因一纸设计稿，我把竞品APP扒得裤衩不剩(中)

原创

coder-pig 于 2019-12-19 15:37:45 发布 3555 收藏 25

分类专栏： Android实战 文章标签： 逆向 Android 混淆 反混淆 脱壳

本文链接：https://blog.csdn.net/coder_pig/article/details/103615488

严正声明：

> - 1、相关破解技术仅限于技术研究使用，不得用于非法目的，否则后果自负。

- 2、笔者仅出于对技术的好奇，无恶意破坏APP，尊重原开发者的劳动成果，未用于商业用途。

0x1、无形之刃，最为致命 => 碎碎念

上一篇文章《因一纸设计稿，我把竞品APP扒得裤衩不剩(上)》是一篇比较简单的：

> - jsw => **技师文**，呸，
> - jsw => **记述文**，呸呸，
> - jsw => **技术文**，呸呸呸，这什么垃圾输入法！
>
> **技术文**，但是这评论区的风气，貌似有点不对？？？

> **冤枉啊**，小弟真没去过这种地方，也没体验过这种“**服务**”，只是道听途说，可能：
>
> > 我描述得「**绘声绘色**」，加之各位看官「**浮想联翩**」，才会觉得「**煞有介事**」。
>
> em...那个，可以扶下我起来么，那个...**跪久了**...**腿有点麻**...

顺带恭喜下：**FPX 3-0 G2，喜提S9总冠军，FPX牛逼！！！破音！！！**

亚索的快乐你不懂~

哈哈，**回到本文：**

- **发现** => 很多童鞋对APP逆向很感兴趣；
- **但是** => 网上关于APP逆向文章的比较零散；
- **不知** => 如何入手，毕竟逆向的水可深了；
- **笔者** => 也只是个小白玩家，兴趣使然玩玩而已；
- **分享** => 目前会的一些「**Android APP基础逆向姿势**」；
- **还望** => 各位真丶逆向大佬轻喷；
- **如有** => 有更好的工具或者方法安利，欢迎评论区指出，谢谢~

顺带分享几个笔者常逛的**逆向论坛：**

- **看雪论坛**：https://bbs.pediy.com/
- **吾爱破解**：https://www.52pojie.cn/
- **蚁安网**：https://bbs.mayidui.net/
- **逆向大佬姜维**：http://www.520monkey.com/

贴心提醒：

> 此文内容较多，可能会有些枯燥，**建议先点赞收藏**，茶余饭后再慢慢品尝~

0x2、提莫队长，正在待命 => 硬件准备

在开始折腾Android APP逆向前，你需要：

1、一台「**具有完整Root权限**」的**Android手机**，注意是「**完整Root**」权限！！！

> 比如「**魅族手机**」在设置->安全->Root权限，中可以开启Root权限，但是却是「**阉割的Root权限**」，安装SuperSu重启后就一直卡气球。

2、**怎么Root？根据自己的手机机型百度和逛各种搞机论坛吧**（不要问我！）一般的常见的流程：

> **解BL锁**(BootLoader) -> **刷第三方Recovery**(如TWRP) -> **卡刷Maglisk** 或 **SuperSU(Android 8.0以前)**

3、不要轻易尝试使用哪种「**一键Root**」的软件(大部分是**毒瘤**，如KingRoot)，当然不是就不能用，可以**过河拆桥**，比如我的魅蓝E2的Root流程：

> - 安装KingRoot(v5.0)授予Root权限，修复Root权限异常，此时就有完整Root权限了；
> - 接着用「**移除KingRoot**」删掉KingRoot，此时还有完整Root权限；
> - 安装**SuperSu**(v2.8.2)，常规方式更新二进制文件，重启，Root完成。

4、**推荐些能Root的手机？**

> **Google Pixel亲儿子**(真原生，香，就是性价比不高)，**小米**，**一加** 等。

5、没钱买Android机或者已经有不能Root的手机了，可以试试「**Android模拟器**」

> **AS自带的AVD模拟器Root可以参见**《搞机：AS自带模拟器AVD Root 和 Xposed安装》
> 也可以使用其他第三方的安卓模拟器，比如「**夜游安卓模拟器**、**BlueStacks蓝叠**」等。

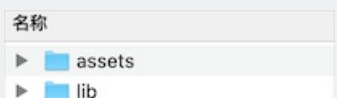
0x3、一点寒芒先到，随后枪出如龙 => 概念与名词

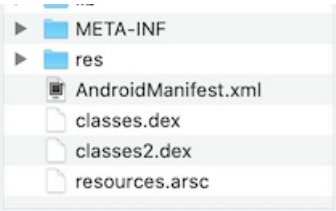
在开始折腾APP逆向前，先来了解一些概念与名词~

---

① APK文件里都有什么？

获取APK的渠道：**酷安**、**应用宝**，**豌豆荚**等应用市场**下载**，有些还提供「**应用历史版本**」下载。
APK本质上是一个「**压缩包**」，把「**.apk后缀**」改为「**.zip后缀**」**后解压**，可以看到如下**目录结构** (可能还有其他文件)：



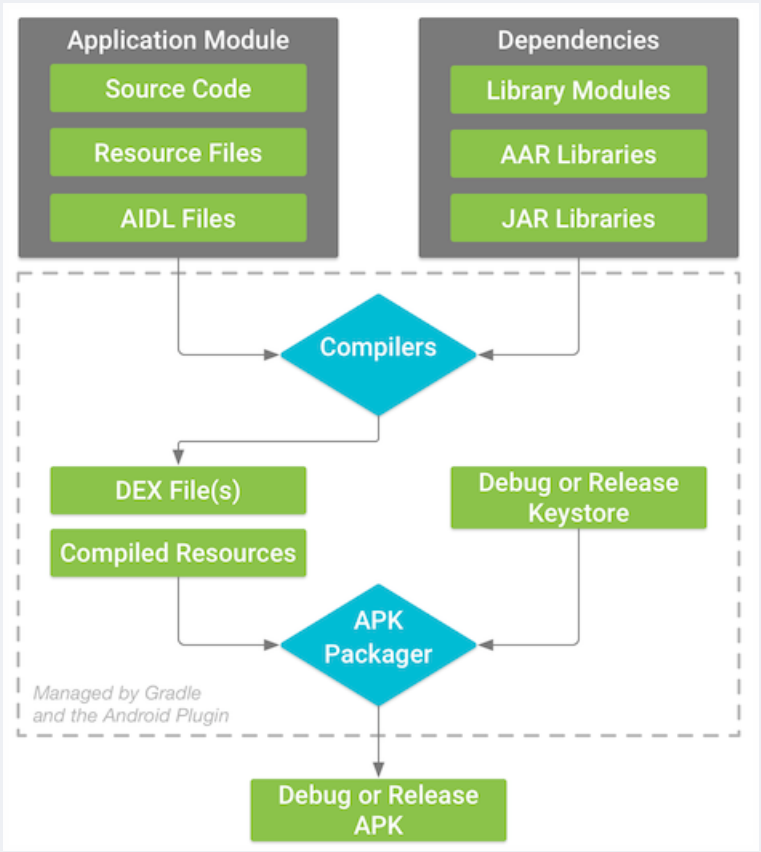


简单介绍下：

---

② 编译APK和反编译APK

所谓的「**编译**」，就是把「**源码、资源文件等**」按照一定的「**规则**」打包成APK，官网 提供了详细的**编译构建过程图：**



简述下大概流程：

Step 1：资源文件处理「**AAPT**」

- **assets**会原封不动地打包在APK中；
- **res**中每一个资源会赋予资源ID，以常量形式定义在R.java中，生成一个resource.arsc文件(资源索引表)。

Step 2：aidl文件「**aidl**」

- 将aidl后缀的文件转换为可用于进程通信的C/S端Java代码。

Step 3：Source Code「**Java Compiler**」

- 编译生成.class文件。

Step 4：代码混淆「**ProGuard**」(可选)

- 增加反编译难度，命名缩短为1-2个字母的名字，压缩(移除无效类、属性、方法等)，优化bytecode移除没用的结构。

Step 5：转换为dex「**dx.bat**」

- 把所有claas文件转换为classes.dex文件，class -> Dalvik字节码，生成常量池，消除冗余数据等。(方法数超65535会生成多个dex文件)

Step 6：打包「**ApkBuilder**」

- 把resources.arsc、classes.dex、其他的资源一块打包生成未签名apk。

Step 7：签名「**Jarsigner**」

- 对未签名apk进行debug或release签名。

Step 8：对齐优化「**zipalign**」

- 使apk中所有资源文件距离文件起始偏移为4字节的整数倍，从而在通过内存映射访问apk文件时会更快。

如果想了解更多编译构建流程可移步至：《10分钟了解Android项目构建流程》
而「**反编译**」则是反过来了，通过一些反编译工具，**提取出源码**，转换过程如下：

> 「**APK ====> Dex ====> Jar(class文件)/Smali ==> Java源码**」。

---

③ 加固和脱壳

APK可以说是每个Android开发仔的「**心血结晶**」，把**各种自己觉得「牛逼哄哄的奇淫巧技」**封装其中。但总有些「**心怀叵测**」想去搞你的APP，通过一些「**反编译工具**」获取你的源码，**然后为所欲为**：

> - **加广告**：你应用免费，给你加点广告，亦或者改成付费，然后下载量比你的多，气不气？
> - **破解付费**：你应用收费，Hook掉你的检测方法，发个破解包，还到处传播，气不气？
> - **恶意攻击**：逆向得出请求接口规律，批量短信验证注册，耗光你的短信池等，气不气？

这种「**恶劣**」的行径令人「**气愤**」像极了某类经典「**动作电影**」里的桥段：

> - 男子上进，努力工作，妹子贤惠，料理家务；
> - 妹子每天做好饭菜，等男子回来，一起吃饭，满怀憧憬，畅谈以后的二人世界；
> - 酒饱饭后，**温饱思XX**，不可描述一番，却被「**居心叵测**」的邻居给盯上了；
> - 和往常一样，男子出门上班，妹子在家做家务，晾衣服；
> - **邻居** 上线，用「**谎言**」诱骗妹子开门，接而挤门而入；
> - 用「**暴力和胁迫**」，无视妹子的やめて和反抗，违背个人意愿；

- 粗暴地把衣服一件件褪去，仅剩下那「**万恶的马赛克**」；
- 守护着最后的一处「**绝对领域**」；
- 在几番不可描述后，把妹子占为己有，然后像玩物般戏耍。

看着妹子「**因情绪过激而身体抽搐**」，哭得「**梨花带雨**」，不禁让人「**心生怜惜**」，像我这种感性的蓝孩子：

总会忍不住抽上几张**抽纸**，**“静静抹泪”**，擦拭完，顿觉**索然无味**，**一片空明**，然后开始反思：

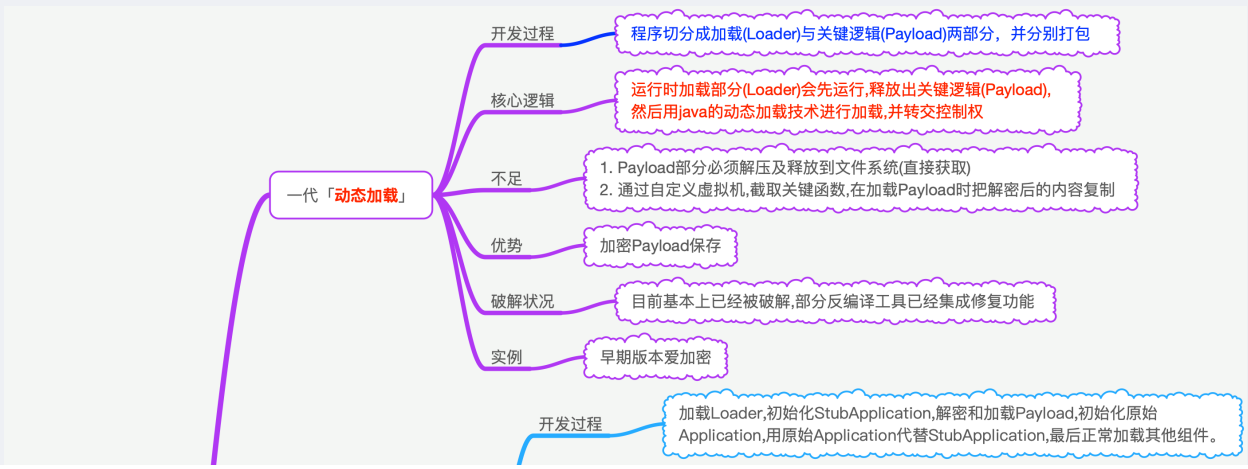
> **为什么那个邻居不是我**？呸呸呸...

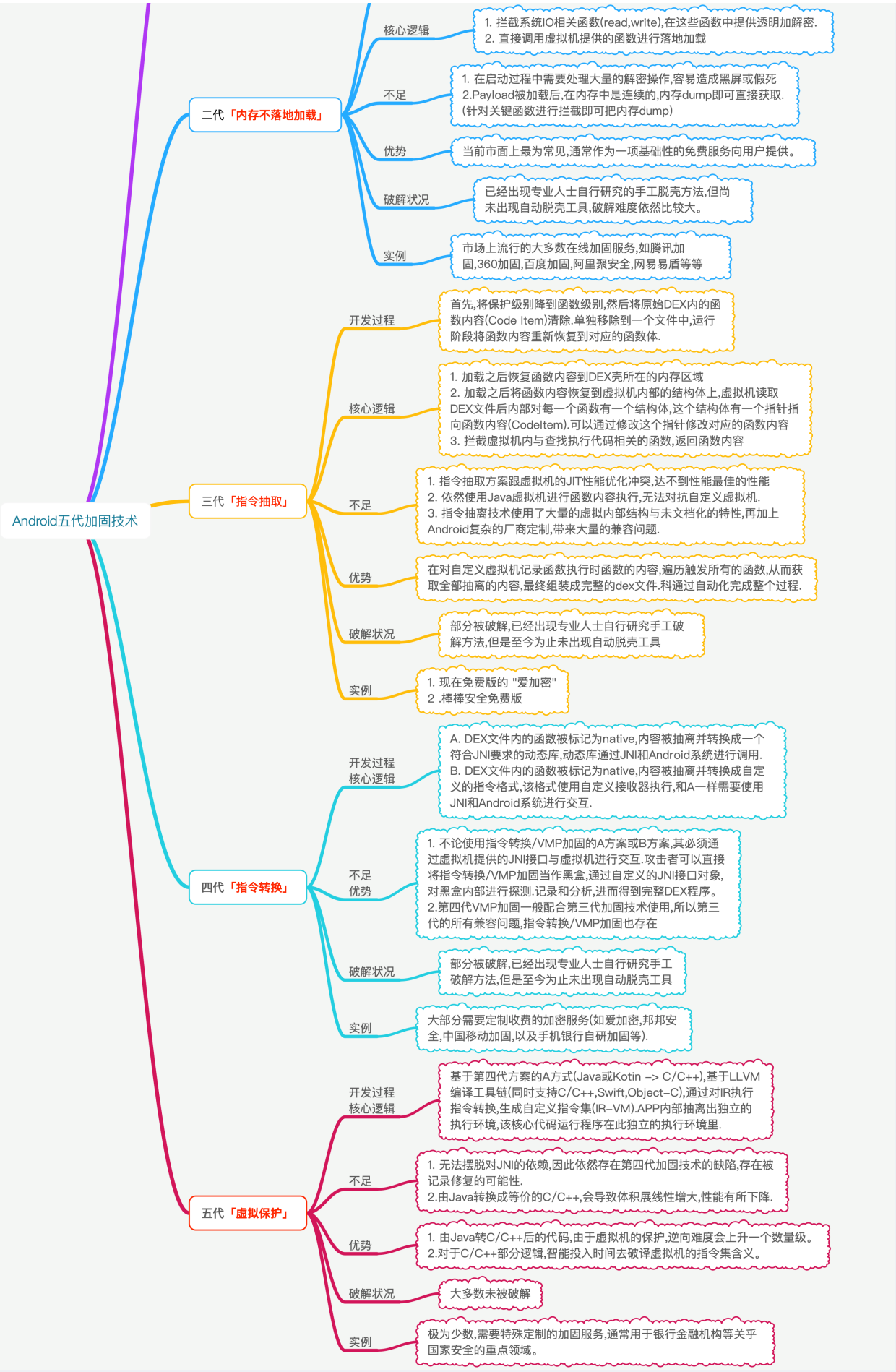
> 除了「**同情女主**」和「**斥责坏人**」外，应该如何避免这种事情的发生呢？
>
> - 1、**花点钱**，请个「**保镖**」看门，坏人想进来要先过保镖这一关；
> - 2、给妹子「**加个锁**」，让坏人无法不可描述，只能望而兴叹。

可以把例子中的「**妹子**」看做是我们编写的「**APK**」，而「**请保镖**」和「**加锁的操作**」则可以看做是「**APK加固**」，另外加固又称「**加壳**」，壳的定义：

> 一段专门负责「**保护软件不被非法修改或反编译的程序**」，一般先于程序运行，拿到控制权，然后完成它们保护软件的任务。

有「**加壳**」，自然也有「**脱壳**」，即去掉这层壳，拿到源码，也称为「**砸壳**」。
关于加固技术的发展，看雪上有篇：《一张表格看懂：市面上最为常见的 Android 安装包（APK）五代加固技术发展历程及优缺点比较》，不过图不怎么清晰，笔者重新排版了一下，有兴趣的读者可以看看：





④ 混淆与反混淆

「**混淆**」可以类比为上面「**万恶的马赛克**」，阻碍人类进步的绊脚石。而混淆则是增加了反编译的难度，同理，「**反混淆**」则对应「**去除马赛克**」，试图还原它原来的样子。

0x4、发动机已启动，随时可以出发 => 获得APP源码

加固虽然能在一定程度上「**防止反编译和二次打包**」，但加固后的APP可能会带来一些问题：

> 体积增大，启动速度变慢，兼容问题等

网上「**免费加固**」方案有很多，脱壳教程也是烂大街，而且有些恶心的第三方加固还会给你加点料(360加固锁屏广告)，而使用「**企业级的加固**」，则需要支付不菲的费用，所以很多APP直接选择了「**裸奔**」。先来讲解一下未加固的怎么获取源码吧~

① 未加固(笔者使用的工具：apktool + jadx)

- 使用**apktool**：获取「**素材资源，AndroidManifest.xml以及smail代码**」
- 使用**jadx**：把「**classes.dex**」转换为「**.java**」代码

使用Jadx的注意事项：

> 使用**jadx-gui**可直接打开apk查看源码，但如果**APK比较大**(classes.dex有好几个)，会直接**卡死**(比如微信)，笔者的做法是命令行一个个dex文件去反编译，最后再把反编译的文件夹整合到同一个目录下。

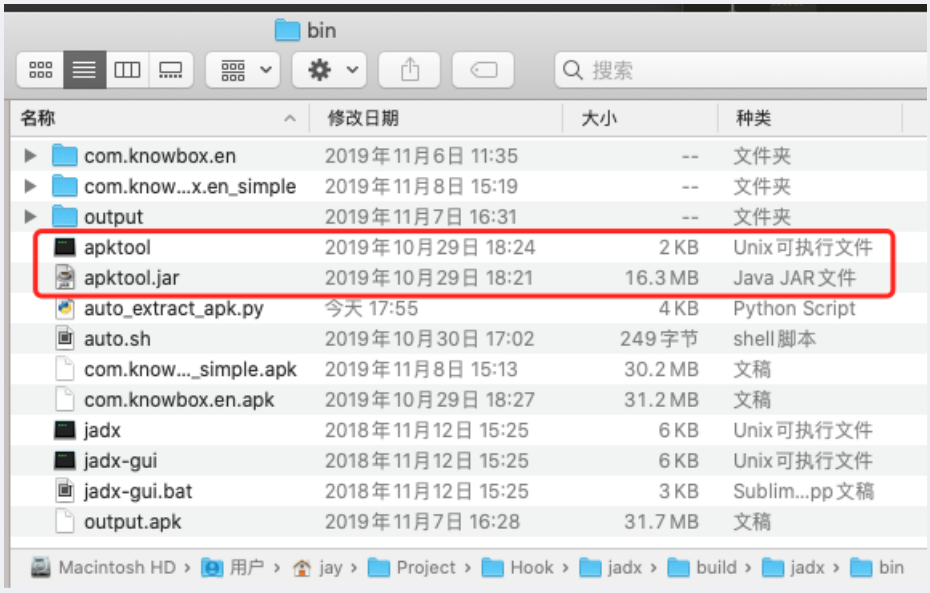
这样的操作繁琐且重复，最适合批处理了，遂写了个反编译的**批处理脚本**(取需)：

```
"""
自动解压apk，批量使用jadx进行反编译，结果代码汇总
"""
import os
import shutil
import zipfile
from datetime import datetime

apk_file_dict = {}  # APK路径字典

# 遍历构造APK路径字典(构造文件路径列表，过滤apk，拼接)
def init_apk_dict(file_dir):
    apk_path_list = list(filter(lambda fp: fp.endswith(".apk"),
                                list(map(lambda x: os.path.join(file_dir, x), os.listdir(file_dir
)))))
    index_list = [str(x) for x in range(1, len(apk_path_list) + 1)]
    return dict(zip(index_list, apk_path_list))


# 移动文件夹
def move_dir(origin_dir, finally_dir):
    shutil.move(origin_dir, finally_dir)


# 如果文件夹存在删除重建
def deal_dir_existed(path):
```

```
def deal_dir_existed(path):
    if os.path.exists(path):
        print("检测到文件夹【%s】已存在，执行删除..." % path)
        shutil.rmtree(path)
    os.makedirs(path)


# 判断目录是否存在，不存在则创建
def is_dir_existed(path, mkdir=True):
    if mkdir:
        if not os.path.exists(path):
            os.makedirs(path)
    else:
        return os.path.exists(path)


# 获取目录下的所有文件路径
def fetch_all_file(file_dir):
    return list(map(lambda x: os.path.join(file_dir, x), os.listdir(file_dir)))


# 解压文件到特定路径中
def unzip_file(file_name, output_dir):
    print("开始解压文件...")
    f = zipfile.ZipFile(file_name, 'r')
    for file in f.namelist():
        f.extract(file, os.path.join(os.getcwd(), output_dir))
    print("文件解压完毕...")


if __name__ == '__main__':
    print("遍历当前目录下所有APK...")
    apk_file_dict = init_apk_dict(os.getcwd())
    print("遍历完毕...\n\n============ 当前目录下所有的APK ============\n")
    for (k, v) in apk_file_dict.items():
        print("%s.%s" % (k, v.split(os.sep)[-1]))
    print("\n%s" % ("=" * 45))
    choice_pos = input("%s" % "请输入需要反编译APK的数字编号：")
    print("=" * 45, )
    choice_apk = apk_file_dict.get(choice_pos)
    apk_name = choice_apk.split(os.sep)[-1][:-4]  # APK名字

    # 创建相关文件夹
    crack_dir = os.path.join(os.getcwd(), apk_name)  # 工程根目录
    deal_dir_existed(crack_dir)
    crack_apktool_dir = os.path.join(crack_dir, "apktool" + os.sep)  # APKTool反编译目录
    deal_dir_existed(crack_apktool_dir)
    crack_jadx_dir = os.path.join(crack_dir, "jadx" + os.sep)  # JADX反编译目录
    deal_dir_existed(crack_jadx_dir)
    crack_temp_dir = os.path.join(crack_dir, "temp" + os.sep)  # 解压后文件的临时存储路径
    deal_dir_existed(crack_temp_dir)

    # 利用APKTool提取资源文件
    begin = datetime.now()  # 计时
    print("APKTool提取资源文件...")
    os.system("./apktool d %s -f -o %s" % (choice_apk, crack_apktool_dir))

    # 复制一份AndroidManifest.xml、res、assets文件到外部
    shutil.copy(os.path.join(crack_apktool_dir, "AndroidManifest.xml"), os.path.join(crack_dir, "
AndroidManifest.xml"))
```

```
    shutil.copytree(os.path.join(crack_apktool_dir, "res" + os.sep), os.path.join(crack_dir, "res
" + os.sep))
    shutil.copytree(os.path.join(crack_apktool_dir, "assets" + os.sep), os.path.join(crack_dir, "
assets" + os.sep))
    print("资源文件提取完毕")

    # 利用jadx反编译源码
    print("JADX反编译提取源码...")
    choice_apk_zip = shutil.copy(choice_apk, choice_apk.replace(".apk", ".zip"))
    unzip_file(choice_apk_zip, crack_temp_dir)
    print("开始批量反编译dex文件")
    for dex in list(filter(lambda fp: fp.endswith(".dex"), fetch_all_file(crack_temp_dir))):
        os.system(
            "./jadx -d {0} {1}".format(os.path.join(crack_jadx_dir, dex.split(os.sep)[-1][:-4]),
dex))
    print("所有dex文件反编译完毕")
    # 将资源文件移入
    shutil.move(os.path.join(crack_dir, "AndroidManifest.xml"), os.path.join(crack_jadx_dir, "And
roidManifest.xml"))
    shutil.move(os.path.join(crack_dir, "res" + os.sep), os.path.join(crack_jadx_dir, "res" + os.
sep))
    shutil.move(os.path.join(crack_dir, "assets" + os.sep), os.path.join(crack_jadx_dir, "assets"
 + os.sep))
    # 删除临时文件夹，压缩文件
    shutil.rmtree(crack_temp_dir)
    os.unlink(choice_apk_zip)
    end = datetime.now()
    print("收尾操作~~~\n反编译完成，总耗时：%s秒" % (end - begin).seconds)
```

执行前，你需要把apktool相关的东西，丢到jadx/build/jadx/bin目录下，如图所示：



接着终端键入：**python3 auto_extract_apk.py**，回车后输入对应编号，回车开始编译：

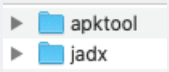
```
4.output.apk

================================================
请输入需要反编译APK的数字编号：4
```

静待片刻后：

```
ERROR -   Method: com.mob.tools.util
ssage):boolean
WARN  - 1911 warnings in 399 nodes
ERROR - finished with errors
所有dex文件反编译完毕
收尾操作~~~
反编译完成，总耗时：76秒
```

> **Tips**：这里没有把多个classes文件夹整合到一起，是因为有些APP会出现合并冲突。

打开反编译后的目录，有如下两个文件夹：



按照自己的需要用Android Studio打开其中一个就好了：

> - **apktool目录**：apktool反编译后的内容，主要用于**smail动态调试**。
> - **jadx目录**：反编译成Java的内容。

---

② 反混淆(simplefy、Deguard)

> 代码是拿到了，但是打开代码，「**一堆的abcd**」，跟到眼花，可以试下「**反混淆**」，方案有两类，一种是通过「**代码逆推**」出名字，另一种是通过「**统计逆推**」出名字。

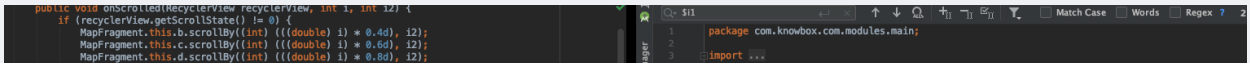
第一种方案的工具有很多（**Jeb2**，**simplify**等），前者付费需破解，Java版本有限制，Mac配置有点麻烦，故笔者用的是后者：「**Simplefy**」，**Github仓库**：https://github.com/CalebFenton/simplify，使用方法也很简单：
打开终端依次键入:

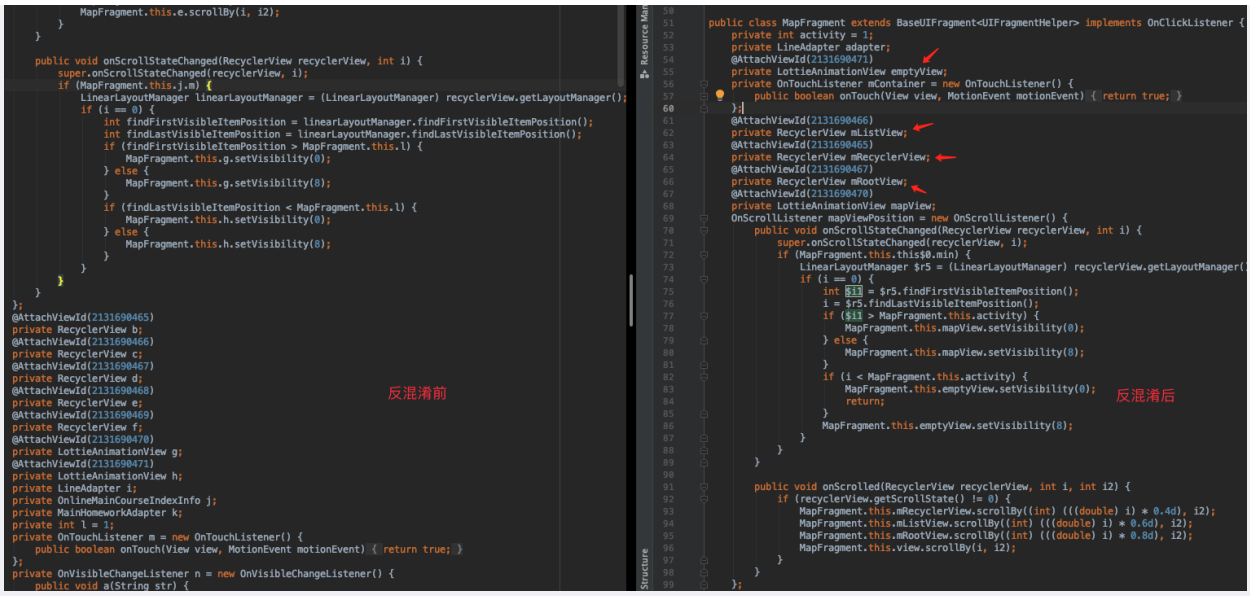
```
# 拉取仓库代码
git clone --recursive https://github.com/CalebFenton/simplify.git

# 来到目录下
cd simplify

# 编译
./gradlew fatjar
```

编译后完，执行下述指令即可反混淆APK：

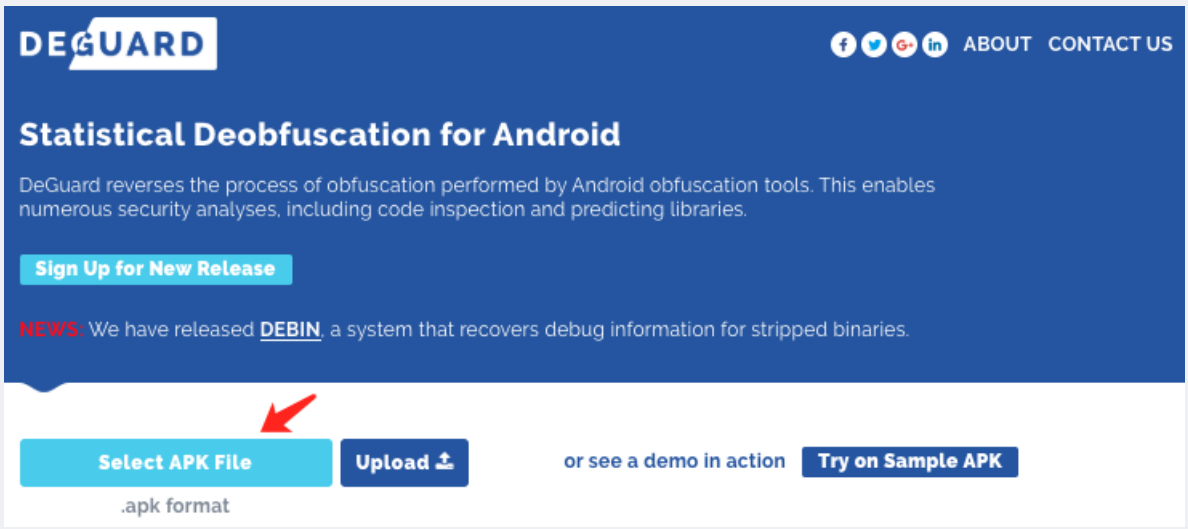
```
# 反混淆APK（需要反混淆的APK，反混淆后的APK名）
./gradlew build && cp xxx.apk yyy.apk
```

静待反混淆完毕，接着反编译批处理脚本走一波，打开MapFragment比对下：

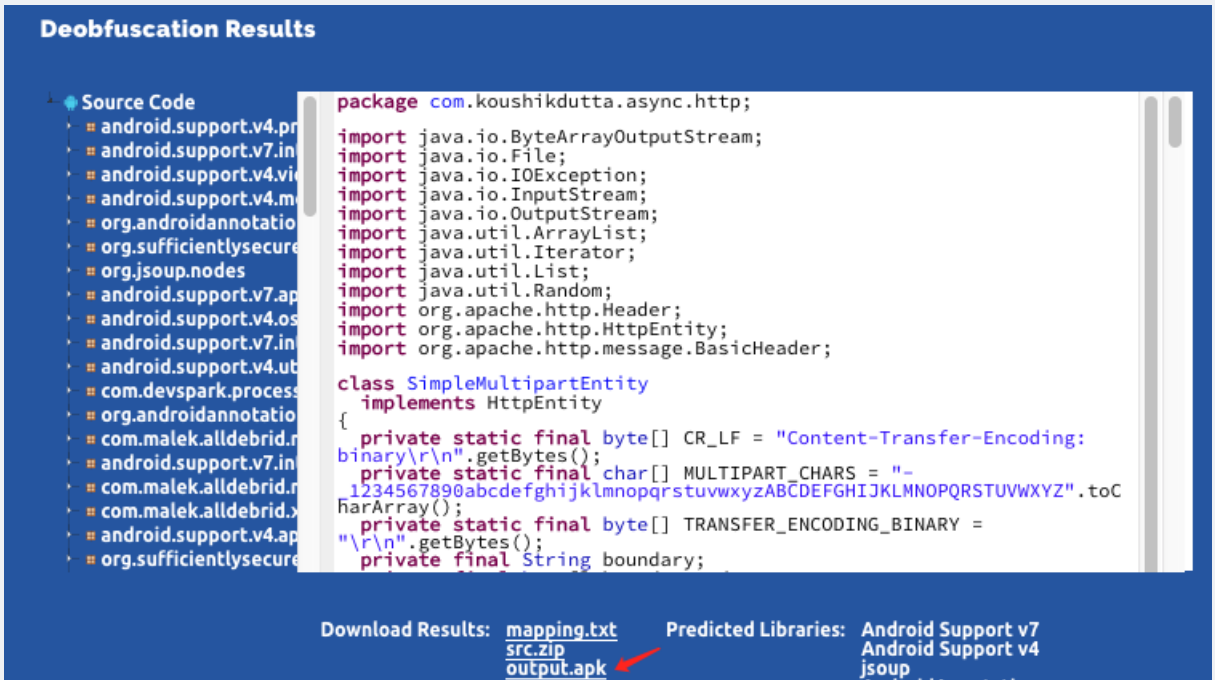




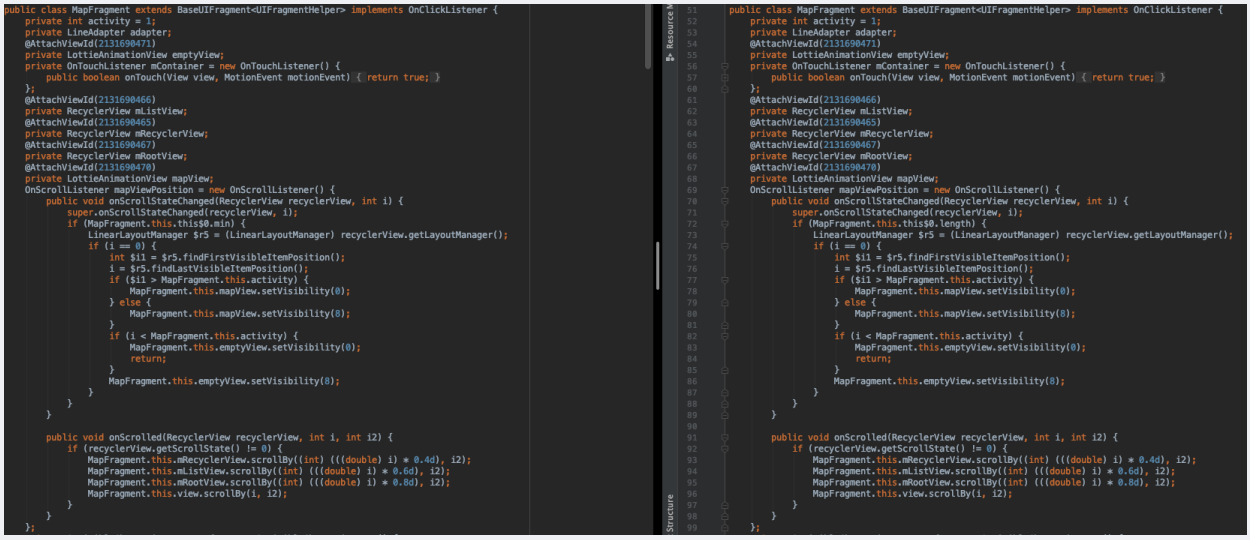
相比混淆前，多了一些变量名，当然也不是完全的，偶尔还是有abcd，但是可读性稍微提高了些，比如查找的时候不用在一个个adcd排除，但是，编译挺耗时的，而且我的电脑风扇呼呼呼地响。
第二种是通过统计的方法，利用统计推断出名字：**DEGUARD**：http://apk-deguard.com/，打开官网：



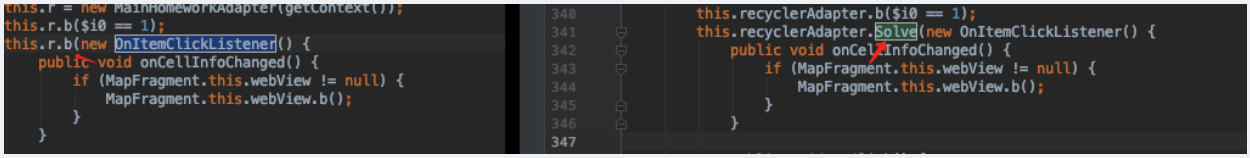
选择需要反混淆的APK后，Upload上传，接着等待处理完成，！！！**别关页面！！！**

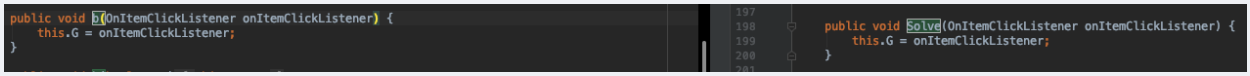


一般需等待1-10分钟，处理完成后，点击output.apk，把APK下载到本地，同样执行批处理脚本反编译一波，和simplefy反混淆后的代码对比下：



大同小异，另外，**反混淆并不能100%还原**，而且还可能**有些小错误**，比如下面的代码：





虽说反编译后的可读性有所提高，但建议还是搭配着混淆的源码看。

---

③ 脱壳（FDex2，反射助手，dumpDex）

终于来到很多同学期待的脱壳环节，先说明下，笔者只是「**工具党**」水平，不会Native层的，so文件调试！如果本节的工具，你脱不出来，或者脱出来有问题，笔者也是爱莫能助。看雪有很多帮人脱壳的大佬，可以在上面发个帖子求助下~

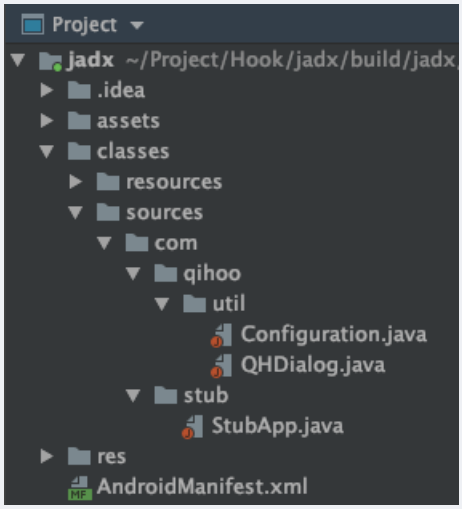
1、判断是哪种加固

> 解压apk后在assets目录下看到so文件，比如360加固宝：libjagu.so和libjiagu_x86.so，百度搜下名字就知道是哪家的加固了，也可以直接用后面讲的「**MT文件管理器2.0**」直接查看。

2、**FDex2脱壳（只适用于Android 7或以下版本**，可以脱市面上大多数免费加固，成功率较高，推荐）

> - **有ROOT**：安装「**XposedInstaller**」和「**FDex2**」
> - **没ROOT**：安装「**VirtualXposed**」「**FDex2**」

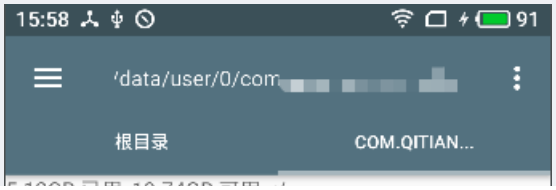
比如：这里有个「**360免费版加固的APK**」，直接用jadx反编译后导入AS，但是反编译后的classes：



只有这么一丢丢点东西，把「**待脱壳应用**」安装到手机上，接着用**FDex2来脱壳**
已Root玩家：XposedInstall启用FDex2插件重启后，按如下步骤脱：

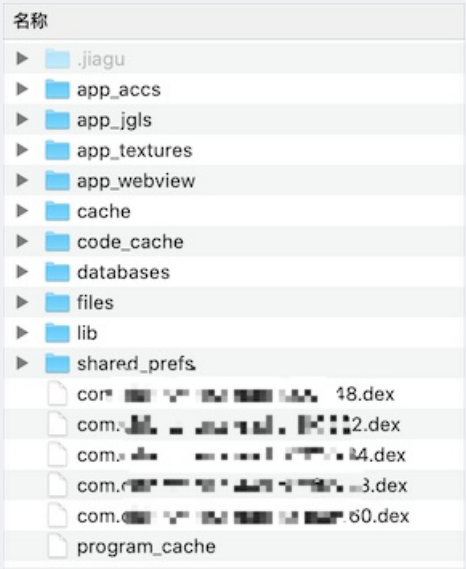
- **Step 1**：FDex2选中待脱壳应用：

- **Step 2**：打开待脱壳应用，接着来到上图的dex输出目录：



- **Step 3**：把整个目录拉到电脑上，这里直接用adb命令拉取：

```
adb root
adb pull /data/user/0/包名 电脑文件夹
```

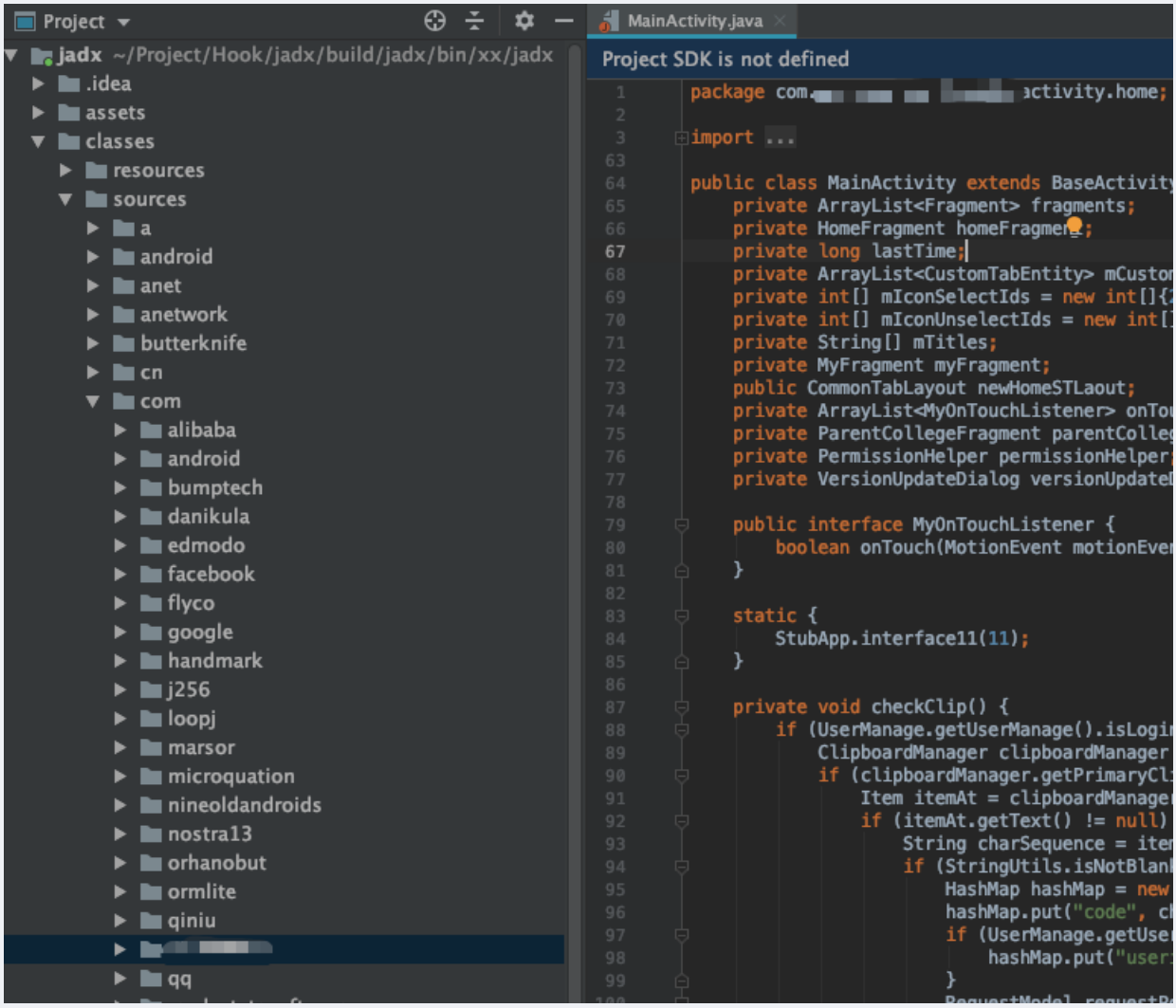


- **Step 4**：「**剔除加固相关的dex**」，用jadx-gui依次打开，看到下面这种，直接把dex删掉

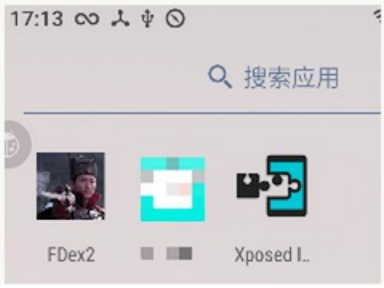
- **Step 5**：使用jadx命令反编译dex，顺带改名，命名规则：**按照文件大小降序**，示例如下：

```
# 按照文件从大到小排序！！！
jadx aaa.dex -d classes
jadx bbb.dex -d classes1
jadx ccc.dex -d classes2
```

- **Step 6**：删掉没脱壳前反编译项目里的classes，把这几个复制到其中：

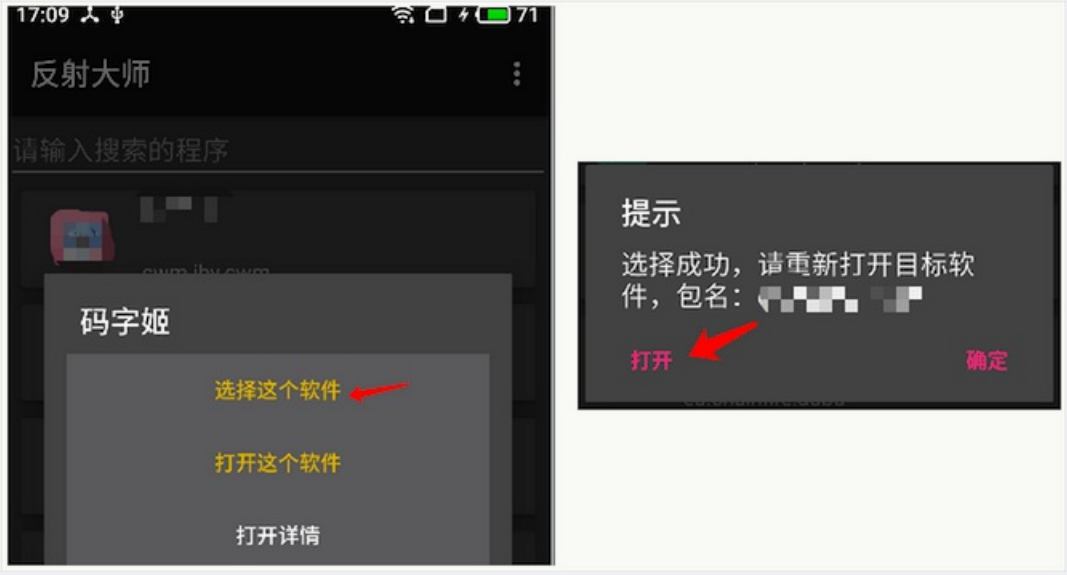


行吧，**脱壳成功**，这里其实还可以还原APK的(二次打包)，等下再讲~
未root玩家，安装打开VirtualXposed，添加应用：**Fdex2和待脱壳应用**



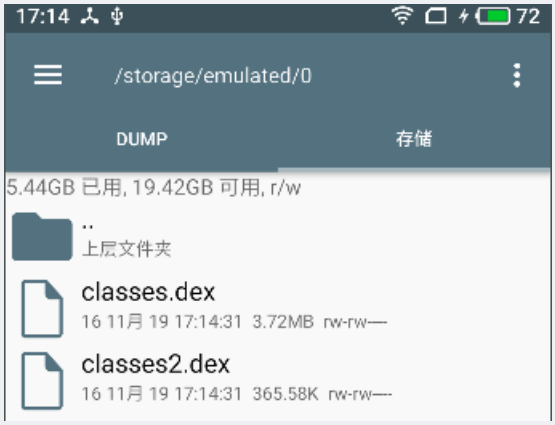
如果炮制，只是dex的路径有些不一样。

3、**反射大师**（和FDex类似，**下载地址**：https://www.lanzous.com/b04xxlujg）
注意，同样只支持Android 7.0及以下，adb安装后，xposed启用插件，重启手机，接着打开反射大师：
Step 1：选中待脱壳APP，弹出对话框选择打开

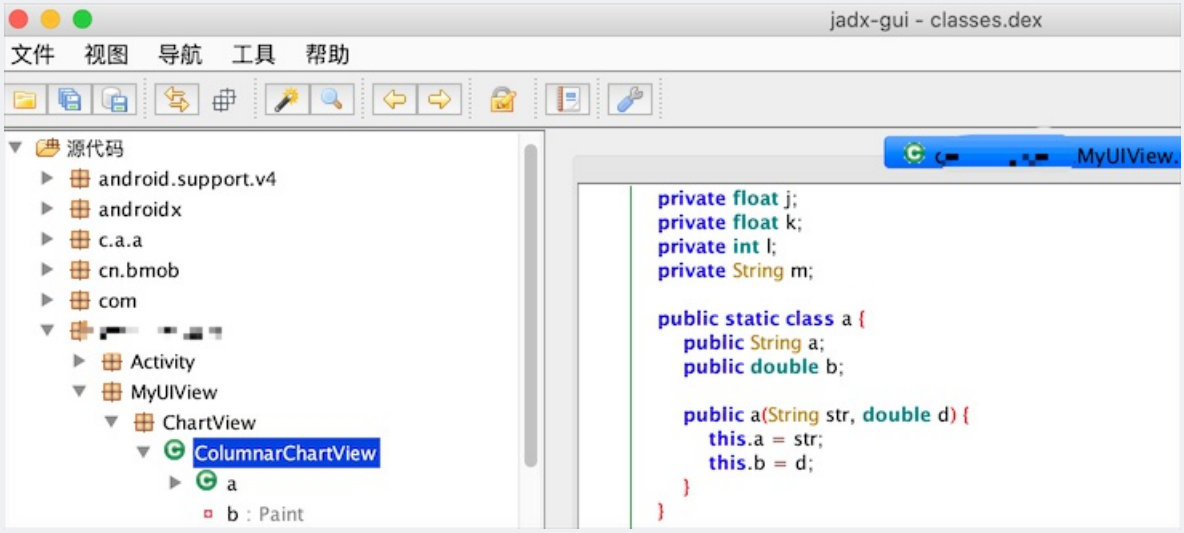


Step 2：点击中间的六芒星，弹出如下对话框，**长按「写出DEX」**

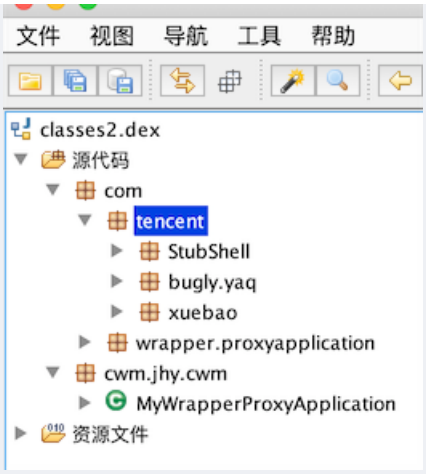
Step 3：等待写出完毕，可以在/storage/emulated/0中找到导出的dex：



Step 4：pull到电脑上用jadx-gui打开看看：



行吧，脱壳成功，就是我们想要的dex了，另一个classes2.dex则是相关的~：

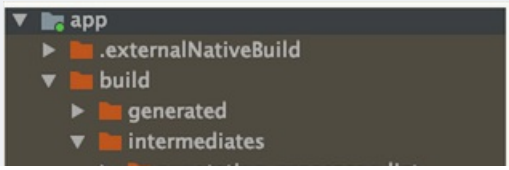


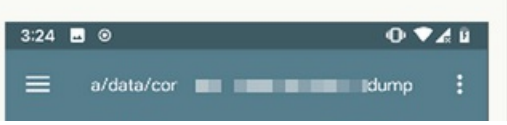
4、**dumpDex脱壳（Github：**https://github.com/WrBug/dumpDex/releases）
官方仓库的README.md中有一句：

apk文件不会实时更新，获取最新apk请自行编译源码

可以的话**建议自己编译**，流程也很简单：

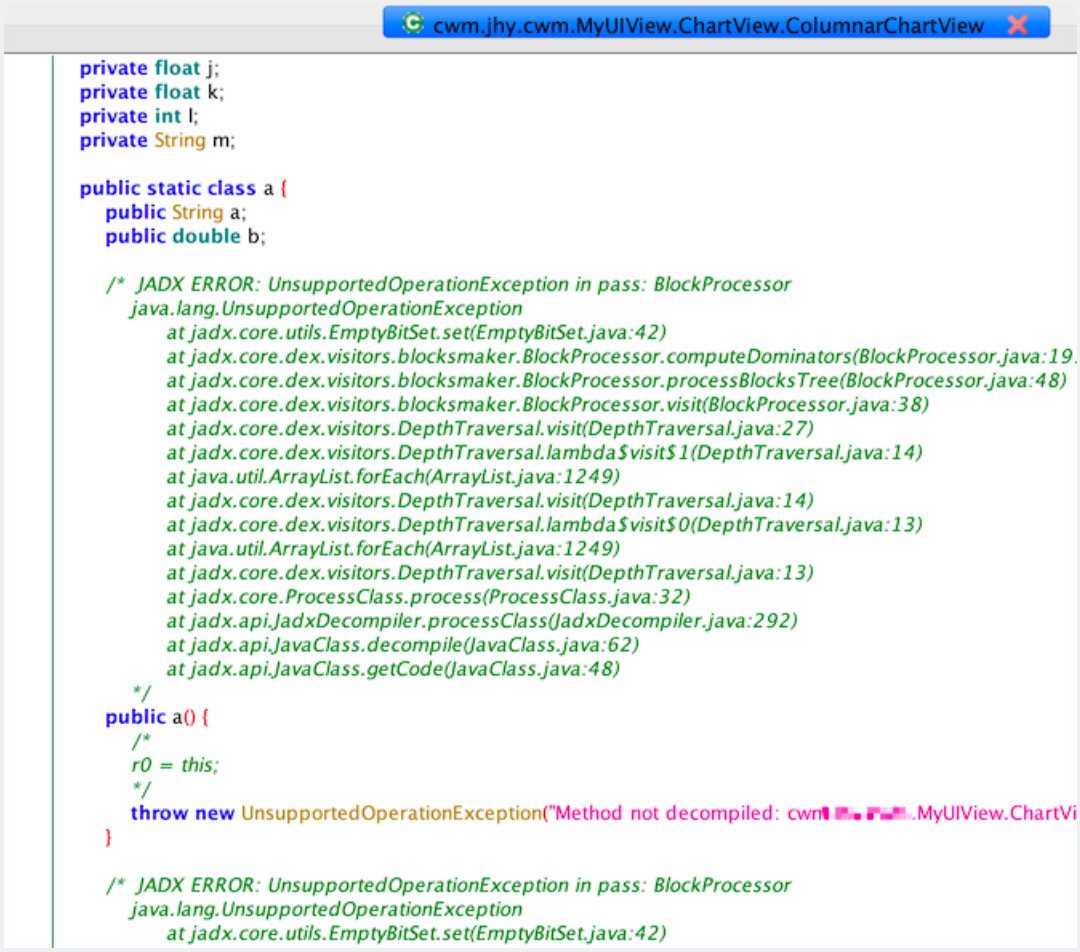
```
# 1、拉取项目代码到本地
git clone https://github.com/WrBug/dumpDex.git

# 2、AS中Open项目，等待编译完成

# 3、删掉build.gradle里签名相关的代码

# 4、点击顶部菜单栏Build -> Build APK，或者直接在终端./gradlew clean build

# 5、adb命令直接把编译生成的apk安装到手机上

# 6、接着来到如下左图路径，把对应的so，通过adb push到目录下：
adb push lib/armeabi-v7a/libnativeDump.so /data/local/tmp
adb push lib/arm64-v8a/libnativeDump.so /data/local/tmp/libnativeDump64.so
# 修改权限
adb shell
su
chmod 777 /data/local/tmp/libnativeDump.so
chmod 777 /data/local/tmp/libnativeDump64.so
# 临时关闭SELinux(重启后会失效，可调用getenforce查询)
setenfore 0

# 7、打开XposedInstaller看已经启用DumpDex插件，是的话重启手机

# 8、开机后，打开想脱壳的应用，不用理闪退，接着打开data/data/包名查看是否有Dump目录

# 9、进入如果出现下图所示的多个dex，说明脱壳成功，否则可能是脱壳失败
# (看是否有报错信息)，或者不支持(比如360加固免费版只支持新版，不支持旧版)。
```



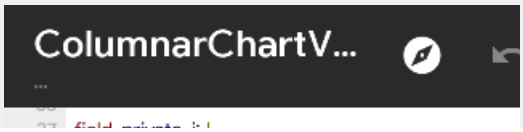


另外，脱出来的dex不一定就可用，比如某个用了「**腾讯御安全**」的应用：

用jadx-gui打开这的dex，一堆这样的错误：



出现这个的原因是「**指令集被抽取**」，打开smail文件你就知道了：



```
.field private i:J

.field private j:F

.field private k:F

.field private l:I

.field private m:Ljava/lang/String;


# direct methods
.method public constructor <init>(Landroid/conter
    .registers 4

    nop

    nop

    nop

    nop
```

方法指令都被**nop(零)替换**了，工具党到这里就可以放弃了，要调试so文件。

> **Tips**：本节用到的东西，**都有给出比较官方的下载链接！！！**你也可以到公号「**抠腚男孩**」输入000，回复对应序号下载，谢谢~

参考文献：

- Android打包流程
