SavedState ViewModelを使ってデータを保持する

Kenji Abe Following
Mar 16, 2019 · 4 min read

`ViewModel` を使うことで画面回転等のConfigurationが変更されたときもデータを保持することが可能になりました。しかし、OSからプロセスKillされてActivityの再生成に対応できません。
この場合は `onSaveInstanceState` を使うことで対応することになります。

ほとんどケースでUIの状態はViewModelと多いので、このあたりもViewModelのほうで対応できたらだいぶ楽になると思います。

これを `SavedStateHandle` というのを使って保存と取得を行えるようになりました。

まだalpha01なので今後変更はあるかもしれません。

環境

```
implementation 'androidx.lifecycle:lifecycle-viewmodel-savedstate:1.0.0-alpha01'
```

実装

ViewModel定義

ViewModel自体はコンストラクタに `SavedStateHandle` もらうようにするだけです。

ViewModel生成

普通のViewModelの生成と変わらないのですが、Factoryに `SavedStateVMFactory` を使います。

これだけで準備完了です。

データを保存と取得

保存と取得は非常に簡単で、 `SavedStateHandle` を上記のように使うだけです。これでプロセスがKillされてもsetした値はActivity再生成後も取得できます。

LiveDataを保存と取得

`SavedStateHandle` は更にLiveDataに対応しています。

`getLiveData` を使うことでLiveDataとして取得することが可能です。更に値をsetしたときにはsetValueと同じで、observeへ通知がいきます。

注意: コードを見る限りLiveData#setValueを使ってるのでBackgroudスレッド等からは呼び出さないようしてください。

扱えるデータ型

`Bundle` に保存できるものと同じです。

おまけ: Daggerを使う場合

Daggerを使う場合はいつものようなViewModelFactoryが使えません。ちょっと別のアプローチが必要です。

例えば、次のようにViewModelにRepositoryを渡したい場合です。

```
class MainViewModel(
    private val savedStateHandle: SavedStateHandle,
    private val userRepository: UserRepository
) : ViewModel() {

}
```

MainViewModel.kt hosted with ❤ by GitHub — view raw

おまけ: Daggerを使う場合

Daggerを使う場合はいつものようなViewModelFactoryが使えません。ちょっと別のアプローチが必要です。

例えば、次のようにViewModelにRepositoryを渡したい場合です。

```
class MainViewModel(
    private val savedStateHandle: SavedStateHandle,
    private val userRepository: UserRepository
) : ViewModel() {

}
```

MainViewModel.kt hosted with ❤ by GitHub — view raw

この場合はViewModelごとにFactoryを作る必要があります。以下のような感じです。

```
class MainViewModelFactory @Inject constructor(
    owner: SavedStateRegistryOwner,
    private val userRepository: UserRepository
) : AbstractSavedStateVMFactory(owner, null) {

    override fun <T : ViewModel?> create(key: String, modelClass: Class<T>, handle: SavedStateHandle): T {
        @Suppress("UNCHECKED_CAST")
        return MainViewModel(handle, userRepository) as T
    }
}
```

MainViewModelFactory.kt hosted with ❤ by GitHub — view raw

`AbstractSavedStateVMFactory` を継承したFactoryを作ります。このコンストラクタには、 `SavedStateRegistryOwner` が必要なります。これはActivityとFragmentが実装してるので、それを渡せば大丈夫です。

Moduleでその `SavedStateRegistryOwner` を解決できるようにしてあげます。

```
@Module
class FeatureModule(private val owner: SavedStateRegistryOwner) {
    @FeatureScope
    @Provides
    fun provideOwner() = owner
}
```

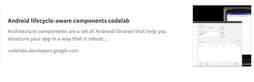
FeatureModule.kt hosted with ❤ by GitHub — view raw

実際は他にも色々必要ですが、最終的にActivityはこんな感じになります。

```
class MainActivity : AppCompatActivity() {

    @Inject
    lateinit var viewModelFactory: MainViewModelFactory

    private val viewModel by lazy {
        // injectされたViewModelFactoryを使ってViewModelを作る
        ViewModelProviders.of(this, viewModelFactory).get(MainViewModel::class.java)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val appComponent = (application as App).appComponent
        val featureComponent = DaggerFeatureComponent.builder()
            .appComponent(appComponent)
            .featureModule(FeatureModule(this))
            .build()
        // ViewModelFactoryをinject
        featureComponent.inject(this)

        viewModel.value.observe(this) {
            // ...
        }
    }
}
```

MainActivity.kt hosted with ❤ by GitHub — view raw

参考



Android

WRITTEN BY
Kenji Abe
Following
Programmer / Gamer / Google Developers Expert for Android, Kotlin / @STAR_ZERO