随笔：公司App升级第三方SDK踩坑两周小记

CoderPig 郭霖 2023-01-06 08:00 发表于江苏

/ 今日科技快讯 /

近日，亚马逊CEO安迪·贾西证实，该公司计划在去年11月宣布和今日媒体曝出的裁员中裁撤略多于1.8万个岗位，将从1月8日起与受裁员影响的员工沟通。贾西表示，有多个团队受到影响，但大部分裁员涉及亚马逊商店与人员、体验和技术（PXT）部门。

/ 作者简介 /

明天周六啦，天气越来越冷，大家注意做好防护！

本篇文章转自coder_pig的博客，文章主要分享了他在升级第三方SDK踩坑并解决的过程，相信会对大家有所帮助！

原文地址：

> https://juejin.cn/post/7130532039870119973

/ 引言 /

如题，最近两周疲于折腾公司第三方SDK升级，掉坑里一直出不来，前天突发奇想，终于定位到问题原因。

问题很简单，却花费了这么长时间，终究还是我太菜了！

BUG虽然解了，但觉得还是有必要记录复盘下，以便下次遇到类似问题时，能找到更好的切入点，更快地把问题解决。

问题概述

现在哪个房产类APP不上VR全景图啊，所以我司也整了一个，经纪人录盘时，使用全景相机对盘源进行拍摄，在录盘填信息的同时，把全景图也上传到服务器。而他们录盘的过程是这样的：

- 使用全景相机官方APP，连接相机，对盘源进行拍摄，官方APP拼接生成全景图；
- 拍完打开公司APP，来到录盘页，填写录盘信息，然后打开手机相册，选择对应的全景图上传；

录一次盘得在两个APP间来回切换，可以是可以，但流程繁琐，效率也低，他们更希望：

直接就在公司APP上就可以完成相机连接、拍摄、拼接和上传。

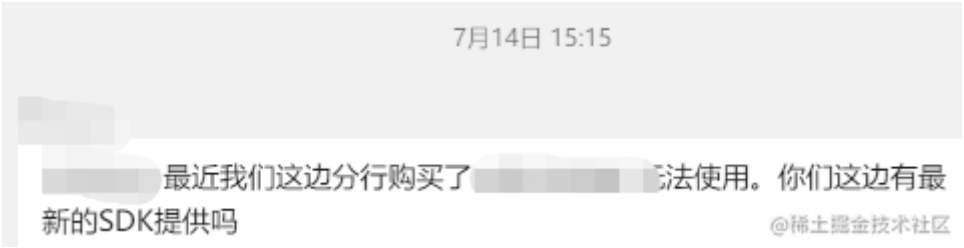
任务传到客户端这边，所以我们需要：

集成官方SDK，参考官方Demo和文档，根据具体业务进行二次开发。

杰哥徒手封装的Library已经稳定跑了两年：

feat: VR相机包修改	jay	2020-07-23 17:31
fix: 版本号修正	jay	2020-07-22 20:26
fix: 删除混淆，更新VR相机 Maven版本号	jay	2020-07-22 19:42
fix: 新增VR相机混淆文件+升级maven版本号	jay	2020-07-22 16:00
feat: AR相机包修改	jay	2020-07-21 16:38

本以为可以一劳永逸，安享晚年，最近却出了幺蛾子：



有经纪人买了新款相机，然后用公司APP无法拍照。

em...硬件、固件升级，需要更新下SDK，可以理解，那就升吧，正当我以为这次升级，会像当初刚集成时那样丝滑流畅，却是磨人踩坑的开始...

/ 第一道坎：SDK自带预览组件黑屏 /

很快新的SDK到位，Library替换上新AAR，API有些变动，跟着Demo改改，算是跑起来了，但问题也来了：

拍摄预览页 → 关闭并跳转新页面 → 再次打开预览页 → 预览组件黑屏

看了下官方Demo有定义这个预览组件，但却没用到，再看一下文档：

9. 获取相机预览流数据

（1） `HZCameraPreviewer` 类用于获取和展示相机的预览流数据。
初始化预览控件，可直接在layout的xml文件中配置进行初始化，或者直接调用初始化函数：

```
public HZCameraPreviewer(Context context, HZIFrameCallback frameCallback)
```

（2）开始和结束预览，在Activity的onResume和onPause方法中调用HZCameraPreviewer实例的对应方法：

```
@Override
protected void onResume() {
    super.onResume();
    mPreviewer.onResume();
}

@Override
protected void onPause() {
    super.onPause();
    mPreviewer.onPause();
}
```

（3）预览流接口原始数据回调接口：

```
public interface HZIFrameCallback {
    void onFrame(byte[] frameData, int dataLen, int width, int height);
}
```

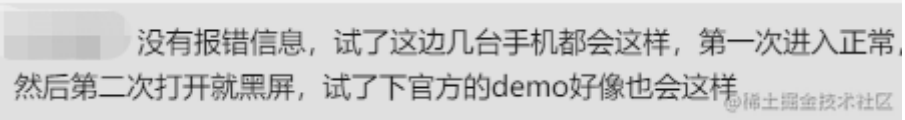
> 接口说明：
> *onFrame*: rgb数据字节数组

（4）刷新预览UI，在预览流数据回调接口中调用HZCameraPreviewer实例的 `requestRender` 方法：

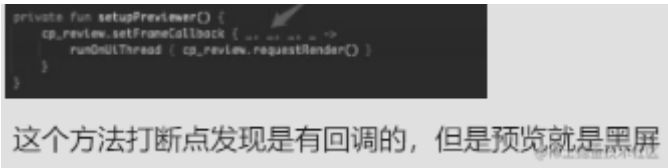
```
@Override
public void onFrame(byte[] frameData, int dataLen, int width, int height) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            mPreviewer.requestRender();
        }
    });
}
```


难不成是销毁时还要手动去释放什么资源么？上个版本没这个问题啊，尝试寻求技术支持：

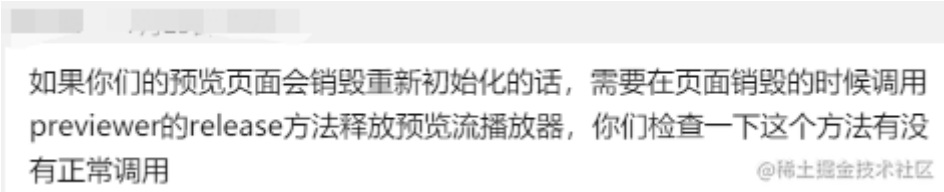
隔天早上，我又试着改了下官方Demo，加上预览组件部分的代码，也会黑屏，再次发问：



在等待回应的同时，我还对可能有关的方法调用都加上了日志打印，尝试找出问题原因：



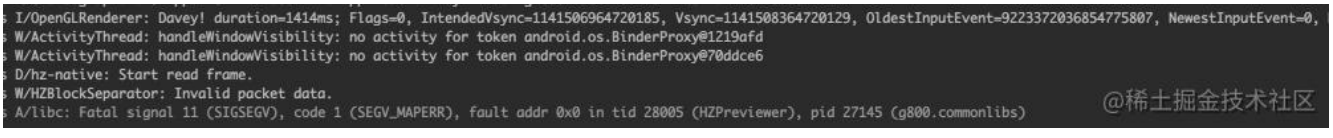
对面说周一开早会，晚点看问题，结果等到了第二天中午：



按照他说的，在页面销毁时主动调用预览组件的releasePreview()，预览的确正常了，问题好像解决了？但这样的操作，却引起了第二个问题...

/ 第二道坎：小米机型闪退 /

页面销毁和创建多次会报异常崩溃，有时甚至发生在图片拼接时，再次发问，并提供机型和报错日志：



可能是我表达能力的问题，对面有点懵了：

不好意思，我这边的信息有点混，之前你们反馈的是预览流页面反复打开会crash，今天你们提到反复拍摄会crash，我这边从日志里看起来是在拼接的时候crash了。
我想先确认一下，预览流展示现在是不是已经没有问题了？

重新组织了一下语言并附上更详细的奔溃日志，同时把demo中的改动文件发给他们测试：

[redacted] 您好，我大概捋下问题信息：

1、新相机使用原先的SDK(1.0.1)拍照后，图片下载到手机会失败，错误码：1057 (加载图片模式不支持)
2、将SDK升级到2.4.0，图片可以下载到手机，但发现预览View会有问题(页面销毁、跳转新页面、点击跳转新页面，预览黑屏)
3、尝试在onDestory 主动调用 setFrameCallback(null) 和 releasePreview() 预览黑屏问题解决
4、在图片拼接时，小米机型会奔溃，其他机型正常，然后有一个奇怪的现象，不用这个预览View就正常了；
5、看了下sdk demo，预览View有定义，但并没有真正使用；
6、尝试xml引入，拍照完成跳转相机连接页面，点击跳转拍照页面，会出现预览View黑屏；
7、同样主动调用释放方法，黑屏解决，但图片拼接时奔溃问题时有发生，有时拍照中途就奔溃了，奔溃日志如下：

得到的回复：

你们这个改过的demo，就是点击连接 -> 拍照 再重复吗？中间还有没有其他操作？
我这边替换了你们修改过的文件，找了两台小米手机（mi play和mi 5s），按照上面的操作重复了30多次，没有出现crash

em...重复了30多次，看得出对面也很想帮我们解决问题，奈何鞭长莫及。

我甚至跟领导开玩笑说，要不把测试机寄过去那边给他们排查下吧，哈哈哈！另外，同事说他们这两款机型都是很老的机型了，会不会是CPU架构的问题，于是在build.gradle中指定了架构：

```
ndk {
   abiFilters 'armeabi-v7a', 'arm64-v8a'
}
```

运行后发现并没什么卵用，而网上搜了一圈这个异常：

```
Fatal signal 11 (SIGSEGV), code 1 (SEGV_MAPERR), fault addr 0x10 in tid 2261
```

发现一个唯一有用且这边能做的事情，删掉这个文件夹 → /data/local/tmp/perfd，删了可以，但治标不治本，删完下次又得删，麻烦且不说，公司APP也没有访问这个文件夹的权限啊(root权限)。

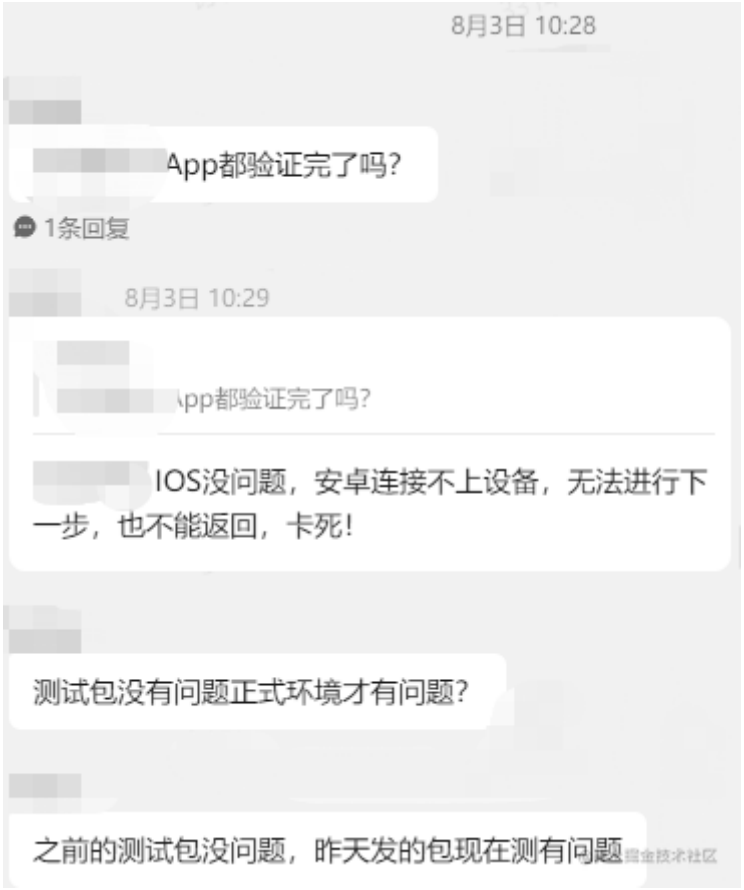
唉，无解，关键代码都写在so库里，就一黑盒子，奈何笔者太菜，不会so库逆向调试，只能到这里了...

```
class HZGLFrameRenderer implements Renderer {
    private j a;
    private HZIFrameCallback b;
    private HZICalibratedFrameCallback c;

    private native void nativeGLESInit(float var1);

    private native void nativeGLESRender();

    private native void nativeGLESResize(int var1, int var2);

    private native void nativeStartReadStream(String var1);

    private native void nativeStopReadStream();

    private native void nativeRelease();
```

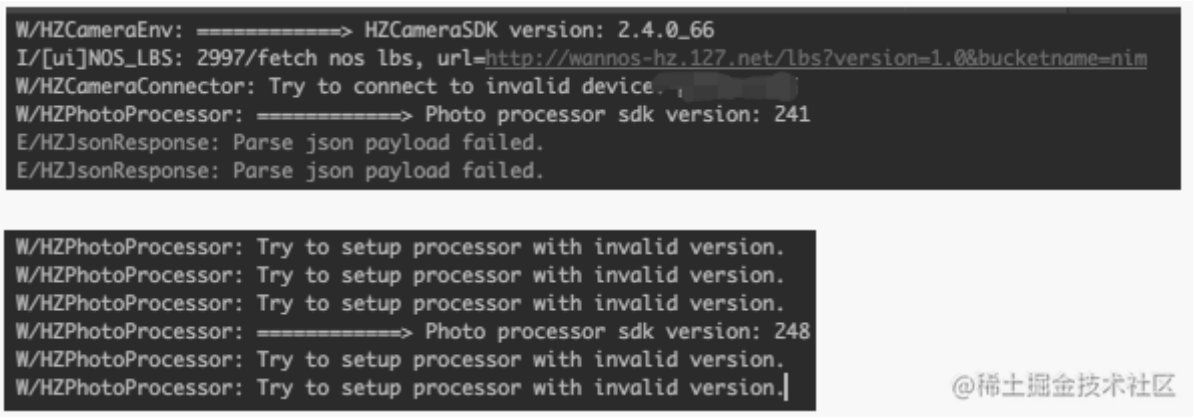

后面领导说先这样把，目前就一个经纪人用新相机，而且用的华为手机，打包给测试测下，没其它问题就先发包吧~

/ 第三道坎：打Release包无法拍摄 /

隔空艰难对接一周，最后妥协收场，正当我以为此事了解，可以回归摸鱼日常，结果测试一声惊雷平地起：



焯！跟进了一下发现：Debug包可以，Release就不行，旧相机可以，新相机不行，看下日志：



果断再次向技术支持请求援助，这次吸取教训， 把问题整理清楚再提问：

唉，大概是因为人类的悲欢并不相通吧，并没有得到有效帮助，但也给了我们一个思路，可能是混淆的问题。

但混淆代码没Keep住，调用时不是报ClassNotFound之类的Exception吗？难不成他们用try-catch包住不往外抛？半信半疑，不过眼下好像没有更好排查方向了，死马当活马医吧，这也是钻牛角尖的开始...

混淆排查

说到混淆，这不得打开：《补齐Android技能树 - 从害怕到玩转Android代码混淆》，边看边排查，首先混淆规则是叠加的：

3. 混淆规则的叠加

不知道你有没有想过：**上面日常使用的创建的代码示例，proguard-rules.pro没有配置混淆规则，却混淆了？**

其实是因为 **混淆规则是叠加的** ，而混淆规则的来源不止主模块里的proguard-rules.pro，还有这些：

- ① `<module-dir>/proguard-rules.pro`

不止主模块有proguard-rules.pro，子模块也可以有，因为规则是叠加的，故某个模块的配置都可能影响其它模块。

- ② `proguard-android-optimize.txt`

AGP编译时生成，其中包含了对大多数Android项目都有用的规则，并且启用 `@Keep*` 注解。

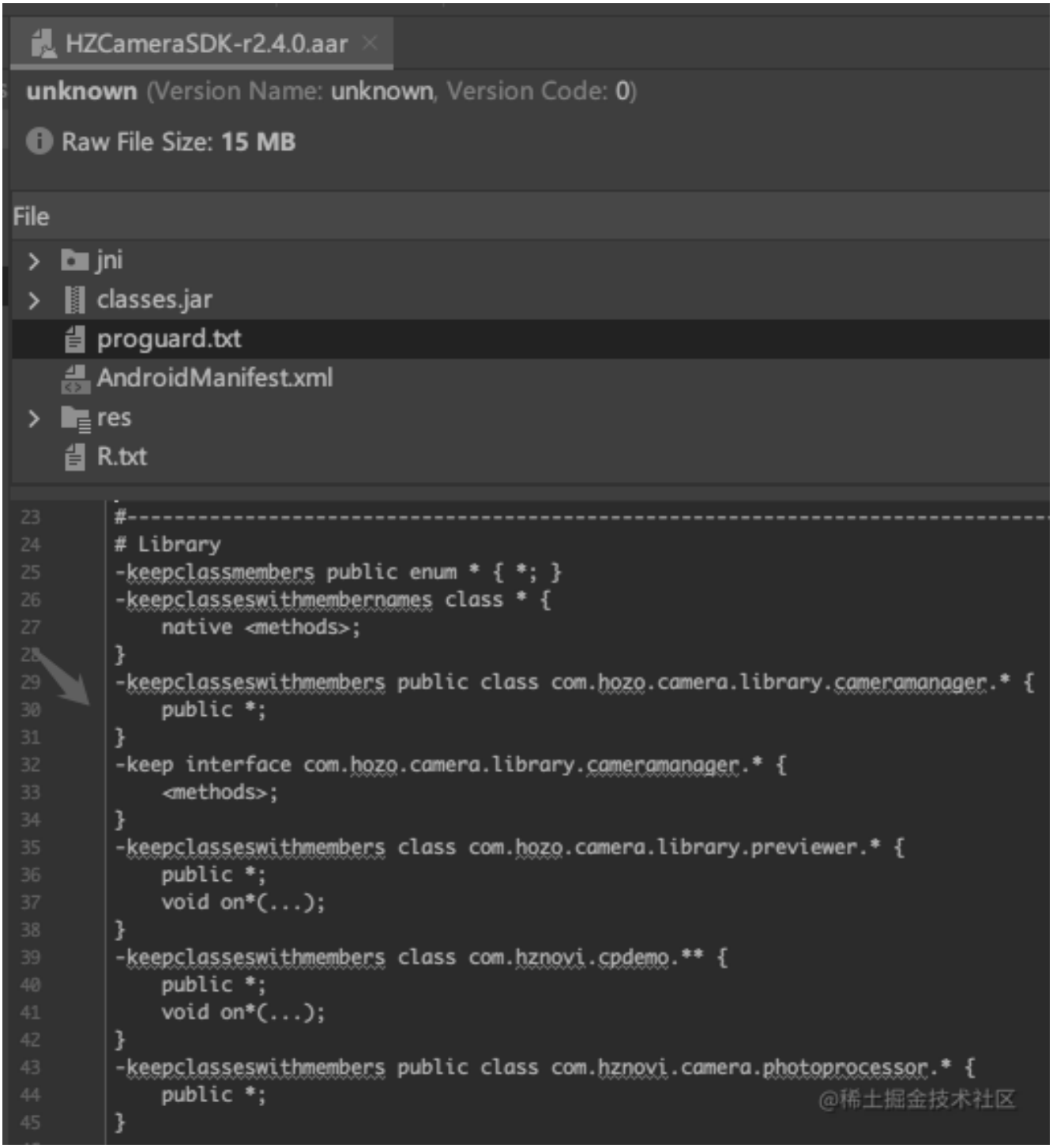
AGP提供的规则文件还有proguard-defaults.txt或proguard-android.txt，可通过 **getDefaultProguardFile** 进行设置，不过建议还是使用这个文件(多了些优化配置)。

- ③ `<module-dir>/build/intermediates/proguard-rules/debug/aapt_rules.txt`

自动生成，AAPT2会根据对应用清单中的类、布局及其他应用资源的引用，生成保留规则，如不混淆每个Activity。

- ④ AAR库 → `<library-dir>/proguard.txt`
- ⑤ Jar库 → `<library-dir>/META-INF/proguard/`

在他们的AAR里，有看到混淆文件：

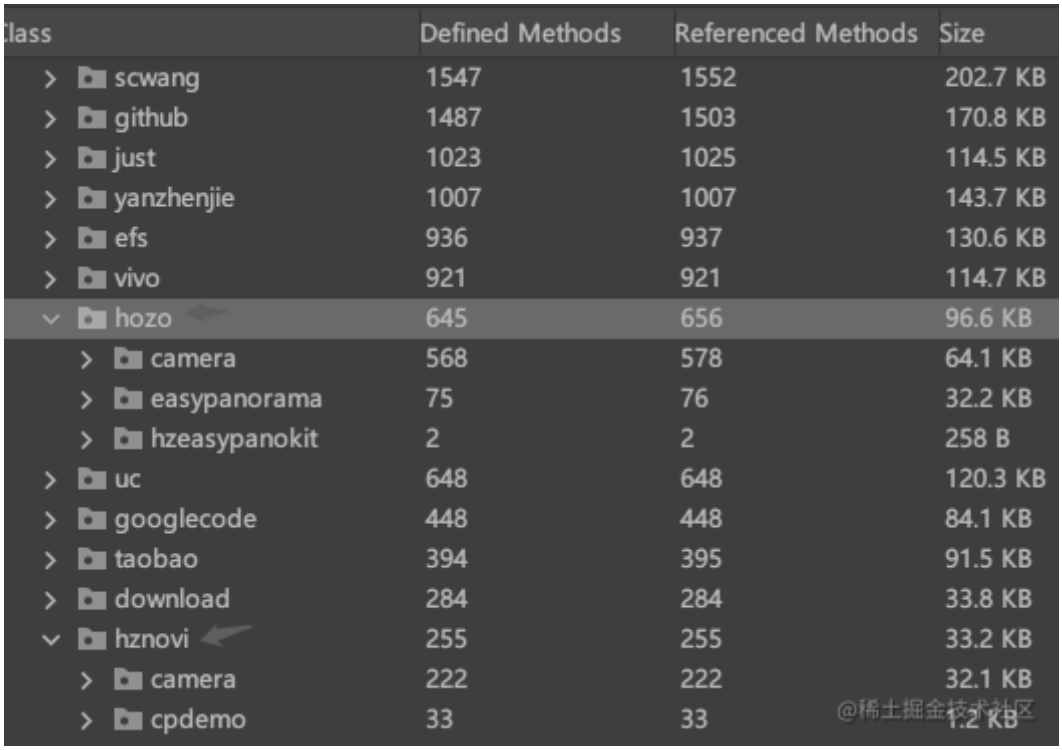


理论上来说，这里写了的话，app模块是不要再写一遍的，2333，但是为了保证一定生效，还是再写一遍吧。但重新打包后一样不行，可能这个 keep规则没有完全覆盖？于是写了匹配范围更广的规则：

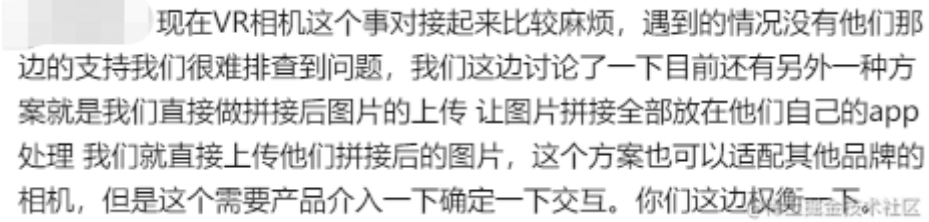
```
-keep class com.hozo.** { *;}
-keep class com.hznovi.** { *; }
-keep interface com.hozo.** { *; }
-keep interface com.hznovi.** { *; }
```

一样不行，不应该啊？难道其他模块的混淆规则影响了？又或者Dex分包？会不会是AAR嵌套的原因？等等。

先看下是不是我们项目问题吧，于是尝试写一个Demo来引用这个库，Copy混淆规则，并打Release包，结果发现Demo能正常使用！！！焯，那就是我们项目有问题了，难道因为不知名原因，导致类丢失？于是我用AS打开两个Release的APK（公司APP和Demo）进行比对：



方法数啥的完全一样，难道真不是混淆的问题？那还可能是什么原因呢？一时有些找不着方向，同事放弃了，领导甚至在群里提出改方案的建议了（公司App只负责上传图片）：

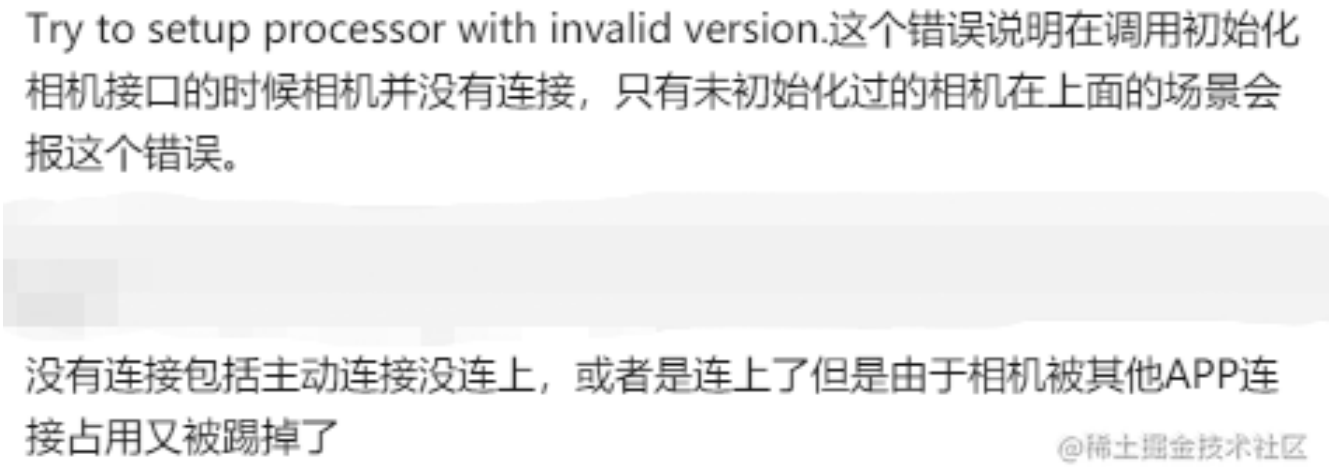

现在VR相机这个事对接起来比较麻烦，遇到的情况没有他们那边的支持我们很难排查到问题，我们这边讨论了一下目前还有另外一种方案就是我们直接做拼接后图片的上传 让图片拼接全部放在他们自己的app处理 我们就直接上传他们拼接后的图片，这个方案也可以适配其他品牌的相机，但是这个需要产品介入一下确定一下交互。你们这边权衡一下。

配置文件缺失/错误排查

而我还在 坚持，回归现象本身：

- 连接相机报错HZJsonResponse: Parse json payload failed。
- 等待很久后回调onNeedInitCamera()，然后跳转相机初始化页面。
- 点击后打印警告日志：Try to setup processor with invalid version，无法正常初始化。

对方的回复：

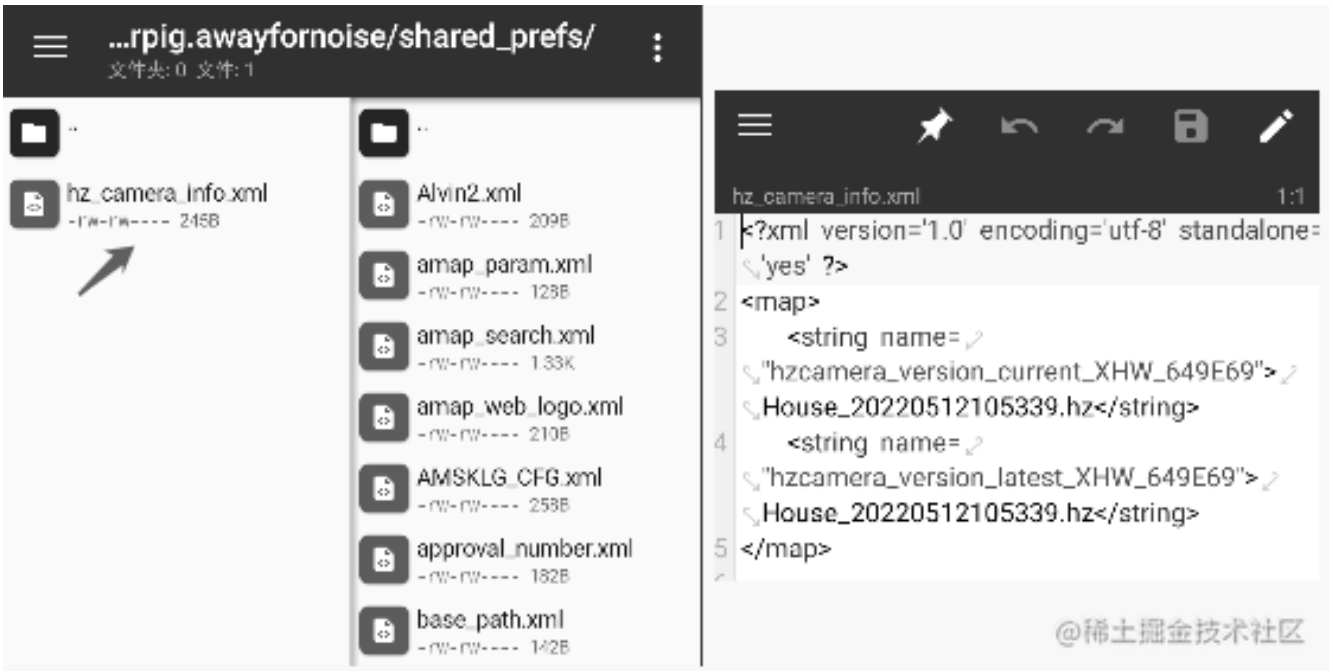


- 简单点说就是：没连上相机，就调用初始化接口
- 但明显和我们的代码逻辑相违背：只有连上了，才会进入初始化页面：

```
override fun onCameraConnected() {
    refreshWifiStatus()
    enterNextActivity()
}
```

```
/**
 *  相机连接发生变化时，需要切换相机，第一次连接相机时，需要进行初始化操作
 * */
private fun enterNextActivity() {
    HZCameraEnv.sharedEnv().switchToCamera(mSSID, object : HZCameraEnv.ISwitchCameraDelegate {

        新的相机连接，需要进行初始化

        override fun onNeedInitCamera(deviceId: String?, isUpdate: Boolean) {
            "ARCamera onNeedInitCamera() -> $deviceId === $isUpdate".logD()
            this@ConnectionActivity.runOnUiThread {
                fly<InitCameraActivity>(isFinish = true)
            }
        }
```

这时，我突发奇想，手机连接相机耗时，会不会是接收相机发过来的某些文件出错导致的？比对下是否有文件缺失不就好了？于是我打开Demo和公司APP的 data/data/包名目录进行比对，发现 shared_prefs 目录下没有这个文件：



copy一下，然后又看了下files目录，看到有个easy_panorama目录，里面明显是相机相关的东西，对比了下公司APP，有部分文件缺失：

同样copy覆盖一下，然后再次运行公司APP连相机，发现报错：没有文件操作权限，直接修改所有者和用户组为我司APP：

再次连相机，可以，不用初始化直接进入了拍摄页面，点击拍照，相机开始拍照，卧槽？难道真的是文件损坏或缺失引起的BUG吗？如果是的话可以往这个方面继续排查了，但事实打脸，拼接图片时卡住，然后提示照片下载失败，依旧是这个异常：

```
HZJsonResponse: Parse json payload failed。
```

又是空欢喜一场，唉，累了，毁灭了，就这样吧，改方案吧，什么垃圾SDK...

Xposed Hook调试AAR

度过了慵懒的周末，又到周一。

一到公司，看到桌面的相机就烦，焯！破玩意，浪费我两个星期时间...

表面上恨不得立马丢了，实际上又看起了源码，嘴上说着不要，身体却很诚实。

尝试定位到报错位置（jd-gui反编译class.jar搜字符串常量）：

```
public class r extends b {
    public JSONObject g;

    public static r e() {
        byte[] var0;
        (var0 = new byte[1])[0] = 0;
        var0 = a.b(a.b(q.a, var0), q.b);
        r var1;
        r var10000 = var1 = new r;
        var10000.<init>(var0);

        try {
            var10000.d();
        } catch (l var2) {
        }

        return var1;
    }

    public r(byte[] var1) { super(var1); }

    public void b(byte[] var1) {
        r var10003 = this;
        String var3 = a.a(var1);

        try {
            var10003.g = new JSONObject(var3);
        } catch (JSONException var2) {
            Log.e( tag: "HZJsonResponse",  msg: "Parse json payload failed.");
            this.g = new JSONObject();
            return;
        }

        var3 = this.g.optString( name: "requestId",  fallback: "");
        String var10001 = this.g.optString( name: "messageName",  fallback: "");
        this.g.optLong( name: "status",  fallback: 0L);
        this.a(var3);
        this.b(var10001);
    }

    void a(byte[] var1) {
    }
}
```


报JSONException异常，才会走这里的代码，而这个异常的引发条件：

Attempts to parse or construct malformed documents Use of null as a name Use of numeric types not available to JSON, such as NaNs or infinities. Lookups using an out of range index or nonexistent name Type mismatches on lookups

中文(简体)

尝试解析或构造格式错误的文档 使用 null 作为名称 使用 JSON 不可用的数字类型，例如 NaN 或无穷大。使用超出范围的索引或不存在的名称进行查找 查找时类型不匹配

大概率是Json的问题，比如格式不正确，而这个方法是将byte数组转换成Json字符串：

```
public static String a(byte[] var0) {
    return var0 != null && var0.length > 0 ? (new String(var0, StandardCharsets.UTF_8)).trim() : null;
}
```


如果可以把入参byte数组搞出来看看就好了，尝试在此下 断点，然后调试附加到 APP进程（手机Root了，并通过Magisk将APP都弄成Debug模式，所以能Debug Release APP）。

但是，并没有走进来，可能不能这样玩？得想下其他法子把参数打印出来。想到好久没玩的Xposed，立马打开专栏：Xposed从入门到入土「插件开发记录 x 框架源码解析」温故知新。到Github找到前几年写的玩具：CPWechatXposed，clone后Build一下。新建一个Hook类：

```
object CameraHook {
    fun hook(lpparam: XC_LoadPackage.LoadPackageParam) {
        val clazz = XposedHelpers.findClass( className: "com.hozo.camera.library.f.r", lpparam.classLoader)
        XposedHelpers.findAndHookMethod(clazz,  methodName: "b", ByteArray::class.java, object : XC_MethodHook() {
            override fun beforeHookedMethod(param: MethodHookParam?) {
                super.beforeHookedMethod(param)
                Log.e( tag: "相机",  msg: "方法调用前：${String(param.args[0] as ByteArray, StandardCharsets.UTF_8)}")
                try {
                    JSONObject(String(param.args[0] as ByteArray, StandardCharsets.UTF_8))
                } catch (e: JSONException) {
                    Log.e( tag: "相机",  msg: "JSON转换异常：${String(param.args[0] as ByteArray, StandardCharsets.UTF_8)}")
                }
            }
            override fun afterHookedMethod(param: MethodHookParam) {
                super.afterHookedMethod(param)
                Log.e( tag: "相机",  msg: "方法调用后~")
            }
        })
    }
}
```


XposedInit.kt加上它：

```
@Throws(Throwable::class)
override fun handleLoadPackage(lpparam: XC_LoadPackage.LoadPackageParam) {
    when (lpparam.packageName) {
        //微信相关
        "com.tencent.mm" -> {
            xsp.reload()
            StepHook.hook() //步数助手
            EmojiGameHook.hook(lpparam) //猜拳和投骰子助手
            RevokeMsgHook.hook(lpparam) //消息防撤回
            RedPacketHook.hook(lpparam) //抢红包
        }
        // 公司APP
        "[redacted]" -> CameraHook.hook(lpparam)
```


运行模块后重启设备，然后连相机，日志断断续续打印出来了：

```
{
    "status":   1023
}
```

连接相机时，会报这个码，错误码么，文档上却没找着，再次发文：

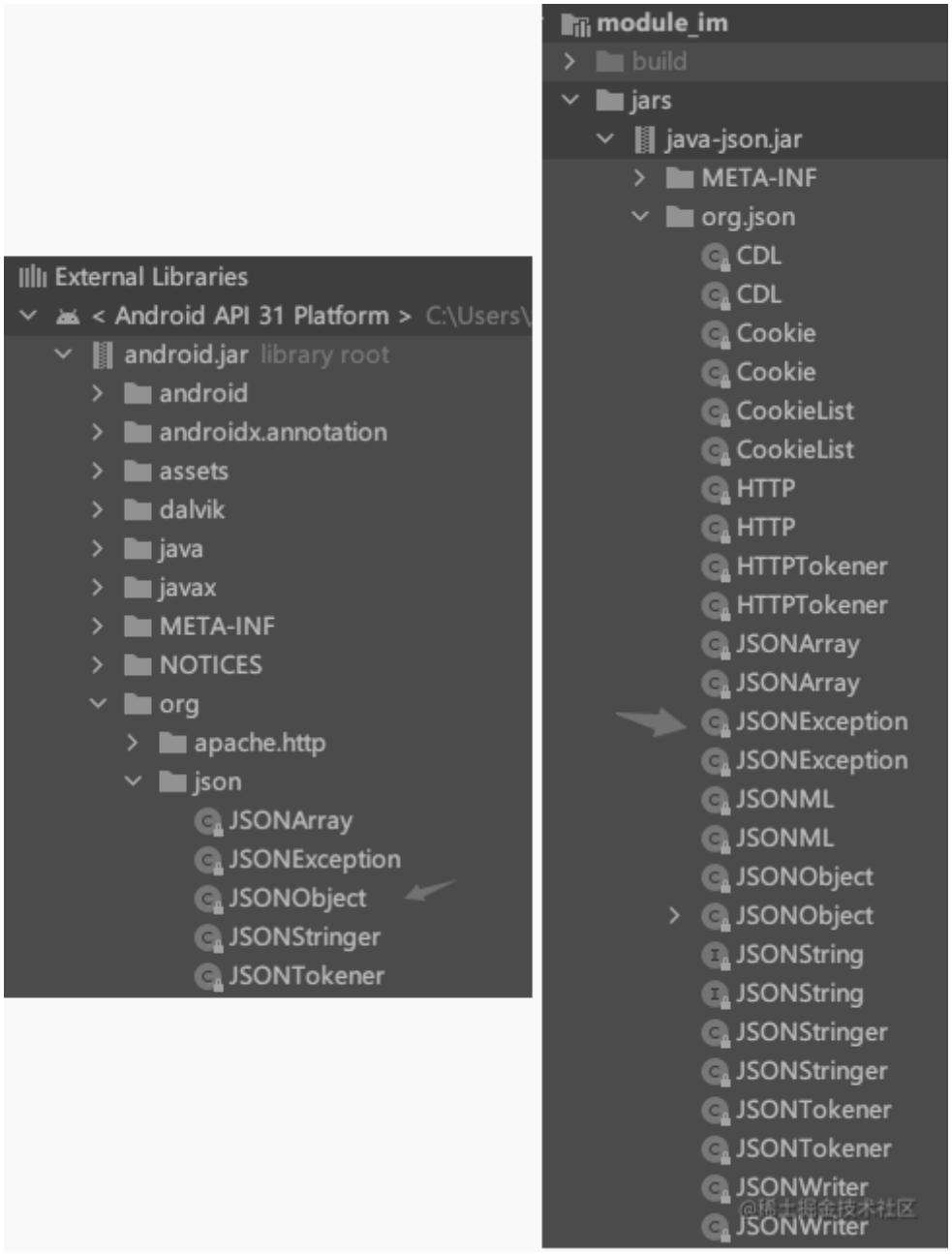
噢，并不是错误码，后续Json数据也陆续打印出来了，复制到Json格式化工具，是正常的Json，这...

不过细心的我发现了，我在Xposed里try-catch这个JsonObject实例化的过程，并不会触发JsonException。但是Parse json playload failed. 还是打印出来了，说明是触发JsonException？？？

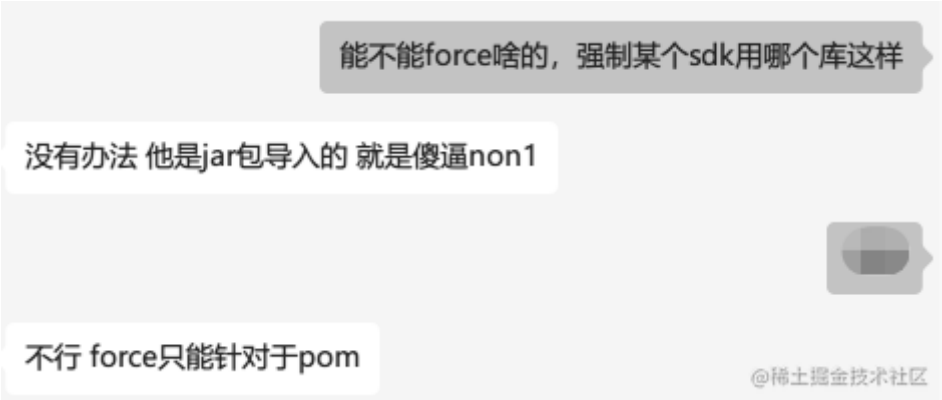
好家伙，难不成你们家的 JsonObject 和 JsonException 是金做的还是银做的？

随手点进 JsonObject 的一定位，直接真相大白：

JsonObject 和 JsonException 指向不同的库！！！



卧槽，这谁埋的雷啊，虽然没定位到具体哪里抛异常，但绝壁是这个库的原因，问了下虾哥能否强制指定，被告知不行：



后面看了一下引入这个库，只是为了调用下xml转json的方法：

```
JSONObject rootJsonObject = XML.toJSONObject(xml);
parse(rootJsonObject);
```


把 implementation files('jars/java-json.jar') 干掉，引入另一个xml转json的库：implementation 'com.github.smart-fun:XmlToJson:1.4.5'，修改调用处相关代码，编译打Release包，安装运行，连接相机，拍照拼接，一气呵成，完美解决。就这样的小BUG磨了我两周，好一个前人挖坑，后人填坑啊...

/ 写在最后 /

事情的大概经过就这样，BUG说解了但又没有完全解，比如没有具体定位到哪里抛JsonException，也有一些让步，比如小米机型还能偶现奔溃问题，但这在我的能力边界外。可惜新相机寄回去了，不然我肯定趁机扩展这方面的姿势~

能用就行，在这个过程中也意识到自己的一些不足：

- 在与人对接描述问题时，并不能简要描述清楚自己的诉求，导致沟通效率并不高；
- 绝大部分时间都花在打包验证上了，每次一打就十来分钟，一直硬钢，没有想办法去提高编译效率；
- 容易钻牛角尖，明明自己认知里知道这样做是没问题的，还要去硬怼，而不是尝试换个方向；
- Native Crash 的排查能力较弱，so库逆向调试也不会，有时间有机会可以了解下~
- 没有一个可以随时快速验证问题的项目，比如上面写Demo验证，我还得去新建项目，折腾kt版本等；
- 与之形成鲜明对比的就是我的Xposed插件玩具，改改就能用，当然如果能加上 注解简化调用就更棒了；

综上，还是自己太Vegetable了，得多多磨练和思考啊~

文中引用到的文章链接如下

补齐Android技能树 - 从害怕到玩转Android代码混淆：

> https://juejin.cn/post/6966526844552085512

Xposed从入门到入土插件开发记录 x 框架源码解析：

> https://juejin.cn/column/6961682067495059469

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

2022年终总结，我的10年Android之旅

深入探究Kotlin的可见性控制，从internal入手

欢迎关注我的公众号

学习技术或投稿

▼

长按上图，识别图中二维码即可关注

阅读原文

2022 全球网络黑产常用攻击方法 Top 10

FreeBuf