Android MVC , MVP, MVVM 架構案例學習

蒼耳叔叔 鴻洋 2022-03-29 08:35

本文作者

作者：**蒼耳叔叔**

鏈接：

https://juejin.cn/post/7067733584865394718

1 前言

這是架構學習系列的第三篇，主要介紹一下MVC, MVP 以及MVVM 架構，至於MVI 後面會單獨介紹。這些MVX 的目的都是為了將業務和視圖分離，松耦合，作為Android 程序猿，大多不陌生了。

> 一個App 離不開Model 和View 這兩個角色, Model 決定了App 的數據，而View 決定怎麼向用戶展示這些數據，大多框架或組件基本上都是用來處理這兩者之間的交互關係的。

因此一個App 的架構需要處理兩個任務：

1. 更新Model —— 如何處理View action?

2. 更新View —— 如何將Model 的數據表現到View 上?

基於此，在Android 上一般有如下三種常用架構(本期不講MVI)：

MVC —— Model-View-Controller: 作為Controller 層的Actvity/Fragment 等充當了View 的角色，代碼過於臃腫；同時在View 層又容易直接操作Model，導致View 和Model 層耦合，無法獨立復用。有時候看到一個Activity 能有幾千甚至上萬行的代碼，簡直噩夢。

MVP —— Model-View-Presenter: Presenter 和View 層之間通過定義接口實現通信，解耦了View 和Model 層。然而當業務場景比較複雜時，接口定義會越來越多，且可能定義模糊，接口一旦變化，對應實現也需要發生變化。

MVVM —— Model-View-ViewModel: MVVM 解決了MVP 的問題，使得ViewModel 和View 之間不再依賴接口通信，而是通過LiveData, RxJava, Flow 等響應式開發的方式來通信。

我們在這裡可以看下Model 和View 的理解：

View：視圖，向用戶呈現的界面，與用戶直接交互的一層。

Model：Model 通常應包括數據和一些業務邏輯，即數據的結構定義，以及存儲和獲取等。而針對外部組件而言, Model 往往表示向其提供的數據，畢竟它們不關心數據是咋來的，咋走的，它們只關心它們自己。

2 MVC

該架構涉及三個角色: Model-View-Controller。其中Controller 是Model 與View 之間的橋樑，用來控製程序的流程。

我記得曾經在網上看過不少MVC 的文章，但是貌似有些文章裡面給的模型圖不太一樣，一度有些費解，其實這些不一樣的地方在於MVC 模型經過發展存在著變體而已。一個版本的MVC 是這樣子的：

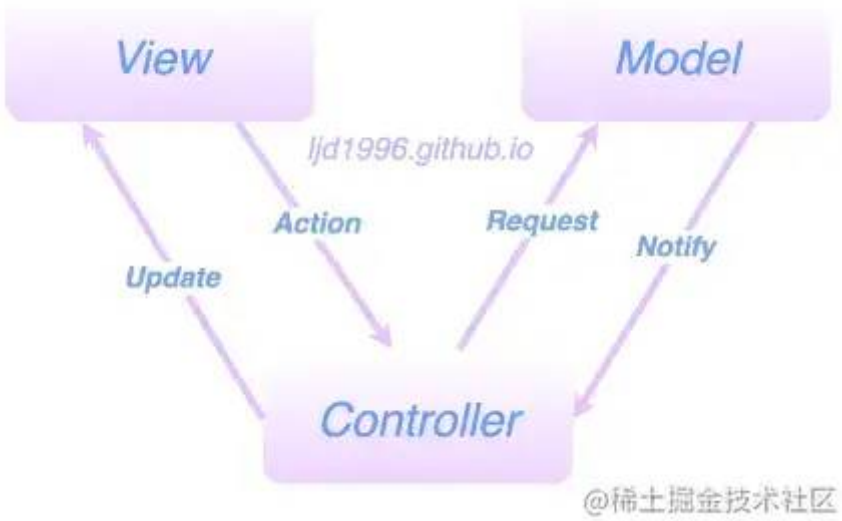
該版本一般的交互流程是：

1. 用戶操作View, 比如說產生了一個點擊事件。

2. Controller 接收事件，對其作出反應。比如說是點擊登錄事件，它會校驗用戶輸入是否為空，若為空則直接返回View 讓其提示用戶；若不為空則請求Model 層。

3. Model 作出處理後，需要把登錄用戶的數據通知到相關成員，上圖中即是View 層。View 收到後作出相關展示。

在上圖中View 層依賴了Model 層，降低了View 的可複用性，為了解耦，出現了下圖的版本：



這個版本的主要改動就是View 和Model 不直接通信了，View 通過Controller 去更新Model 層的數據，Model 層完成邏輯後通知Controller 層，Controller 再去更新View。

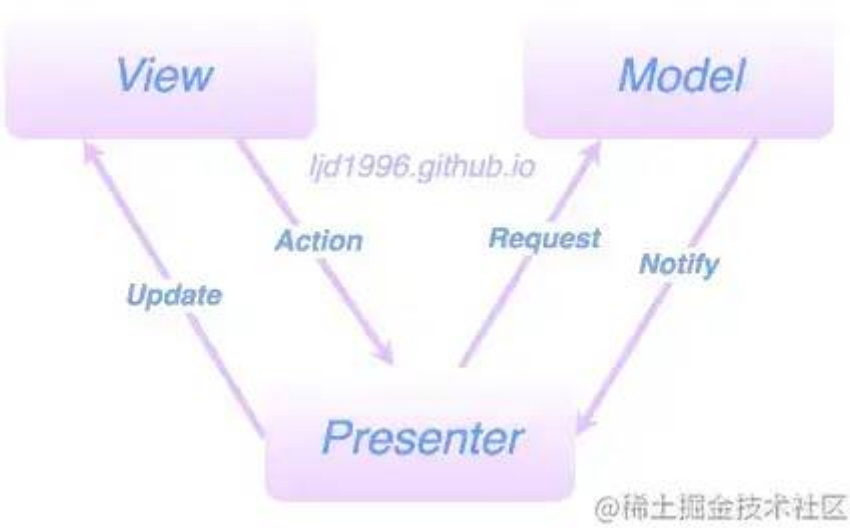
MVC 架構小結

MVC 為視圖和業務的分離提供了開創性的思路，解耦了View 和Model 層，提高了復用性。

然而在Android 的實際應用中, 容易出現一個新的角色—— ViewController, 比如說Activity 又當View 又當Controller 的，十分臃腫，耦合也隨之變得嚴重了起來，還不方便單元測試。

3 MVP

該架構涉及三個角色: Model-View-Presenter。關係圖如下：



這張圖跟上面第二個版本的MVC 結構很像，不一樣的地方在於Controller 換成了Presenter 層，其職責是類似的，但是實現方式不一樣。MVP 之間是通過接口來通信的，三個層都有各自的接口來定義其行為與能力，這樣可以降低耦合，提高複用性，也方便了單元測試。

其交互流程依舊是：**用戶操作View 層，產生了一個事件; Presenter 接收事件，並對其作出反應，請求Model 層; Model 層作出處理後通知給Presenter, Presenter 進而再通知到View 層。**

> 通過登錄場景舉個栗子🌰

1、首先定義各層的接口，一個場景的接口寫在一起。

```
interface ILogin {
  interface ILoginView {
    fun loginLoading() // 登陆中
    fun loginResult(result: Boolean) // 登陆结果
    fun isAvailable(): Boolean // IView 是否可用
  }

  interface ILoginPresenter {
    fun attachView(view: ILoginView) // attach View
    fun detachView() // detach View, 防止内存泄漏
    fun isViewAvailable(): Boolean
    fun login()
  }

  interface ILoginModel {
    fun login(listener: OnLoginListener)
  }

  interface OnLoginListener {
    fun result(result: Boolean)
  }
}
```

2、View 層實現。

```
class MVPLoginActivity : AppCompatActivity(), ILogin.ILoginView {
  private val loginPresenter = LoginPresenter()

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(Button(this).apply {
      text = "登录"
      setOnClickListener {
        loginPresenter.login()
      }
    })
    loginPresenter.attachView(this)
  }

  override fun loginLoading() {
    Toast.makeText(this, "Login...", Toast.LENGTH_SHORT).show()
  }

  override fun loginResult(result: Boolean) {
    Toast.makeText(this, "Login result: $result", Toast.LENGTH_SHORT).show()
  }

  override fun isAvailable() = !isDestroyed && !isFinishing

  override fun onDestroy() {
    super.onDestroy()
    loginPresenter.detachView()
  }
}
```

3、Presenter 層實現。

```
class LoginPresenter : ILogin.ILoginPresenter, ILogin.OnLoginListener {
  private val loginModel: ILogin.ILoginModel = LoginModel()
  private var loginView: ILogin.ILoginView? = null

  override fun attachView(view: ILogin.ILoginView) {
    loginView = view
  }

  override fun detachView() {
    loginView = null
  }

  override fun isViewAvailable(): Boolean = loginView?.isAvailable() ?: false

  override fun login() {
    loginView?.loginLoading()
    loginModel.login(this)
  }

  override fun result(result: Boolean) {
    if (isViewAvailable()) {
      loginView?.loginResult(result)
    }
  }
}
```

4、Model 層實現。

```
class LoginModel : ILogin.ILoginModel {
```

```
    override fun login(listener: ILogin.OnLoginListener) {
        thread {
            Thread.sleep(1000)
            runOnUIThread {
                // 返回登录结果
                listener.result(Random.nextBoolean())
            }
        }
    }
}
```

以上只是一個示例，實際開發中當然會把一些基礎的重複的邏輯抽成Base 類。

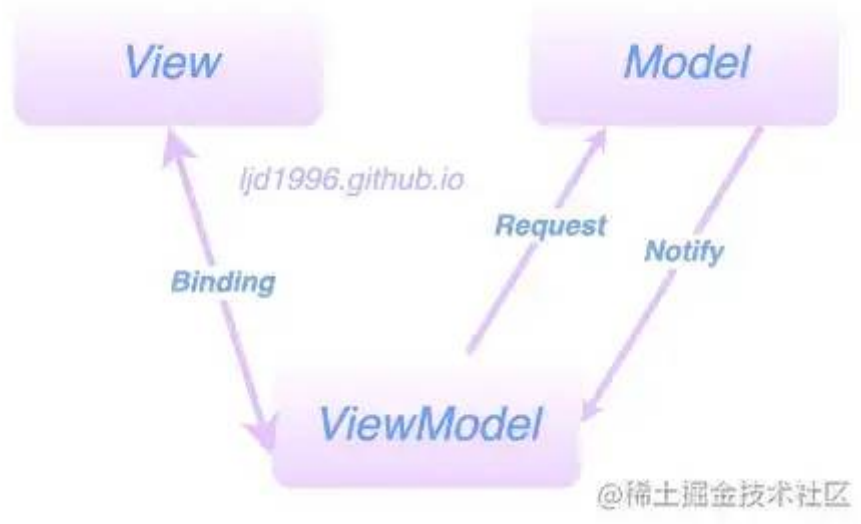
MVP 架構小結

- MVP 模式清晰劃分了各個層的職責，避免了ViewController 的問題，降低了代碼的臃腫程度。
- 解除View 與Model 的耦合，通過接口來交互，提高了可複用性和擴展性，利於單元測試。
- 但隨著業務的複雜化，接口的定義越來越多，提高了項目的複雜度，對開發的設計能力要求也更高了。
- Presenter 如果持有Activity 等的引用，容易出現內存洩漏，生命週期不同步等問題。

4 MVVM

MVVM模式

該架構涉及三個角色: Model-View-ViewModel。關係圖如下:



> 它跟MVP 看起來也是比較類似的，不同之處在於Presenter 換成了ViewModel, ViewModel 負責與Model 層交互，並且將數據以可觀察對象的形式提供給View, ViewModel 與View 層分離，即

> ViewModel 不應該知道與之交互的View 是什麼。

上面說過Model 層裡包括了一些業務邏輯和業務數據模型，而ViewModel 層即是視圖模型(Model of View)，其內是視圖的表示數據和邏輯。比如說Model 層的業務數據是1, 2, 3, 4, 而翻譯到View 層，則可能是表示A, B, C, D 了。ViewModel 除了做這個事情外，還會封裝視圖的行為動作，如點擊某個控件後的行為等。另外注意這裡的ViewModel 和Jetpack 包裡提供的ViewModel 組件不是一個東西，這裡的ViewModel 是一個概念，而Jetpack 包則提供了一個比較方便的實現方式。

很多講MVVM 的文章示例都會用DataBinding ，然而沒有DataBinding照樣可以使用MVVM 架構，比如說借用LiveData, RxJava, Flow 等，這些工具都是基於響應式開發的原理，來替代基於接口的通信方式。實際開發中基本沒看到過使用DataBinding的，另外如果真要使用DataBinding的話，盡量避免在xml 裡寫代碼邏輯，而應替換成變量來表示某個屬性，在Kotlin 代碼裡賦值。

這裡的響應式開發強調一種基於觀察者模式的開發方式: View 訂閱ViewModel 暴露的響應式接口，接收到通知後進行相應邏輯，而ViewModel 不再持有任何形式的View 的引用，減少耦合，提高了可複用性。

另外如果使用LiveData 的話, ViewModel 對View 層僅暴露LiveData 接口，在View 層不允許直接更新LiveData, 因為一旦View 層擁有直接更新LiveData 的能力，就無法約束View 層進行業務處理的行為：

```
class LoginViewModel : ViewModel() {
  private val _loginResult: MutableLiveData<Boolean> = MutableLiveData()
  val loginResult: LiveData<Boolean> = _loginResult
}
```

關於flow 之前有在掘金上寫過一篇學習筆記，感興趣可以看看: Kotlin協程之flow工作原理。
https://juejin.cn/post/6966047022814232613

> 以登錄結果為例, MVVM 基於LiveData 的交互流程: 首先ViewModel 中有一個LiveData 屬性表示登錄結果，對外暴露出LiveData 而不是MutableLiveData, View 層會訂閱這個數據; View 層點擊登錄後，調用VM 的登錄接口, VM 然後請求Model 層的登錄能力; Model 完事後通知到VM, VM 更新MutableLiveData 登錄狀態, 而View 則收到了LiveData 的變化通知，進而更新UI。

實例

1、Model 層模擬登錄，返回登錄結果。

```
class LoginModel {
  // 模拟登录
  suspend fun login(): Boolean = withContext(Dispatchers.IO) {
    delay(1000)
    Random.nextBoolean()
```

```
    }
}
```

2、ViewModel 層暴露login方法，並提供LiveData 數據表示登錄狀態，讓View 層訂閱。

```
class LoginViewModel : ViewModel(), CoroutineScope by MainScope() {
    private val loginModel = LoginModel()

    private val _loginResult: MutableLiveData<Int> = MutableLiveData()
    val loginResult: LiveData<Int> = _loginResult

    fun login() {
        launch {
            _loginResult.value = 0
            val result = loginModel.login()
            _loginResult.value = if (result) 1 else -1
        }
    }

    // 模拟状态
    fun loginProgressText(result: Int): String = when (result) {
        0 -> "登录中"
        1 -> "登录成功"
        else -> "登录失败"
    }
}
```

3、View 層處理點擊事件，並訂閱登錄狀態。

```
class MVVMLoginActivity : AppCompatActivity() {
    private val viewModel: LoginViewModel by lazy {
        ViewModelProvider(this).get(LoginViewModel::class.java)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(Button(this).apply {
            text = "登录"
            setOnClickListener {
                viewModel.login()
            }
        })
        // 监听登录状态
        viewModel.loginResult.observe(this, {
            Toast.makeText(
                this,
                "Login result: ${viewModel.loginProgressText(it)}",
                Toast.LENGTH_SHORT
            ).show()
        })
    }
}
```

5 Repository

> Repository 模式的概念來自於領域驅動開發(Domain Driven Design)。主要思想是通過抽像一個 Repository 層，對業務(領域)層屏蔽不同數據源的訪問細節，業務層(可能是ViewModel)無需關注具體的數據訪問細節。

Repository 內部實現了對不同數據源(DataSource)的訪問，典型的DataSource 包括遠程數據, Cache 緩存, Database 數據庫等，可以用不同的Fetcher 來實現, Repository 持有多個Fetcher 引用。

因此上面實例中的LoginModel 可以換成LoginRepository類, LoginRepository不暴露具體的數據訪問方式，只暴露出這一能力的接口。

6 寫在最後

Android架構學習之路系列

https://juejin.cn/column/7051923621182341127

> 架構不是一蹴而就的，希望我們有一天的時候，能夠從自己寫的代碼中找到架構的成就感，而不是乾幾票就跑路。這個系列應該會一直更新，記錄我在架構之路上學習的腳印兒，一件一件扒開架構神秘的面紗。

文中內容如有錯誤歡迎指出，共同進步！覺得不錯的同學留個贊再走哈~你的三連是我寫作的動力！

最後推荐一下我做的網站，玩Android: *wanandroid.com* ，包含詳盡的知識體系、好用的工具，還有本公眾號文章合集，歡迎體驗和收藏！

推薦閱讀：

新技術又又又又叒叒叒來了?

分享一個Kotlin 高端玩法：DSL！

新技術ViewBinding 最佳實踐& 原理擊穿

鴻洋

你好，歡迎關注鴻洋的公眾號，每天為您推送高質量文章，讓你每天都能漲知識。點擊...

265篇原創內容

公眾號

點擊 關注我的公眾號

如果你想要跟大家分享你的文章，歡迎投稿~

┏(^ 0 ^)┛明天見!

閱讀原文

喜歡此內容的人還喜歡

一篇文章為你圖解Kubernetes 網絡通信原理，運維請收藏~

高效運維

交換機、路由器接口、線纜介紹，值得收藏學習

弱電智能化工程2018

一文詳解Kubernetes 中的服務發現，運維請收藏

高效運維