03-2. LiveData

Android Architecture Component

LiveData

- ViewModel 의 결과
- ViewModel 에서 String 등의 결과를 리턴 시킬 수도 있지만 LiveData 을 리턴 시켜 Observer로 결과 이용

```
class MyViewModel: ViewModel() {
    fun someData() : String {
        return "hello"
    }
    fun someData2() : MutableLiveData<String> {
        val liveData= MutableLiveData<String>()
        thread {
            SystemClock.sleep(3000)
            liveData.postValue("world")
        }
        return liveData
    }
}
```

Observer

- LiveData의 변경을 감지하는 observer는 Observer 를 구현한 클래스
- LiveData 의 값이 변경되면 Observer 의 onChanged() 함수가 자동 호출

```
val observer = object : Observer<String> {
    override fun onChanged(t: String?) {
        Log.d("kkang","onChanged.....$t")
    }
}
model.someData2().observe(this, observer)
```

- 더 이상 감지가 필요 없는 경우 명시적으로 removeObservers 함수 호출

```
val liveData = model.someData2()
//.....
liveData.removeObservers(this)
```

postValue() vs setValue()

- 결과 데이터를 LiveData 에 담는 역할
- postValue는 내부적으로 background thread 에 의해 동작
- setValue는 main thread 에 의해 동작

```
class MyViewModel: ViewModel() {
    fun get_postValue(): MutableLiveData<String> {
        val result = MutableLiveData<String>()
        result.postValue("get_postValue....hello")
        return result
    }
    fun get_setValue(): MutableLiveData<String> {
        val result = MutableLiveData<String>()
        result.value = "get_setValue....hello"
        return result
    }
}
```

```
model.get_postValue().observe(this, { result ->
    Log.d("kkang", result)
})
model.get_setValue().observe(this, { result ->
    Log.d("kkang", result)
})
```

postValue() vs setValue()

- Background thread 에서 값 등록
- postValue() 의 경우 내부적으로 background thread 임으로 정상 실행
- setValue() 는 런타임 에러

```
fun get_postValue(): MutableLiveData<String> {
    val result = MutableLiveData<String>()
    thread {
        result.postValue("get_postValue....hello")
    }
    return result
}

fun get_setValue(): MutableLiveData<String> {
    val result = MutableLiveData<String>()
    thread {
        //java.lang.IllegalStateException: Cannot invoke setValue on a background thread
        result.value = "get_setValue....hello"
    }
    return result
}
```

postValue() vs setValue()

•Activity 에서 observer를 이용하지 않고 ViewModel 의 데이터를 직접 받는다면 postValue 함수에 의한 결과는 전달 안됨.

```
val liveData = model.get_postValue()
Log.d("kkang", "postValue() : ${liveData.value}")//postValue() : null

val liveData2 = model.get_setValue()
Log.d("kkang", "setValue() : ${liveData2.value}")//setValue() : get_setValue....hello
```

Custom LiveData

- LiveData 를 상속받아 작성
- ViewModel 이외에 다른 곳에서 사용 가능

```
class MyLiveData : LiveData<String>() {
    fun sayHello(name: String) {
        postValue("Hello $name")
    }
}
```

```
val liveData1 = MyLiveData()
liveData1.observe(this) {
    Log.d(
        "kkang", "result : $it"
    )
}
liveData1.sayHello("kkang")
```

LiveData Lifecycle

- onActive() 와 onInactive() 라이프 사이클 함수
- LiveData 에 대한 감지가 시작되면 onActive() 함수가 자동 호출
- 더 이상 감지하는 곳이 없으면 onInactive() 함수가 자동 호출

```
class MyLiveData : LiveData<String>() {
    fun sayHello(name: String) {
        postValue("Hello $name")
    }

    override fun onActive() {
        super.onActive()
        Log.d("kkang", "MyLiveData...onActive")
    }

    override fun onInactive() {
        super.onInactive()
        Log.d("kkang", "MyLiveData....onInactive")
    }
}
```

감사합니다

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

윌리엄 셰익스피어
William Shakespeare