must be 0 or 1")
    }
}
```

QA