Android Fragments

deepwaterooo

June 27, 2022

Contents

1 关于 Fragment 的 onResume() 以及可见状态的判断

- Fragment 中有 onResume() 和 onPause() 方法，一般来说这两个方法是跟 Activity 绑定的，即触发 Activity 的这两个方法时就会触发 Fragment 的这两个方法，另外 Fragment 还有可见与不可见状态，但是 Fragment 中的 onResume()/onPause() 和可见/不可见完全是两码事。这里我将这两种情况做了一下分析，以做记录。

1.1 onResume() 和 onPause()

- 这两个方法会跟随 Activity 而变化，但是也不一定只有经历 Activity 的生命周期才会触发，这又分两种情况：
- 1. 在 Activity 中单独使用 Fragment
  - 单独使用 Fragment 的情况下，除了会跟随 Activity 的 onResume 和 onPause 的情况外，在当前 Activity 中添加一个 Fragment 时，被添加的 Fragment 都会调用 onResume，这包括通过 add 和 replace 的方式添加；在 Fragment 销毁的情况下都会触发 onPause，如果是 add 到回退栈的情况，back 键就会触发 onPause，如果是 replace 的情况，则被替换掉的 Fragment 会触发 onPause。
  - 如果 Activity 中添加了多个 Fragment，那么当 Activity 触发 onResume 和 onPause 时，Activity 中的每一个 Fragment 都会触发这两个方法，即便这个 Fragment 不在最顶层。
- 2. 在 ViewPager 中使用 Fragment
  - 在 ViewPager 中使用 Fragment 时这两个方法就更加奇妙了，因为你看不见的 Fragment 也会触发 onResume 和 onPause，这时跟 ViewPager 预加载相关，默认情况下，ViewPager 左右会预加载 1 页，所以如果 ViewPager 中有四个 Fragment A、B、C、D，当你展示 A Fragment 的时候，A 和 B 两个 Fragment 都会触发 onResume，当你切换到 B，则 C Fragment 触发 onResume，当你再切换到 C，则 A Fragment 会触发 onPause(这时 A 已经销毁了)，然后 D Fragment 会触发 onResume，再回来的时候也是这样，如此反复。
  - 即被 ViewPager 预加载的 Fragment 全部会调用 onResume，被 ViewPager 销毁的 Fragment 全部会调用 onPause。
  - 当 ViewPager 所在的 Activity 触发 onResume 和 onPause 时，ViewPager 中所有已创建的 Fragment 都会触发这两个方法（即包括 ViewPager 预加载的 Fragment），比如假如你当前处于 B Fragment，当你按下 Home 键时，A B C 三个 Fragment 都会触发 onPause，当你再返回来的时候，A B C 三个 Fragment 都会触发 onResume。

1.2 Visible 和 Hidden

- Fragment 中有两个方法 isHidden() 和 isVisible()，注释说是可以用来判断其可见性，但是经过测试如果你是在 Fragment 的 onResume/onPause 方法中调用这两个方法判断是不准确的，onResume 的方法中 isVisible() 有可能为 false，而 onPause 的方法中 isVisible() 也有可能为 true。可见性需要通过下面两个方法做判断：

```
@Override
public void onHiddenChanged(boolean hidden) {
    super.onHiddenChanged(hidden);
}
@Override
public void setUserVisibleHint(boolean isVisibleToUser) {
    super.setUserVisibleHint(isVisibleToUser);
}
```

- 这也分两种情况，onHiddenChanged 是在单独使用 Fragment 的时候进行 show/hide 操作时使用，setUserVisibleHint 是在 ViewPager 中使用时判断。
- 1. 单独使用 Fragment 的情况
  - 单独使用 Fragment 的情况比较简单，就是对 Fragment 进行 show 和 hide 的时候会回调 onHiddenChanged 方法。除此之外的操作，如 add/replace/remove 都不会触发这个方法。
- 2. 在 ViewPager 中使用 Fragment 的情况
  - 在 ViewPager 中使用 Fragment 时，同样跟 ViewPager 的预加载有关，假如我们有 A B C D 四个 Fragment，初次打开默认是 A Fragment，这时会调用 A 和 B 的 setUserVisibleHint 方法设置参数值为 false，紧接着它会再调用一次 A 的 setUserVisibleHint 方法设置参数值为 true。当切换到 B 时，则会调用 C 和 A 的 setUserVisibleHint 方法参设置数值为 false，当切换到 C 时，则会调用 D 和 B 的 setUserVisibleHint 方法设置参数值为 false，当再次返回 B 时，则会调用 A 和 C 的 setUserVisibleHint 方法设置参数值为 false。
  - 即被 ViewPager 预加载的 Fragment 全部会调用 setUserVisibleHint 方法设置参数值为 false，只有当前展示的那一页则会调用 setUserVisibleHint 方法设置参数值为 true。另外值得注意的一点是 setUserVisibleHint 这个方法的调用时机很早，会在 Fragment 的 onAttach()、onCreate() 等方法之前调用，所以，如果你在这里用到了全局变量一定要判断是否初始化了。

1.3 总结

- 以上是经过测试分析的结果，看了一些博客，有些人喜欢将 onResume 和可见状态联合起来做 Fragment 的可见性的判断，这个是非常麻烦并且十分不靠谱的，因为在 Fragment 的 onResume/onPause 的时候它的可见状态是完全不确定的。因此不能把 onResume/onPause 和 visible/hidden 状态混为一谈，要分开对待，因为这两种有可能会交叉发生。

2 其它一些零零碎碎的补充

- 当碎片第一次被加载到屏幕上时，会依次执行 onAttach ()，onCreate ()，onCreateView ()，onActivityCreated ()，onStart () 和 onResume () 方法。
- 当点击按钮切换到下一张图片时，第二张将第一张图片覆盖掉，此时用户看不到第一张图片，第一个碎片进入了停止状态。因此 onPause ()，onStop () 和 onDestroyView () 方法会得到执行。当然如果在替换的时候没有调用 addTobackStack () 方法，此时第一个碎片就会进入销毁状态，onDestroy () 和 onDetach () 方法就会得到执行。
- 接着按下 Back 键，第一个图片就会重新回到屏幕，由于第一个碎片重新回到了运行状态，因此 onCreateView ()，onActivityCreated ()，onSart () 和 onResume () 方法会得到执行。注意此时 onCreate () 方法不会执行，因为我们借助了 addToBackStack () 方法使得第一个碎片并没有被销毁。
- 再次按下 Back 键后，依次会执行 onPause ()，onStop ()，onDestroyView () onDestroy () 和 onDetach () 方法，最终将碎片销毁掉。
- 另外值得一提的是，在碎片中你也是可以通过 onSaveInstanceState () 方法来保存数据的，因为进入停止状态的碎片有可能在系统内存不足的时候被回收。保存下来的数据在 onCreate ()，onCreateView () 和 onActivityCreated () 这 3 个方法中都可以重新得到，他们都含有一个 Bundle 类型的 savedInstanceState 参数。

3 AndroidX 下你得知道的 Activity 和 Fragment 的变化：可能还稍微有点作显得假大空而缺少实际体会

3.1 在构造器中传入布局 ID

- 从 AndroidX AppCompat 1.1.0 和 Fragment 1.1.0（译者注：AppCompat 包含 Fragment，且 Fragment 包含 Activity，详情见【整理】Jetpack 主要组件的依赖及传递关系）开始，您可以使用将 layoutId 作为参数的构造函数：
- class MyActivity : AppCompatActivity(R.layout.my_activity)class MyFragmentActivity: FragmentActivity(R.layout.my_fragment_activity)class MyFragment : Fragment(R.layout.my_fragmen
- 这种方法可以减少 Activity/Fragment 中方法重写的数量，并使类更具可读性。无需在 Activity 中重写 onCreate() 即可调用 setContentView() 方法。另外，无需手动在 Fragment 中重写 onCreateView 即可手动调用 Inflater 来扩展视图。

3.2 扩展 Activity/Fragment 的灵活性

- 借助 AndroidX 新的 API，可以减少在 Activity/Fragment 处理某些功能的情况。通常，您可以获取提供某些功能的对象并向其注册您的处理逻辑，而不是重写 Activity / Fragment 中的方法。这样，您现在可以在屏幕上组成几个独立的类，获得更高的灵活性，复用代码，并且通常在不引入自己的抽象的情况下，对代码结构具有更多控制。让我们看看这在两个示例中如何工作。

3.2.1 1. OnBackPressedDispatcher

- 有时，您需要阻止用户返回上一级。在这种情况下，您需要在 Activity 中重写 onBackPressed() 方法。但是，当您使用 Fragment 时，没有直接的方法来拦截返回。在 Fragment 类中没有可用的 onBackPressed() 方法，这是为了防止同时存在多个 Fragment 时发生意外行为。
- 但是，从 AndroidX Activity 1.0.0 开始，您可以使用 OnBackPressedDispatcher 在您可以访问该 Activity 的代码的任何位置（例如，在 Fragment 中）注册 OnBackPressedCallback。

```
class MyFragment : Fragment() {
    override fun onAttach(context: Context) {
        super.onAttach(context)
        val callback = object : OnBackPressedCallback(true) {
            override fun handleOnBackPressed() {
                // Do something
            }
        }
        requireActivity().onBackPressedDispatcher.addCallback(this, callback)
    }
}
```

- 您可能会在这里注意到另外两个有用的功能：
- OnBackPressedCallback 的构造函数中的布尔类型的参数有助于根据当前状态动态打开/关闭按下的行为
- addCallback() 方法的可选第一个参数是 LifecycleOwner，以确保仅在您的生命周期感知对象（例如，Fragment）至少处于 STARTED 状态时才使用回调。
- 通过使用 OnBackPressedDispatcher，您不仅可以获得在 Activity 之外处理返回键的便捷方式。根据您的需要，您可以在任意位置定义 OnBackPressedCallback，使其可复用，或根据应用程序的架构进行任何操作。您不再需要重写 Activity 中的 onBackPressed 方法，也不必提供自己的抽象的来实现需求的代码。

3.2.2 2. SavedStateRegistry

- 如果您希望 Activity 在终止并重启后恢复之前的状态，则可能要使用 saved state 功能。过去，您需要在 Activity 中重写两个方法：onSaveInstanceState 和 onRestoreInstanceState。您还可以在 onCreate 方法中访问恢复的状态。同样，在 Fragment 中，您可以使用 onSaveInstanceState 方法（并且可以在 onCreate，onCreateView 和 onActivityCreated 方法中恢复状态）。

- 从 AndroidX SavedState1.0.0（它是 AndroidX Activity 和 AndroidX Fragment 内部的依赖。译者注：您不需要单独声明它）开始，您可以访问 SavedStateRegistry，它使用了与前面描述的 OnBackPressedDispatcher 类似的机制：您可以从 Activity / Fragment 中获取 SavedStateRegistry，然后注册您的 SavedStateProvider：

```
class MyActivity : AppCompatActivity() {
    companion object {
        private const val MY_SAVED_STATE_KEY = "my_saved_state"
        private const val SOME_VALUE_KEY = "some_value"
    }
    private lateinit var someValue: String
    private val savedStateProvider = SavedStateRegistry.SavedStateProvider {
        Bundle().apply {
            putString(SOME_VALUE_KEY, someValue)
        }
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        savedStateRegistry
            .registerSavedStateProvider(MY_SAVED_STATE_KEY, savedStateProvider)
    }
    fun someMethod() {
        someValue = savedStateRegistry
            .consumeRestoredStateForKey(MY_SAVED_STATE_KEY)
        ?.getString(SOME_VALUE_KEY)
        ?: ""
    }
}
```

- 如您所见，SavedStateRegistry 强制您将密钥用于数据。这样可以防止您的数据被 attach 到同一个 Activity/Fragment 的另一个 SavedStateProvider 破坏。就像在 OnBackPressedDispatcher 中一样，您可以例如将 SavedStateProvider 提取到另一个类，通过使用所需的任何逻辑使其与数据一起使用，从而在应用程序中实现清晰的保存状态行为。

- 此外，如果您在应用程序中使用 ViewModel，请考虑使用 AndroidX ViewModel-SavedState 使你的 ViewModel 可以保存其状态。为了方便起见，从 AndroidX Activity 1.1.0 和 AndroidX Fragment 1.2.0 开始，启用 SavedState 的 SavedStateViewModelFactory 是在获取 ViewModel 的所有方式中使用的默认工厂：委托 ViewModelProvider 构造函数和 ViewModelProviders.of() 方法。

3.2.3 FragmentFactory

- Fragment 最常提及的问题之一是不能使用带有参数的构造函数。例如，如果您使用 Dagger2 进行依赖项注入，则无法使用 Inject 注解 Fragment 构造函数并指定参数。现在，您可以通过指定 FragmentFactory 类来减少 Fragment 创建过程中的类似问题。通过在 FragmentManager 中注册 FragmentFactory，可以重写实例化 Fragment 的默认方法：

```
class MyFragmentFactory : FragmentFactory() {
  override fun instantiate(classLoader: ClassLoader, className: String): Fragment {
    // Call loadFragmentClass() to obtain the Class object
    val fragmentClass = loadFragmentClass(classLoader, className)


    // Now you can use className/fragmentClass to determine your prefered way
    // of instantiating the Fragment object and just do it here.
```

```
    // Or just call regular FragmentFactory to instantiate the Fragment using
    // no arguments constructor
    return super.instantiate(classLoader, className)
  }
}
```

- 如您所见，该 API 非常通用，因此您可以执行想要创建 Fragment 实例的所有操作。回到 Dagger2 示例，例如，您可以注入 FragmentFactory Provider <Fragment> 并使用它来获取 Fragment 对象。

3.2.4 测试 Fragment

- 从 AndroidX Fragment 1.1.0 开始，可以使用 Fragment 测试组件提供 FragmentScenario 类，该类可以帮助在测试中实例化 Fragment 并进行单独测试：

```
// To launch a Fragment with a user interface:
val scenario = launchFragmentInContainer<FirstFragment>()

// To launch a headless Fragment:
val scenario = launchFragment<FirstFragment>()

// To move the fragment to specific lifecycle state:
scenario.moveToState(CREATED)

// Now you can e.g. perform actions using Espresso:
onView(withId(R.id.refresh)).perform(click())

// To obtain a Fragment instance:
scenario.onFragment { fragment ->
}
```

3.2.5 More Kotlin!

- 很高兴看到 -ktx AndroidX 软件包中提供了许多有用的 Kotlin 扩展方法，并且定期添加了新的方法。例如，在 AndroidX Fragment-KTX 1.2.0 中，使用片段化类型的扩展名可用于 FragmentTransaction 上的 replace() 方法。将其与 commit() 扩展方法结合使用，我们可以获得以下代码：

```
// Before
supportFragmentManager
  .beginTransaction()
  .add(R.id.container, MyFragment::class.java, null)
  .commit()


// After
supportFragmentManager.commit {
  replace<MyFragment>(R.id.container)
}
```

3.2.6 FragmentContainerView

- 一件小而重要的事情。如果您将 FrameLayout 用作 Fragment 的容器，则应改用 FragmentContainerView 。它修复了一些动画 z 轴索引顺序问题和窗口插入调度。从 AndroidX Fragment 1.2.0 开始可以使用 FragmentContainerView。

4 Fragment 的生命周期函数详解

4.1 几个常用生命周期函数的简介

- onAttach()：Fragment 和 Activity 相关联时调用。可以通过该方法获取 Activity 引用，还可以通过 getArguments() 获取参数。
- onCreate()：Fragment 被创建时调用。
- onCreateView()：创建 Fragment 的布局。
- onActivityCreated()：当 Activity 完成 onCreate() 时调用。
- onStart()：当 Fragment 可见时调用。
- onResume()：当 Fragment 可见且可交互时调用。
- onPause()：当 Fragment 不可交互但可见时调用。
- onStop()：当 Fragment 不可见时调用。
- onDestroyView()：当 Fragment 的 UI 从视图结构中移除时调用。
- onDestroy()：销毁 Fragment 时调用。
- onDetach()：当 Fragment 和 Activity 解除关联时调用。

4.2 Fragment 生命周期解析

4.2.1 当一个 fragment 被创建的时候：

- onAttach()
- onCreate()
- onCreateView() //
- onActivityCreated()

4.2.2 当这个 fragment 对用户可见的时候，它会经历以下状态。

- onStart()
- onResume()

4.2.3 当这个 fragment 进入“后台模式”的时候，它会经历以下状态。

onPause() onStop()

4.2.4 当这个 fragment 被销毁了（或者持有它的 activity 被销毁了）：

- onPause()
- onStop()
- onDestroyView() //
- onDestroy()
- onDetach()

4.2.5 就像 Activity 一样，在以下的状态中，可以使用 Bundle 对象保存一个 fragment 的对象。

- onCreate()
- onCreateView() //
- onActivityCreated()

4.3 其他场景的调用

4.3.1 屏幕灭掉

- onPause()
- onSaveInstanceState() // 这里可以保存
- onStop()

4.3.2 屏幕解锁

- onStart()
- onResume()

4.3.3 切换到其他 Fragment

- onPause()
- onStop()
- onDestroyView() //

4.3.4 切换回本身的 Fragment

- onCreateView() //
- onActivityCreated()
- onStart()
- onResume()

4.3.5 回到桌面: 这里写得好像是不对

- onPause()
- onSaveInstanceState() //
- onStop()

4.3.6 回到应用

- onStart()
- onResume()

4.3.7 退出应用

- onPause()
- onStop()
- onDestroyView() //
- onDestroy()
- onDetach()