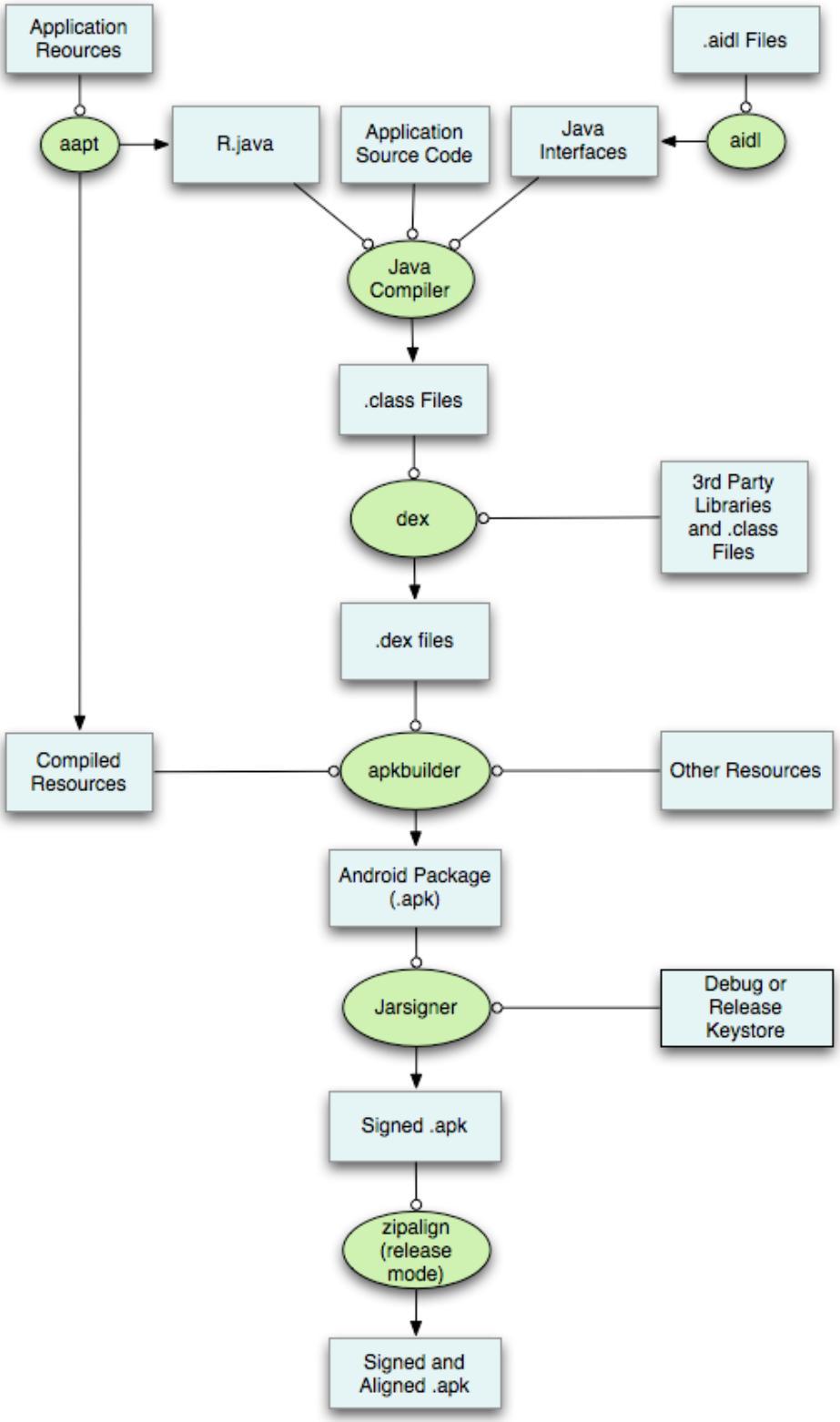
从开始着手公司app安卓原生版本的开发，到如今2.3发布，已经过了快半年的时间，在这半年的时间里，已经逐渐掌握了Android 打包的一些基础知识。今天在这里小小梳理一下，顺便总结下安卓打包中需要注意的问题和一些有效的经验。

打包过程介绍

首先，需要注意的是不管是打什么包，或者用什么工具打包，其背后都是执行的Android提供的构建系统。所以我们先从每个过程介绍一下Android Build系统是如何一步一步将 java文件、资源文件、第三方库等打包成APK的。



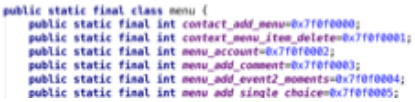
Paste_Image.png

资源文件的预编译

打包过程的第一步，便是资源文件的预编译，资源文件包括所有位于res/目录下的所有文件以及所有的AndroidManifest文件，在这一步中Android使用**aapt**工具将资源文件进行编译，生成R文件。R文件中每个ID的生成都有一个特殊的规则，每个id都是一个四字节的数字 0x PPTTNNNN, 第一个字节PP代表了资源文件代表的包名，在Android应用程序中PP始终是7f。第二个字节TT代表了资源文

件的类型，aapt从0开始每发现一种新的资源类型就加1，最后面两个字节NNNN代表了资源文件在该类型中的顺序索引，跟TT类似，从0开始每发现一个新的资源文件名就加1。比如我们看一下我们工程中生成的R文件

id资源



menu资源

我们可以使用appt命令输出打包好的apk中所有的资源文件列表

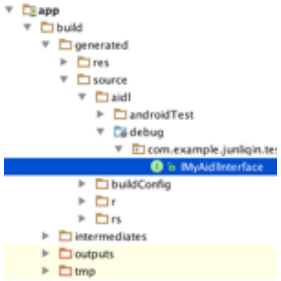
appt dump resources

AIDL文件编译

AIDL 全称Android Interface definition language。在Android如果涉及到多进程比如远程service的时候就需要去定义aidl文件，用来统一描述进行间通信的接口。

```
interface IMyAidlInterface {        /**     * Demonstrates some basic types that you can use as parameters * and return values in AIDL.     */        void basicTypes(int anInt, long aLong, boolean aBoolean, float aFloat,             double aDouble, String aString);}
```

Android打包时会将aidl文件转化为一个同名的java接口，如下图所以，可以在/build/generated/source/aidl下看到。



AIDL打包

Java文件的编译

这一步是标准的java文件编译的过程，编译的输入既包括我们手动编写的代码也包括前两步中生成java文件。编译的输出为.class文件。

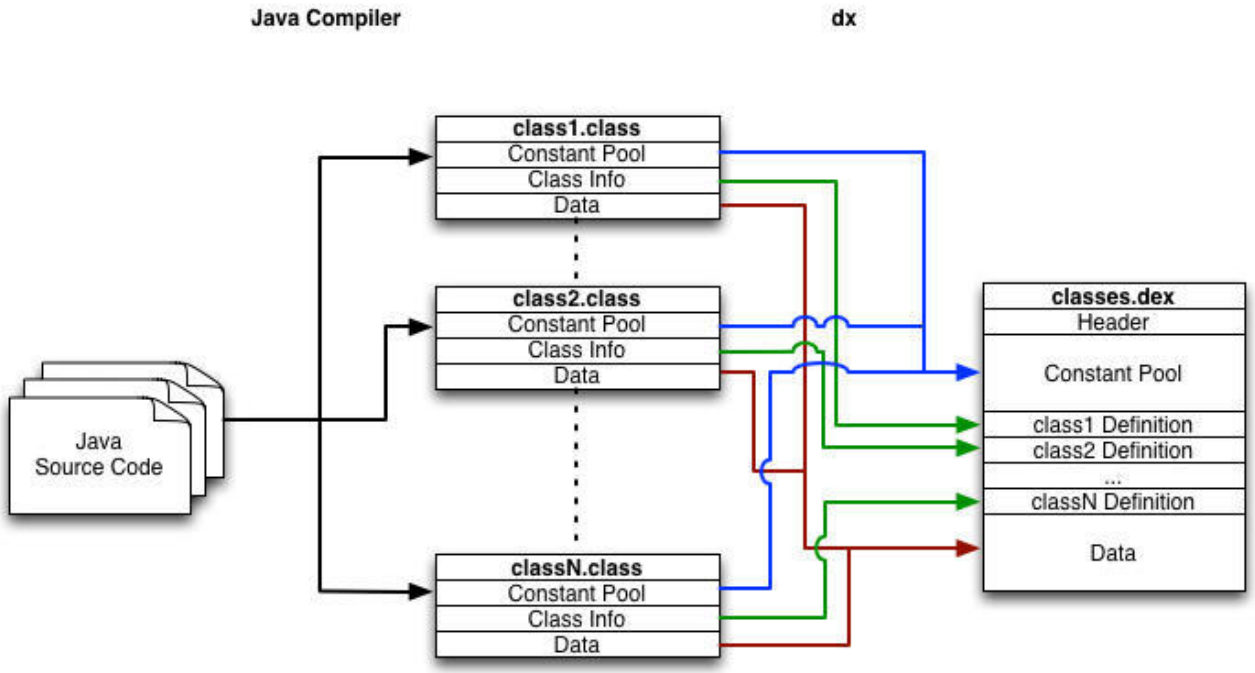
生成.dex文件

第四步将上一步生成的class文件，以及引用的第三方库中的class文件一起转化为Dalvik字节码。虽然Android使用Java语言编程。但是Android并不使用标准的Java虚拟机进行java code。Android有自己的虚拟机 Dalvik以及5.0以上的ART。 可以大致了解一下Dalvik虚拟机和传统虚拟机的区别：

- Dalvik虚拟机占用内存更少

- JVM是基于Stack的(Stack based)，DVM是基于Register的(Register based). 跟进一步说，JVM将局部变量存在stack中，而DVM将变量保存在register中。因此标准java虚拟机需要更多的指令集，而DVM的指令集更少，另一方面DVM需要用寄存器编码指令的source 和 destination ，因此Dalvik的一条指令更长。
- 在JVM运行时，会根据需要去load每个class文件。而Dalvik中所有的class都在一个dex文件中。

基于第三条区别，我们在前第三步中生成的所有class文件以及第三库中引用的class文件都需要一起整合为一个dex文件。



JVM_vs_DVM.jpg

不过这里有一个潜在的危险，由于dex设计上的原因，单个dex内最多能引用的方法数上限为65536，这个数字对于当今很多APP来说都已经不够用了，不行的是，我们项目中就碰到了。不过Google也早早就放出了解决方案。这里就不详细讨论了，有兴趣了解可以戳这里。

资源文件打包

第一步资源文件已经使用aapt进行了预编译，是为了 让所有引用了资源的java文件正常编译，这里在打包之前，Android build system需要将所有的资源文件进行打包。这里依然是使用第一步中用到的aapt工具。打包后会生成一个**resources-debug.ap**_文件，这个文件中包含了图片,layout,menu,anim, AndroidManifest，其中图片都经过了优化，xml文件被编译成二进制格式。string被编译成单独的**resources.arsc**文件。

打包APK

第五步就是打包APK了，这一步的工作就是将dex文件，资源文件（编译的和未编译的）打包在一块，生成apk。

签名

第五步生成的apk文件，必须经过签名才能安装在设备上。签名前，我们必须生成自己的keystore

```
$ keytool -genkey -v -keystore my-release-key.keystore-alias alias_name -keyalg RSA -
keysize 2048 -validity 10000
```

使用keytool，生成一个keystore，这个keystore包含一个有效期10000天的key。

要对apk进行签名，需要用到java提供的签名工具

```
$ jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1-keystore my-release-
key.keystore my_application.apk alias_name
```

如果需要验证某个apk是否签名过，可以使用

```
jarsigner -verify -verbose -certs my_application.apk
```

对齐

打包apk的最后一步是一个对齐工作。这一步的工具简而言之就是使apk内所有未压缩的所有文件相对于apk的其实位置都是4字节对齐的。这样，这些文件既可以直接使用**mmap**映射访问。这样做的目的是为了减少app在运行时所占用的内存。

```
zipalign -c -v 4 application.apk
```