**Android 设备的CPU类型(通常称为”ABIs”)**

1. armeabiv-v7a: 第7代及以上的 ARM 处理器。2011年15月以后的生产的大部分Android设备都使用它.
2. arm64-v8a: 第8代、64位ARM处理器，很少设备，三星 Galaxy S6是其中之一。
3. armeabi: 第5代、第6代的ARM处理器，早期的手机用的比较多。
4. x86: 平板、模拟器用得比较多。
5. x86\_64: 64位的平板。

**问题描述**

今天测试人员测试集成版本时除了一个bug：关于华为 Mate 8手机[Android](http://link.zhihu.com/?target=http%3A//lib.csdn.net/base/android" \t "_blank) 6.0系统运行刚刚提测的版本时，出现闪退的bug，而小米 4 手机Android 6.0系统却没有出现任何bug，运行良好。后来查看本人相关模块的代码，发现本人集成版本相关模块的代码和分支版本相关模块的代码是一模一样的，那就是说本人把分支代码合并到主干代码是没有问题的，所以去查看主干代码的问题。

经过一番查看提交日志，发现有位同事再我合并代码之前，提交了一个关于友盟推送的so文件的记录，原来他加入了一个arm64-v8a文件夹，里面有友盟推送的arm64-v8a的so库文件。而其他的so库文本却没有arm64-v8a对应的版本。

**通过百度查到知乎有一段关于arm64-v8a的解释：**

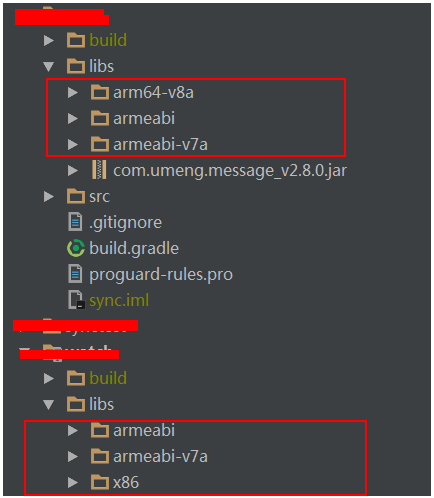
arm64-v8a是可以向下兼容的，但前提是你的项目里面没有arm64-v8a的文件夹，如果你有两个文件夹armeabi和arm64-v8a，两个文件夹，armeabi里面有a.so 和 b.so,arm64-v8a里面只有a.so，那么arm64-v8a的手机在用到b的时候发现有arm64-v8a的文件夹，发现里面没有b.so，就报错了，所以这个时候删掉arm64-v8a文件夹，这个时候手机发现没有适配arm64-v8a，就会直接去找armeabi的so库，所以要么你别加arm64-v8a,要么armeabi里面有的so库，arm64-v8a里面也必须有

作者：green jim   
链接：[微信的安装包在只编译了armeabi，没有x86，arm64-v8a是如何运行在各种处理器的手机上的？ - green jim 的回答](http://www.zhihu.com/question/36893314/answer/78467097)  
来源：知乎   
著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。

发现原来华为 Mate 8手机是64位的[操作系统](http://link.zhihu.com/?target=http%3A//lib.csdn.net/base/operatingsystem" \t "_blank)，而小米 4 手机是32位的操作系统，所以小米 4 手机手机运行APP没bug,而华为 Mate 8手机运行APP出现闪退bug。

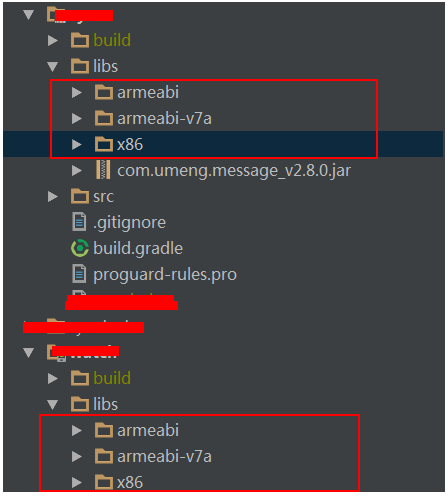
**解决方法**

**1、解决之前的截图：**



从截图可以看出来，第一个项目中有 **arm64-v8a**，而没有**x86**目录，第二个项目中没有**arm64-v8a**，而有**x86**目录。第一个项目是作为项目引用导入到第二个项目中的。

**2、解决后的截图：**



从截图可以看出来，第一个项目中和第二个项目中没有的libs目录下，都是armeabi-v7a、armeabi、x86三个目录，保持一致。第一个项目是作为项目引用导入到第二个项目中的。

**3、解决方法：**

解决方法是：从友盟官方中去下载x86的相关so文件，放在x86目录下，把arm64-v8a目录删除。将所有关于so文件的都要保持一致，即：如果你要添加一个armeabi-v8a目录，下面放第三方的armeabi-v8a相关的so文件，那么你其他的so文件都要有相应想armeabi-v8a版本，不然就会报错。

**4、建议**

来自于博客：[《与 .so 有关的一个长年大坑 》](https://zhuanlan.zhihu.com/p/21359984)给的建议是：

* 为了减小 apk 体积，只保留 armeabi 和 armeabi-v7a 两个文件夹，并保证这两个文件夹中 .so 数量一致
* 对只提供 armeabi 版本的第三方 .so，原样复制一份到 armeabi-v7a 文件夹

下面文章转载于asce1885（简书作者）：[关于Android的.so文件你所需要知道的](http://link.zhihu.com/?target=http%3A//www.jianshu.com/p/cb05698a1968" \t "_blank)  
（原文链接：[关于Android的.so文件你所需要知道的](http://link.zhihu.com/?target=http%3A//www.jianshu.com/p/cb05698a1968)）   
著作权归作者所有，转载请联系作者获得授权，并标注“简书作者”。

早期的Android系统几乎只支持ARMv5的CPU[架构](http://link.zhihu.com/?target=http%3A//lib.csdn.net/base/architecture)，你知道现在它支持多少种吗？7种！

Android系统目前支持以下七种不同的CPU架构：ARMv5，ARMv7 (从2010年起)，x86 (从2011年起)，MIPS (从2012年起)，ARMv8，MIPS64和x86\_64 (从2014年起)，每一种都关联着一个相应的ABI。

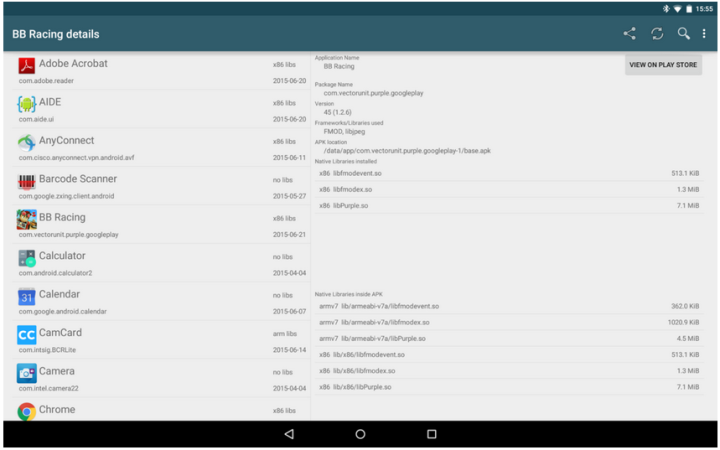
应用程序二进制接口（Application Binary Interface）定义了二进制文件（尤其是.so文件）如何运行在相应的系统平台上，从使用的指令集，内存对齐到可用的系统函数库。在Android系统上，每一个CPU架构对应一个ABI：armeabi，armeabi-v7a，x86，mips，arm64-v8a，mips64，x86\_64。

**为什么你需要重点关注.so文件**

如果项目中使用到了NDK，它将会生成.so文件，因此显然你已经在关注它了。如果只是使用[Java](http://link.zhihu.com/?target=http%3A//lib.csdn.net/base/javaee)语言进行编码，你可能在想不需要关注.so文件了吧，因为Java是跨平台的。但事实上，即使你在项目中只是使用Java语言，很多情况下，你可能并没有意识到项目中依赖的函数库或者引擎库里面已经嵌入了.so文件，并依赖于不同的ABI。

例如，项目中使用RenderScript支持库，[OpenCV](http://link.zhihu.com/?target=http%3A//lib.csdn.net/base/opencv" \t "_blank)，Unity，android-gif-drawable，SQLCipher等，你都已经在生成的APK文件中包含.so文件了，而你需要关注.so文件。

Android应用支持的ABI取决于APK中位于lib/ABI目录中的.so文件，其中ABI可能是上面说过的七种ABI中的一种。



[Native Libs Monitor](http://link.zhihu.com/?target=https%3A//play.google.com/store/apps/details%3Fid%3Dcom.xh.nativelibsmonitor.app)这个应用可以帮助我们理解手机上安装的APK用到了哪些.so文件，以及.so文件来源于哪些函数库或者框架。

当然，我们也可以自己对app反编译来获取这些信息，不过相对麻烦一些。

很多设备都支持多于一种的ABI。例如ARM64和x86设备也可以同时运行armeabi-v7a和armeabi的二进制包。但最好是针对特定平台提供相应平台的二进制包，这种情况下运行时就少了一个模拟层（例如x86设备上模拟arm的虚拟层），从而得到更好的性能（归功于最近的架构更新，例如硬件fpu，更多的寄存器，更好的向量化等）。

我们可以通过Build.SUPPORTED\_ABIS得到根据偏好排序的设备支持的ABI列表。但你不应该从你的应用程序中读取它，因为Android包管理器安装APK时，会自动选择APK包中为对应系统ABI预编译好的.so文件，如果在对应的lib／ABI目录中存在.so文件的话。

**App中可能出错的地方**

处理.so文件时有一条简单却并不知名的重要法则。

你应该尽可能的提供专为每个ABI优化过的.so文件，但要么全部支持，要么都不支持：你不应该混合着使用。你应该为每个ABI目录提供对应的.so文件。

当一个应用安装在设备上，只有该设备支持的CPU架构对应的.so文件会被安装。在x86设备上，libs/x86目录中如果存在.so文件的话，会被安装，如果不存在，则会选择armeabi-v7a中的.so文件，如果也不存在，则选择armeabi目录中的.so文件（因为x86设备也支持armeabi-v7a和armeabi）。

**其他地方也可能出错**

当你引入一个.so文件时，不止影响到CPU架构。我从其他开发者那里可以看到一系列常见的错误，其中最多的是”UnsatisfiedLinkError”，”dlopen: failed”以及其他类型的crash或者低下的性能：

**使用android-21平台版本编译的.so文件运行在android-15的设备上**

使用NDK时，你可能会倾向于使用最新的编译平台，但事实上这是错误的，因为NDK平台不是后向兼容的，而是前向兼容的。推荐使用app的minSdkVersion对应的编译平台。

这也意味着当你引入一个预编译好的.so文件时，你需要检查它被编译所用的平台版本。

**混合使用不同C++运行时编译的.so文件**

.so文件可以依赖于不同的C++运行时，静态编译或者动态加载。混合使用不同版本的C++运行时可能导致很多奇怪的crash，是应该避免的。作为一个经验法则，当只有一个.so文件时，静态编译C++运行时是没问题的，否则当存在多个.so文件时，应该让所有的.so文件都动态链接相同的C++运行时。

这意味着当引入一个新的预编译.so文件，而且项目中还存在其他的.so文件时，我们需要首先确认新引入的.so文件使用的C++运行时是否和已经存在的.so文件一致。

**没有为每个支持的CPU架构提供对应的.so文件**

这一点在前文已经说到了，但你应该真的特别注意它，因为它可能发生在根本没有意识到的情况下。

例如：你的app支持armeabi-v7a和x86架构，然后使用Android Studio新增了一个函数库依赖，这个函数库包含.so文件并支持更多的CPU架构，例如新增android-gif-drawable函数库：

https://pic2.zhimg.com/80/v2-d781628d8bbb95b7004b9f725a892621_hd.png

发布我们的app后，会发现它在某些设备上会发生Crash，例如Galaxy S6，最终可以发现只有64位目录下的.so文件被安装进手机。

解决方案：重新编译我们的.so文件使其支持缺失的ABIs，或者设置

https://pic3.zhimg.com/80/v2-0929b75987b3da4e048995609297a5ee_hd.png

显示指定支持的ABIs。

最后一点：**如果你是一个SDK提供者，但提供的函数库不支持所有的ABIs，那你将会搞砸你的用户，因为他们能支持的ABIs必将只能少于你提供的。**

**将.so文件放在错误的地方**

我们往往很容易对.so文件应该放在或者生成到哪里感到困惑，下面是一个总结：

* Android Studio工程放在jniLibs/ABI目录中（当然也可以通过在build.gradle文件中的设置jniLibs.srcDir属性自己指定）
* Eclipse工程放在libs/ABI目录中（这也是ndk-build命令默认生成.so文件的目录）
* AAR压缩包中位于jni/ABI目录中（.so文件会自动包含到引用AAR压缩包的APK中）
* 最终APK文件中的lib/ABI目录中
* 通过PackageManager安装后，在小于Android 5.0的系统中，.so文件位于app的nativeLibraryPath目录中；在大于等于Android 5.0的系统中，.so文件位于app的nativeLibraryRootDir/CPU\_ARCH目录中。

**只提供armeabi架构的.so文件而忽略其他ABIs的**

所有的x86/x86\_64/armeabi-v7a/arm64-v8a设备都支持armeabi架构的.so文件，因此似乎移除其他ABIs的.so文件是一个减少APK大小的好技巧。但事实上并不是：这不只影响到函数库的性能和兼容性。

x86设备能够很好的运行ARM类型函数库，但并不保证100%不发生crash，特别是对旧设备。64位设备（arm64-v8a, x86\_64, mips64）能够运行32位的函数库，但是以32位模式运行，在64位平台上运行32位版本的ART和Android组件，将丢失专为64位优化过的性能（ART，webview，media等等）。

以减少APK包大小为由是一个错误的借口，因为你也可以选择在应用市场上传指定ABI版本的APK，生成不同ABI版本的APK可以在build.gradle中如下配置：

