# **实战中各种工具的使用和技巧**

看完文章能Get到哪些点？

1.DDMS快速定位关键代码位置

2.Android Studio动态调试smali

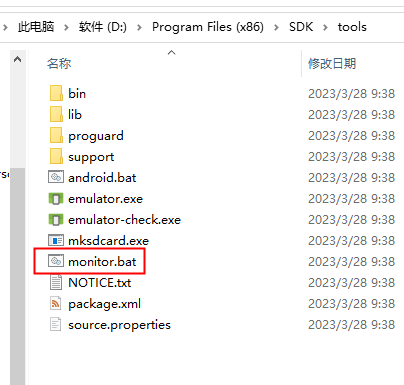
3.Jeb、GDA、jadx-gui等工具的配合使用

4.一些技巧

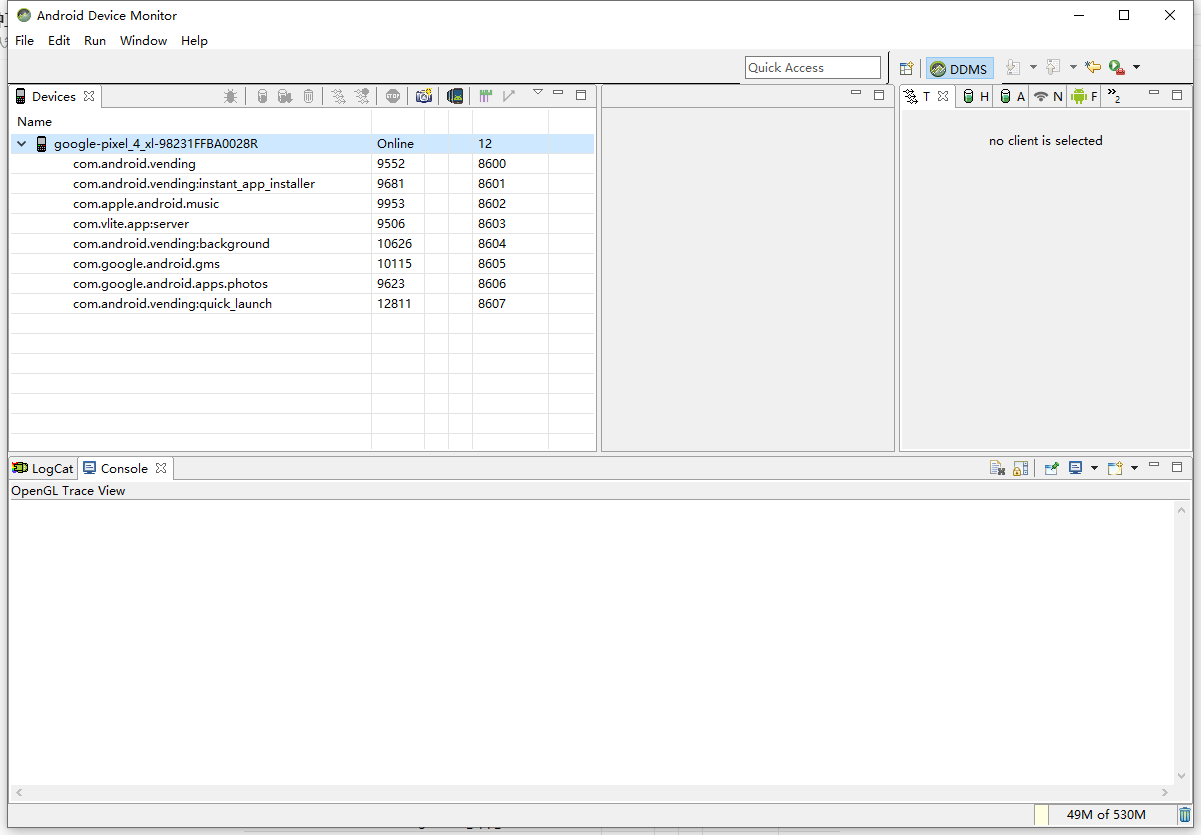
### **一、DDMS快速定位**

#### **1）DDMS配置**

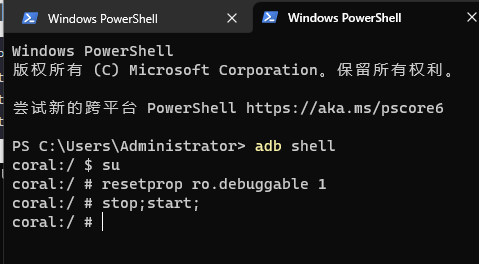
打开路径：SDK\tools\monitor.bat



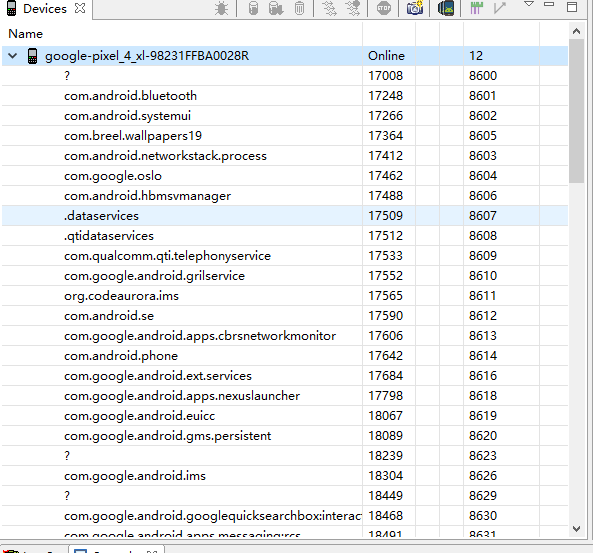
主界面显示



从上图中我们发现，在Devices窗口中，显示的进程非常少。这是什么原因呢？主要原因是我们没有开启全局系统调试，开启方式如下：



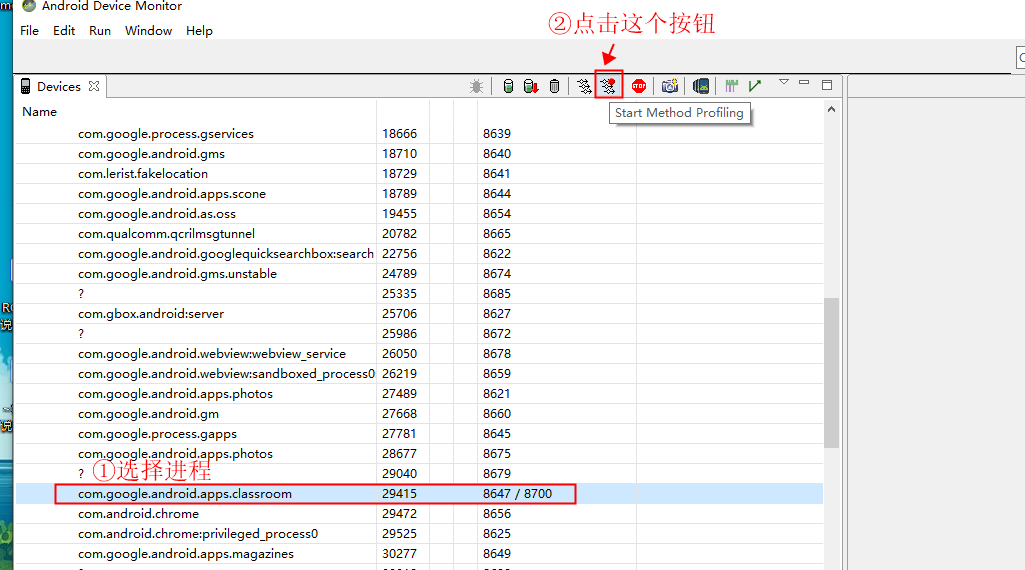
先使用"resetprop ro.debuggable 1" 开启系统调试，再使用"stop"和"start"重启系统。然后我们就发现多了许多的进程可以提供我们选择，如图：



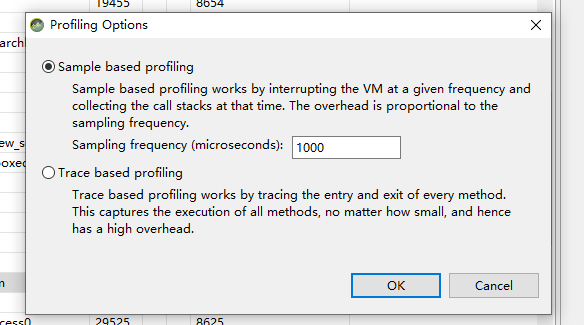
提示：如果打不开该应用的话，得配置系统变JAVA\_HOME

#### **2）Method Profiling**

这是在我们逆向分析中，一个快速定位问题所在方式之一。它能够快速的Trace操作中所调用的函数。具体操作如下图：

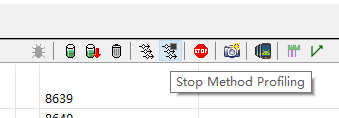


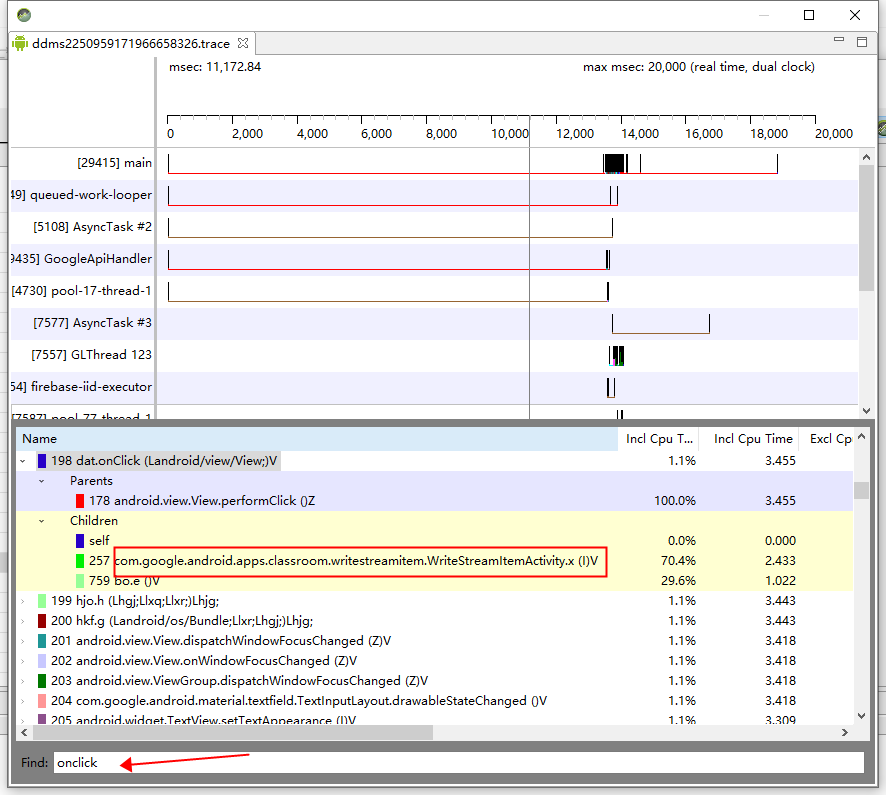
出现两个选择：



第一个选择要比第二个选择少很多，如果第一个选择里面没有trace到我们想要的方法，我们可以选择第二个，一般来说第一种就够了。

接下来当我们在手机或模拟器中做了一些操作之后（点击事件等操作），就可以点击stop按钮，停止trace操作，会显示出来一个界面

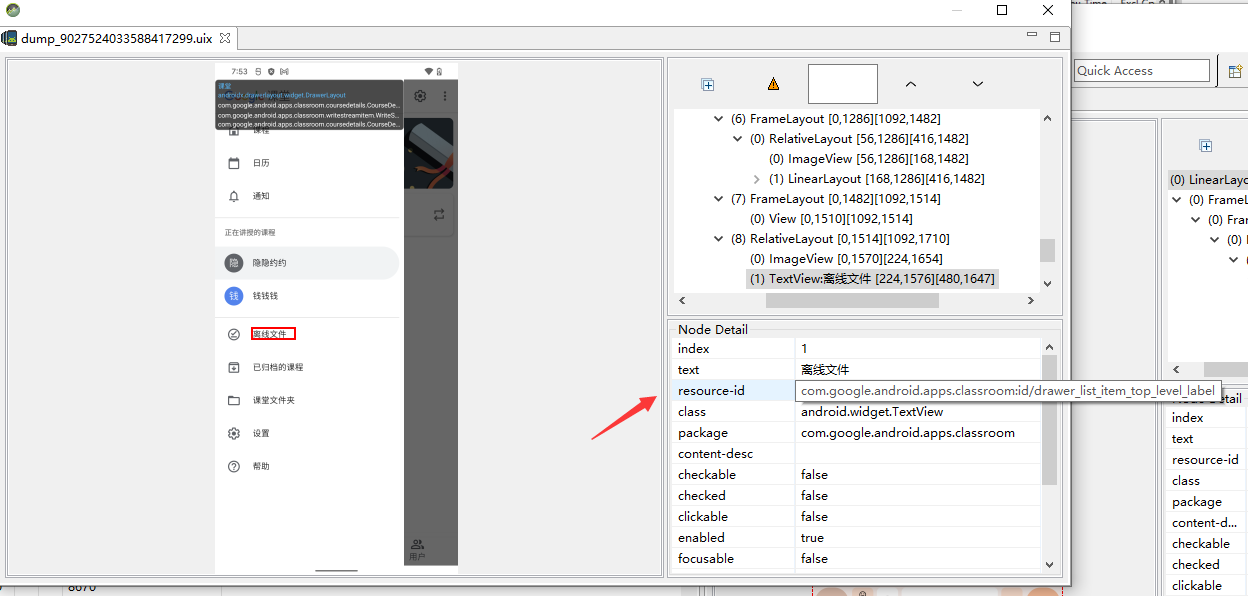




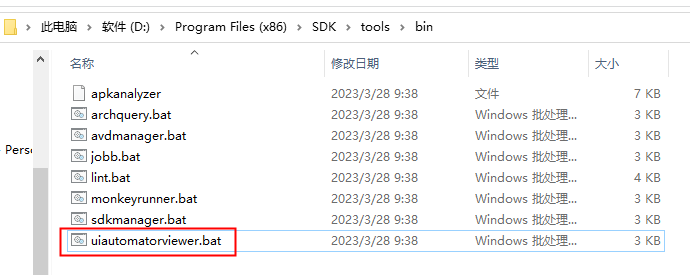
我们可以搜索刚才产生的点击事件onclick，就能直接定位到关键的地方，如上图。Parents指的是dat.onClick()是从android.view.View.performClick()函数调用的，而Children指的是当前dat.onClick()事件中调用了哪些函数。

#### **3）uiautomatorviewer**

这个功能可以帮助我们查看界面的资源ID，有助于定位关键位置。



这功能也可以单独在SDK\tools\bin目录下找到，如图：

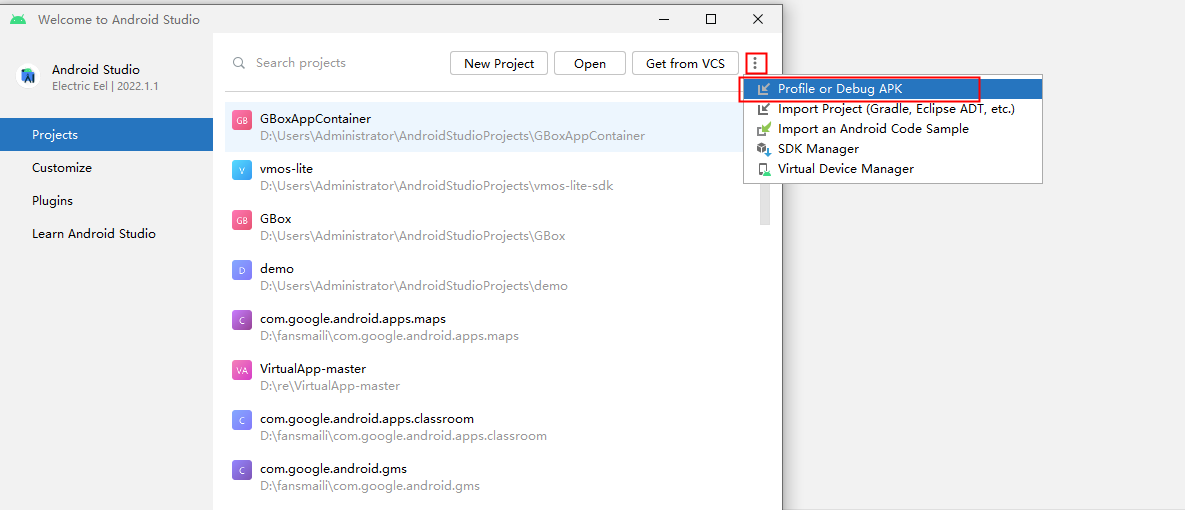


### **二、Android Studio动态调试smali**

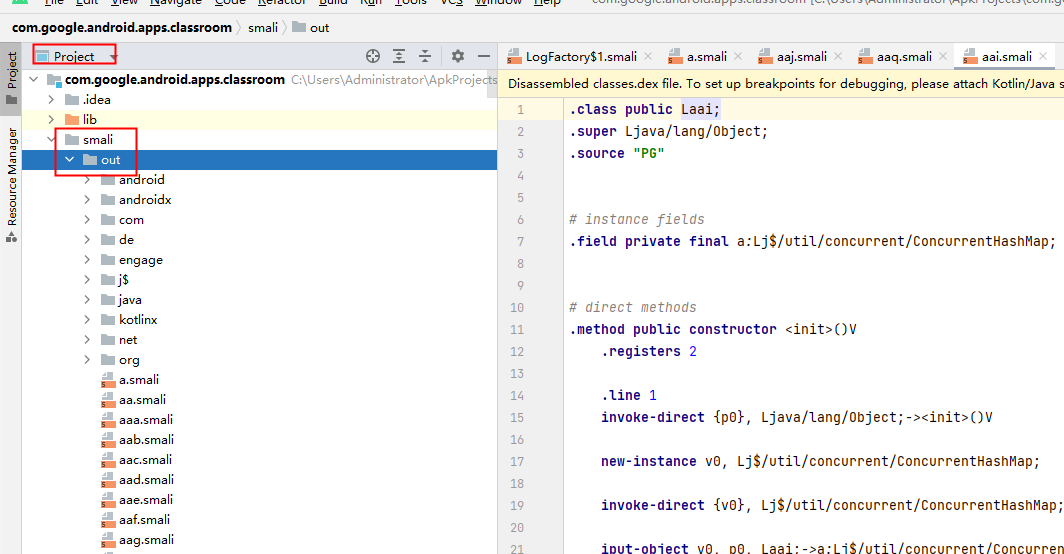
这里的动态调试主要指的是调试smali，为什么需要调试？反而没有选择使用frida去hook？这也是有原因的，当遇到大量的错综复杂的逻辑判断的时候，我们使用frida去一点一点hook，那不知道猴年马月才能分析到关键点，所以说不如直接调试代码(smali)。这里就不讲Jeb调试了，这玩意儿实在是太卡了，有兴趣可以自己研究。

#### **1）方式一**

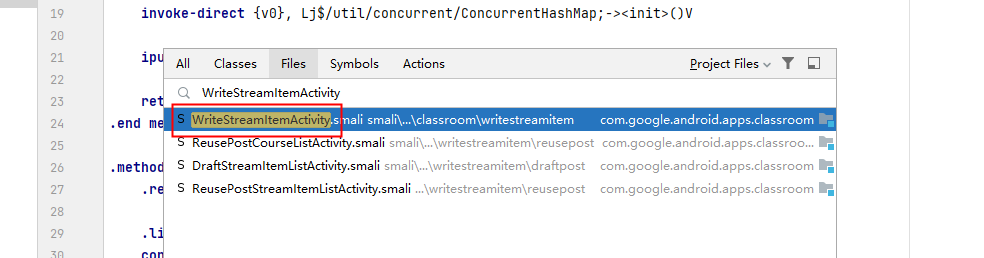
**步骤一**：点击“Profile or Debug APK”直接选择想要调试的apk



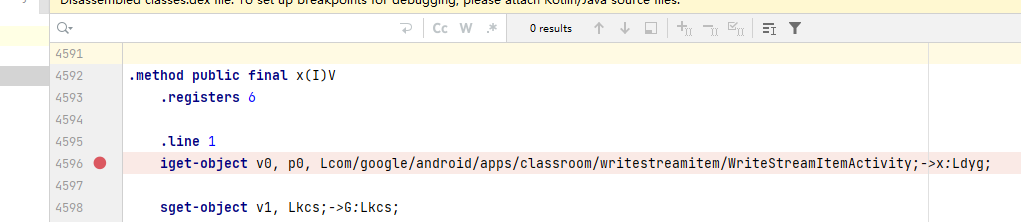
**步骤二**：切换到Project，smali->out。



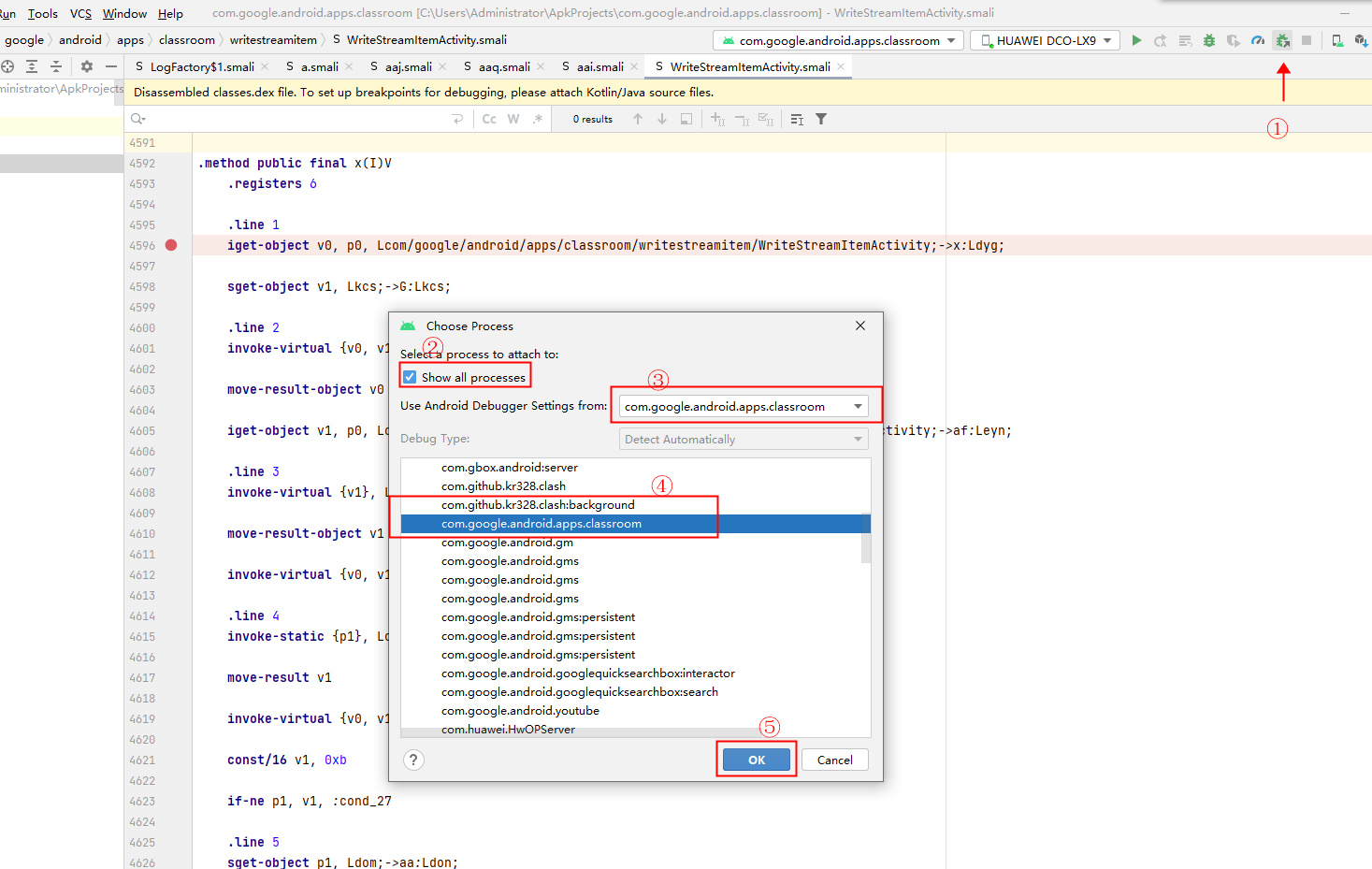
**步骤三**：ctrl+n 搜索具体的类（当然也可以搜索其他的东西）



**步骤四**：找到具体的函数位置下断点，下图红色地方代表下断成功。

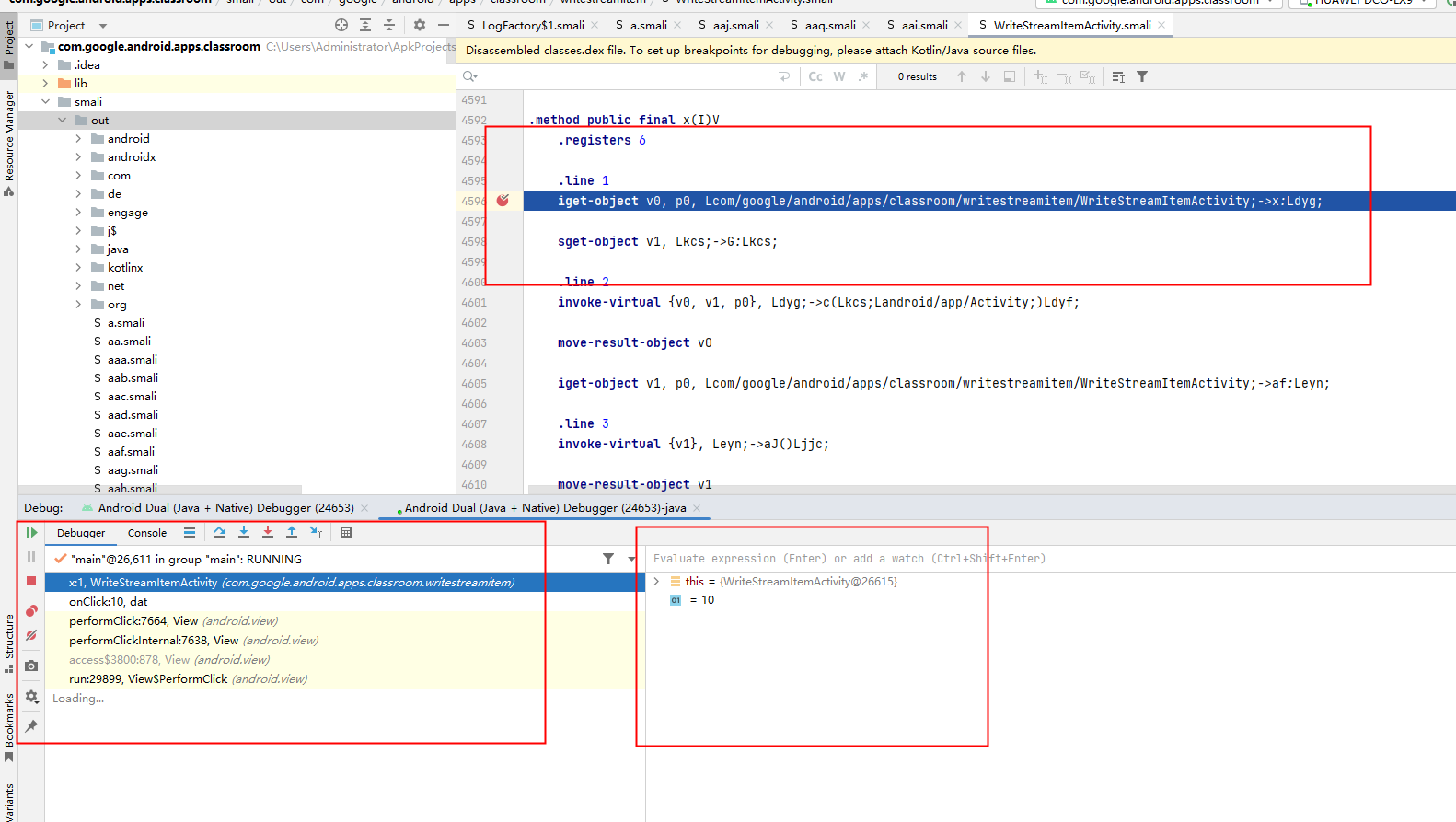


**步骤五**：先点击右上角debug按钮，依次选择对应的进程，点击ok就行



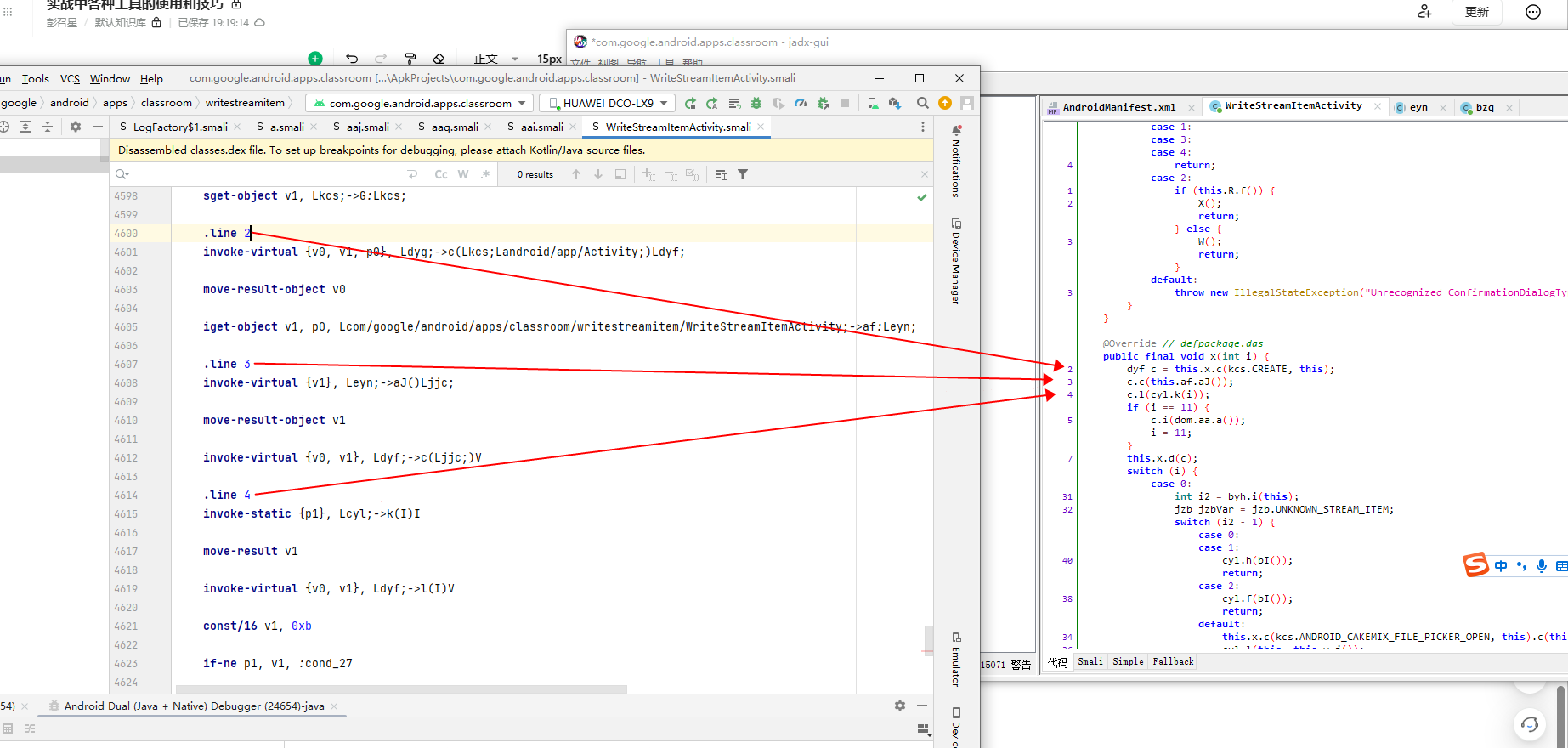
如果说这一步没有出现想要的进程信息，我们得使用开启全局系统调试（开启方式在DDMS配置这节中），一般在公司中华为手机不需要开启，pixel手机有时候会需要。这个得手机root，没有root只能修改apk的manifest.xml文件开启debug，重新打包编译了。

然后就可以愉快的去手机进行操作，就可以断下来了。断下来的话左下角窗口会显示**方法调用栈**，在代码窗口会高亮一行，监视窗口也会出现一些变量的显示。



##### **乱花渐欲迷人眼其一**

对不熟悉smali语法的同学在调试过程中是非常痛苦的，通常都是要对比java去一点调试分析，眼睛都看得花。我这里给大家介绍一款工具“jadx-gui，可以通过这款工具反汇编Java，并且快速的定位当前调试的smali的位置。

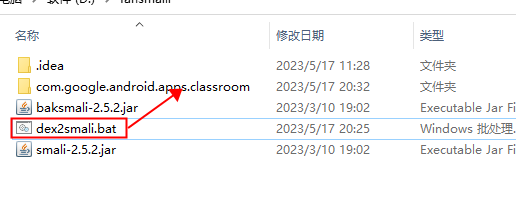


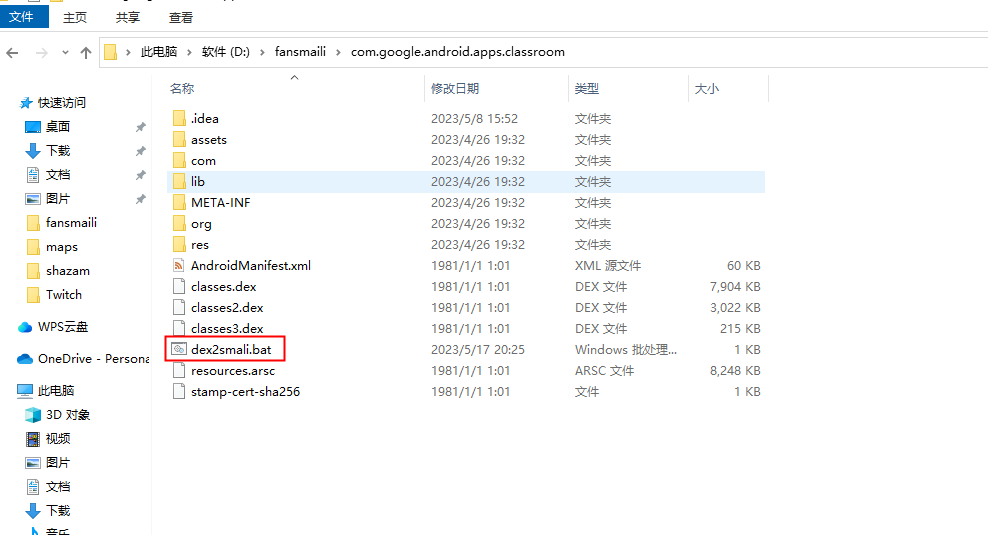
从图中可以发现，smali的.line可以对应到jadx的左边的标号，这样我们就可以知道当前所调试的位置在哪里，不会看花眼。

#### **2）方式二（对不熟悉smali的同学非常友好）**

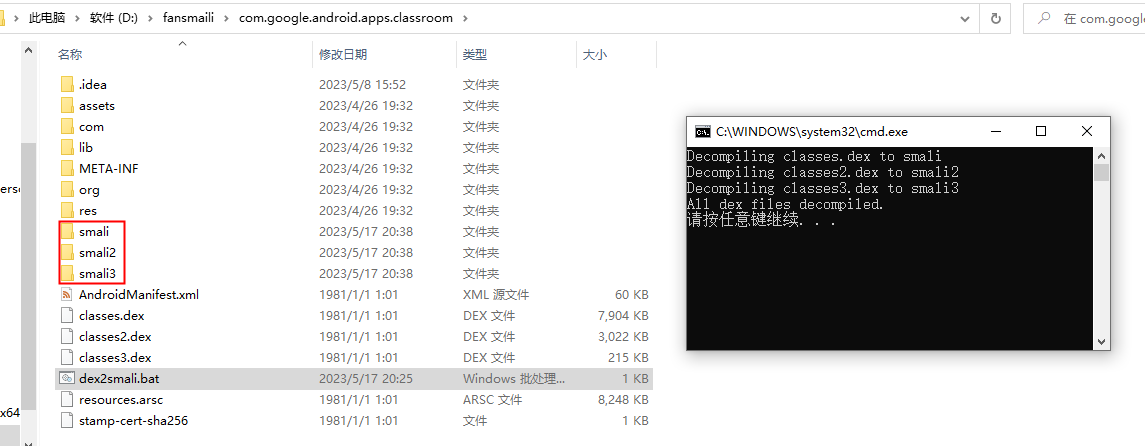
我们在上面的方式中，是通过Android Studio直接反编译dex文件成smali文件的。当然也可以自己手动使用命令去反编译dex，需要用到的工具baksmali.jar与smali.jar，工具脚本我已经打包好了。

步骤一：创建目录，并且把apk解压到这个目录中，把bat脚本放入里面

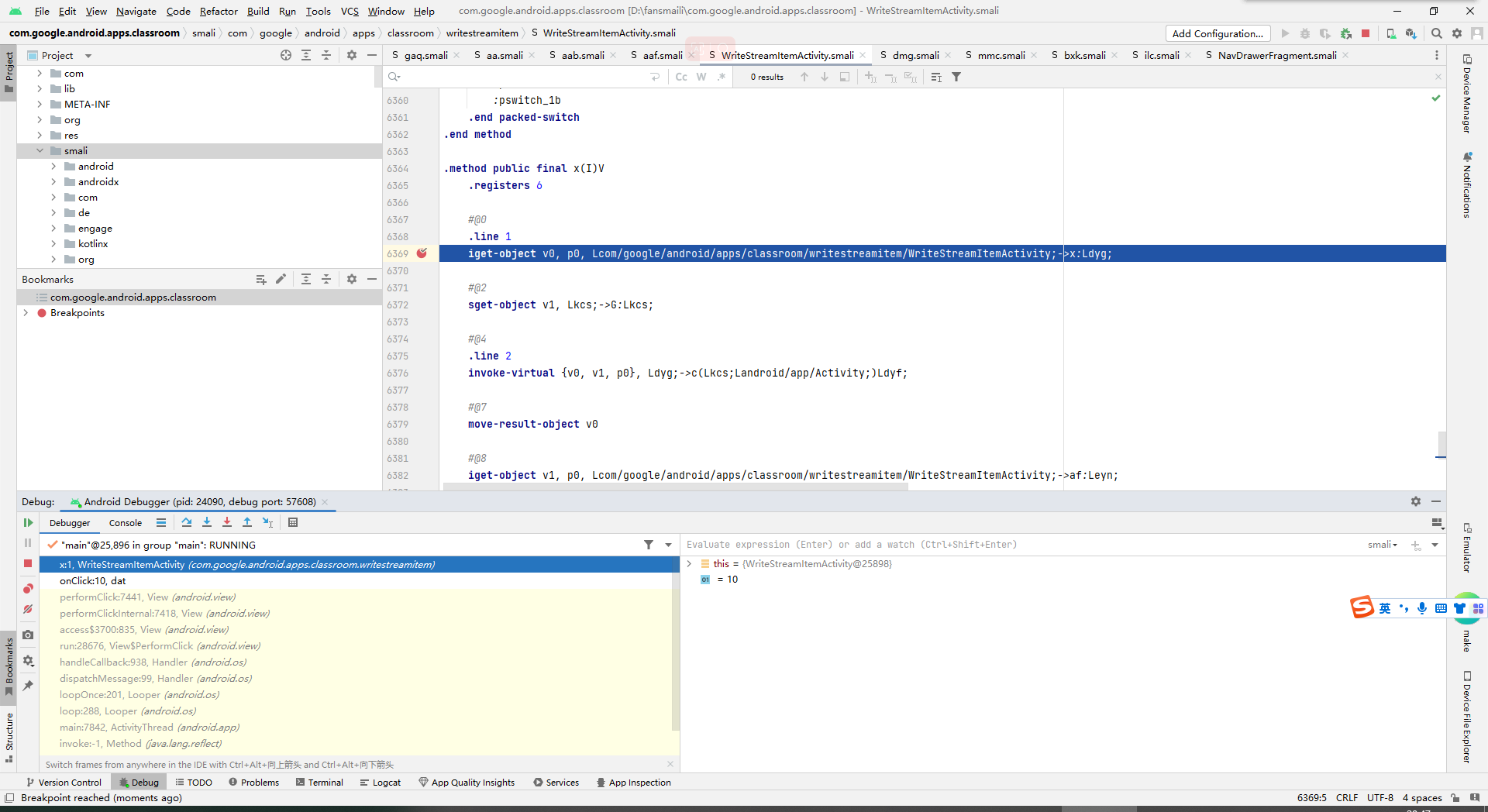




步骤二：然后再点击运行，对比原来的目录发现多了很多的smali开头的目录



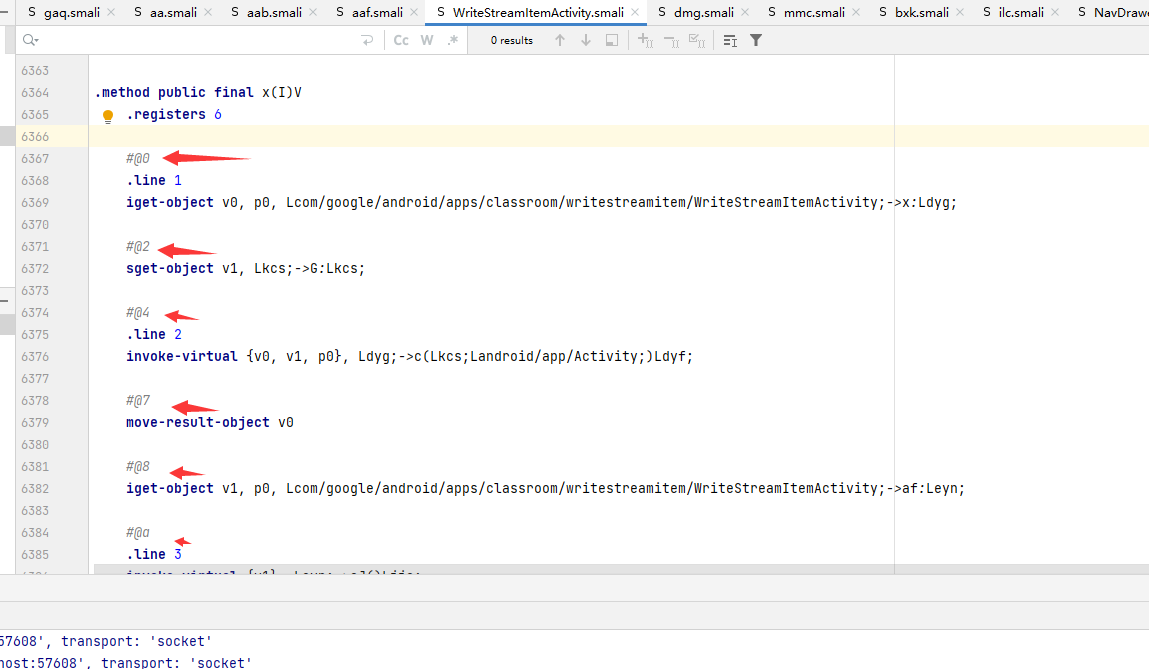
后面的操作就是使用Android Studio打开上面的工程目录，然后找到想要的位置下断点，后面的调试附加步骤基本与 方式一 差不多，可以参考上面的 方式一。



##### **乱花渐欲迷人眼其二**

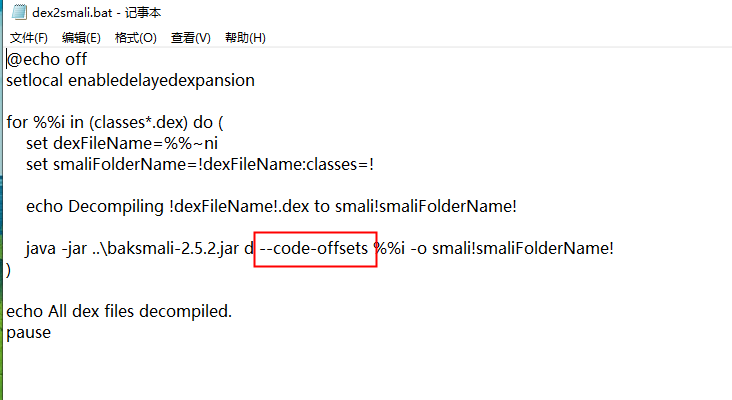
在这一节的标题我们就提到了 “对不熟悉smali的同学非常友好”，为什么这么说呢？我们可以认真对比**方式一**与**方式二**所返编译出来的smali文件。

可以以肉眼能看见，方式二生成的smali，多了很多的#号开头的注释（如下图），那么这个注释是怎么生成出来的？具体到底有什么作用？接下来这些问题就是我们要套论的话题。



**①这个注释是怎么生成的？**

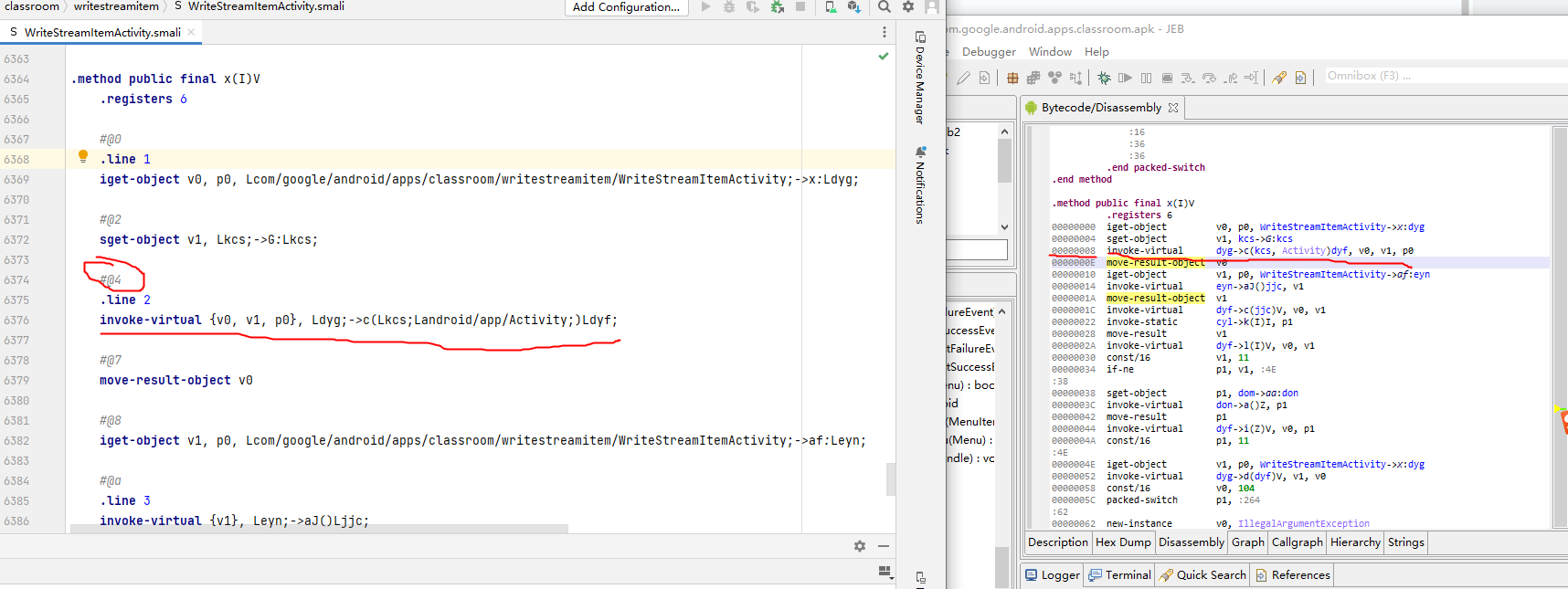
我们可以看看dex2smali.bat里的代码：



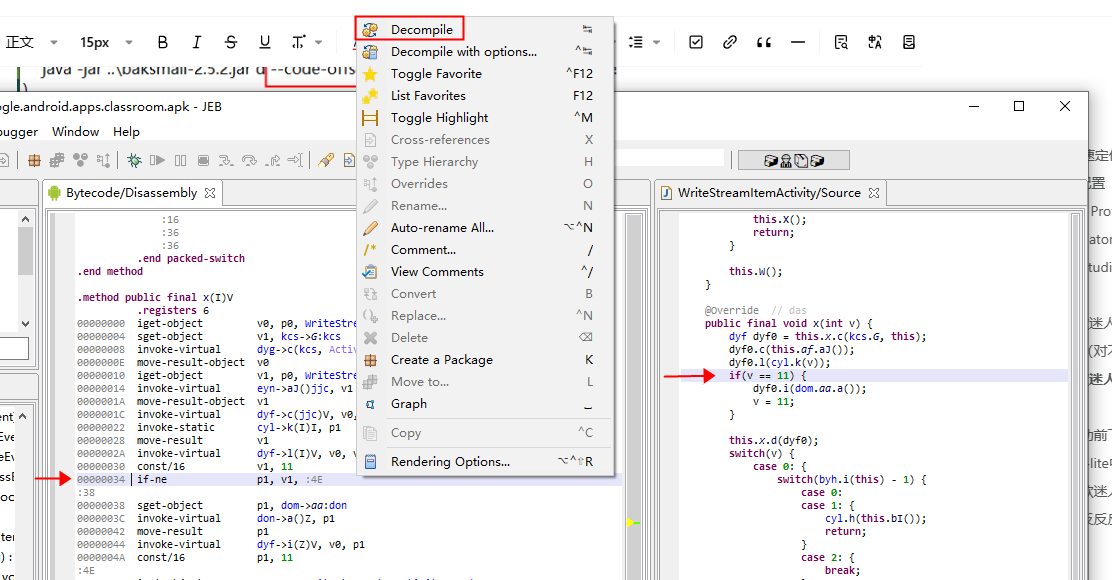
--code-offsets: 这个参数的含义就是在smali文件中添加代码偏移注释，也就是在代码中的"#@偏移"

**②具体的作用**

接下来就需要用到jeb了，我们把apk拖到jeb反编译，然后找到对应的代码：



可以发现上图中的左边的smali#@4 与 右边前面的地址00000008的关系，明显0x4\*2=0x8，大了整整一倍，我们可以利用这个偏移从AS计算转到jeb，接下来可以通过jeb (点击地址->右键->Decompile) 直接把smali转成java，然后对比就能看到当前smali在java中所在的位置。



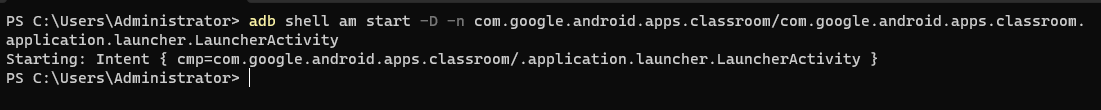
这个方式在我还不熟悉smali的时候，帮助我很多，又节约时间，直接转到关键的java视图。

### **三、一些技巧**

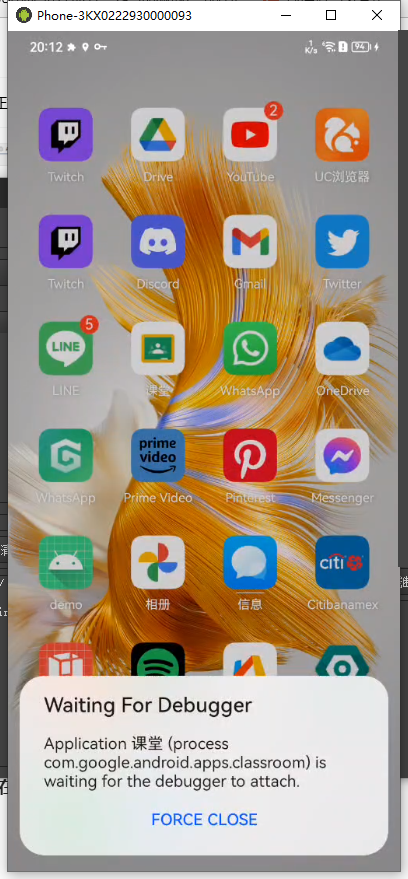
#### **1）应用启动前下断点**

在调试低包的时候，追踪一些流程或值的时候，我们发现在关键位置有时候会断不下来，我们就可以猜测是不是需要在应用启动的时候才会走这一块地方，那么就要想办法在启动之前断下来。

cmd中使用命令：

adb shell am start -D -n 包名/活动

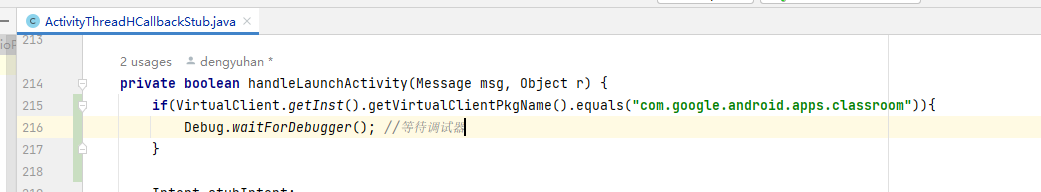
这个指令是以**调试状态**启动我们的应用，会一直等待我们Android Studio附加。



#### **2）在vmos-lite中，应用启动前下断点**

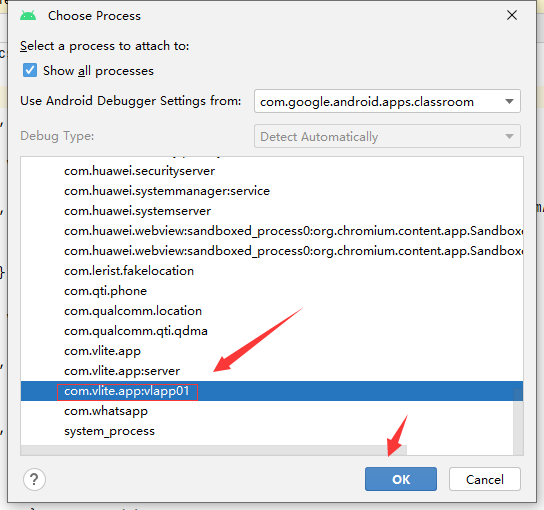
那么如何在vlite中如何调试呢？既然源码在手，那么我们可以在代码中选择一块合适的地方添加调试器代码。

在handleLaunchActivity方法中，添加。如下图代码：



if(VirtualClient.getInst().getVirtualClientPkgName().equals("com.google.android.apps.classroom")){  
 Debug.waitForDebugger(); //等待调试器  
}

点击附加还没有改名的进程

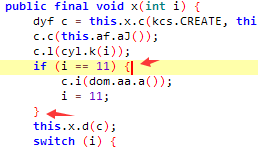


这个过程中应用也会一直等待调试器附加。

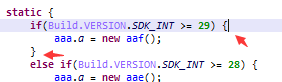
#### **3）乱花渐欲迷人眼其三**

在静态分析代码的时候，通常大家jadx或者jeb去分析，但是遇到很大的一块代码逻辑的时候，我们就不知道{} 或者() 这些括号的对应，干扰我们分析代码，难看奥，迷花了眼。jadx和jeb也没有相对于的高亮显示。

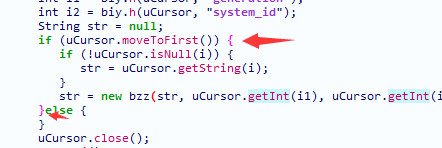
jadx表现×



jeb表现×



Gda √



可以看出来gda以不同颜色显示{}括号了

附件：

[请至钉钉文档查看附件《dex2smali.rar》](https://alidocs.dingtalk.com/i/nodes/Obva6QBXJwD5v1m1U3vM3Gba8n4qY5Pr?doc_type=wiki_doc&iframeQuery=anchorId%253DX02lhshbmr80nnztxl2brbp)

特别感谢

特别感谢小组兄弟们对我的帮助，还有其他同事们的帮助。