# ****1 背景****

本文的核心就是下图：

IMG_256

关于Gradle的Android插件本文不会过多的说明，只给一个抛砖引玉的提示，详细使用参见文档API及Gradle配置，其实个性化的构建配置一般都是Gradle与Groovy的编写，与Android插件没太多关系，所以重点还在Groovy与Gradle构建。

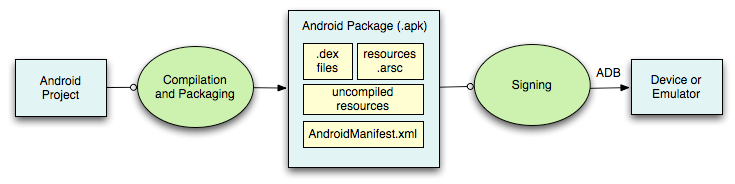
不过在构建之前还需要先了解一些构建的流程和相关IDE及编译特性，具体下面会说到。

# ****2 Android应用编译框架****

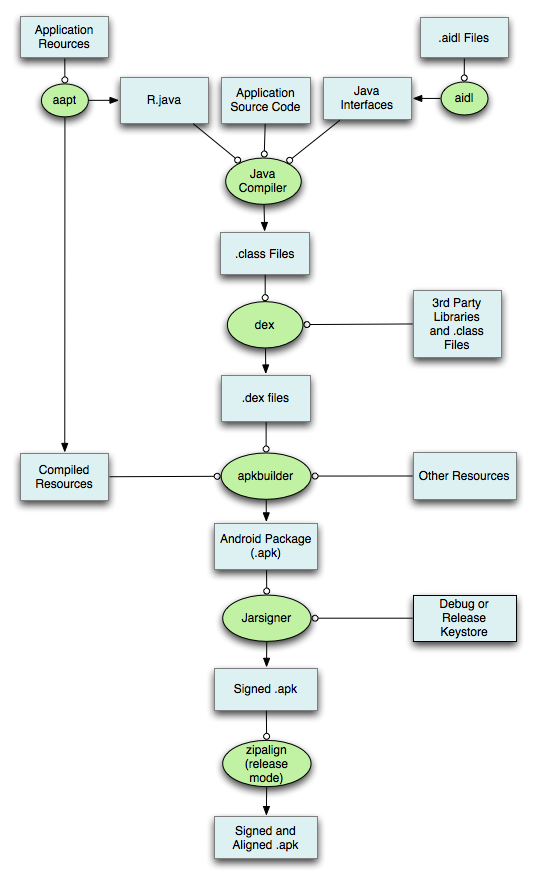
在我们开发App时大多数时候的编译流程都是直接通过IDE的按钮或者命令行一步到位apk的生成，直观上的感觉好像源码到Apk的生成只有一步，实质不是这样的。从我们点击按钮那一刻到生成apk的过程实际是非常复杂的，涉及到很多步骤，下面我们来粗略看看。

## ****2-1 Android应用源码到生成Apk流程****

Android应用的编译其实就是打包为.apk文件，这个被打包的zpk文件实质其实是一个压缩包（譬如你可以重命名后缀解压），这个压缩包至少包含编译为.class文件转换的.dex文件、一个二进制的AndroidManifest.xml文件、编译的资源文件resources.arsc、未编译的资源文件等，然后在运行前再对这个压缩包进行签名操作即可。具体流程如下图：



Android构建系统是用来构建、测试、运行和封装应用程序的一个工具箱，整个构建系统我们可以通过已经集成OK的AS或者命令行来使用，构建过程会涉及许多工具和流程，通过这些工具和流程生成了许多中间文件，然后最终生成了APK。下图描述了构建过程中涉及的不同工具和过程：



构建系统将依据配置的product flavors、build types和dependencies合并所有的资源，如果此时不同文件夹含有相同名称或设置的资源则会按照如下规则进行覆盖：dependencies覆盖build types，build types覆盖product flavors，product flavors覆盖main资源文件夹。下面是对上图典型构建过程的详细描述：

1. Android资源打包工具（AAPT）将应用的资源文件（譬如AndroidManifest.xml文件和Activity相关的XML文件等）进行编译，生成的一个R.java文件就是关联我们Java代码与资源文件的基础。
2. aidl工具将应用中所有.aidl的文件转换为Java接口。
3. Java编译器将应用中所有Java代码（包括R.java和aidl接口）编译输出为class类文件。
4. dex工具将应用编译输出的类文件（包括第三方jar包）转换为Dalvik字节码，.dex文件就可以最终被打包如.apk文件。
5. apkbuilder工具将所有非编译资源（譬如images）、编译资源和DEX文件打包成一个APK文件。
6. 一旦.apk文件生成以后就必须进行签名（debug或者release签名）才能在设备上运行。
7. 通过zipalign工具对齐APK，减少APP在设备上运行时的内存占用。

整个构建过程生成的东东默认都在app/build目录下（包括中间产物和最终产物）。

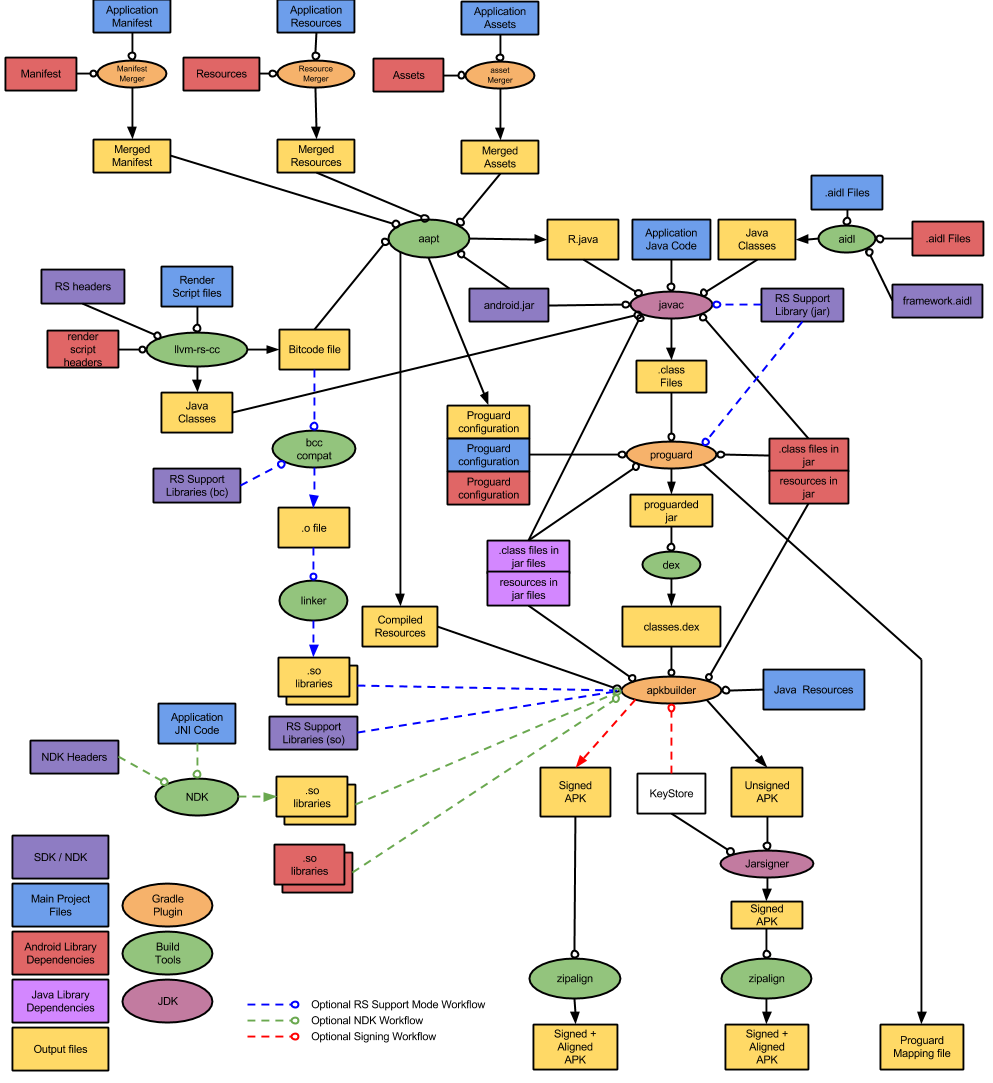
****特别注意：**** 应用程序构建有一个64K方法限制说法，超过64K会抛出如下构建信息：

Java

|  |  |
| --- | --- |
| 1 | Unable **to** execute dex: method ID **not** **in** [0, 0xffff]: 65536. |

如果撞上该问题（一般是超大型项目或者疯狂使用开源框架等就会越界）请[点我参考解决](http://developer.android.com/intl/zh-cn/tools/building/multidex.html)。

下面这幅图就是整个Android应用（不包含NDK部分）的构建编译框架详细流程说明（来源于官方）：



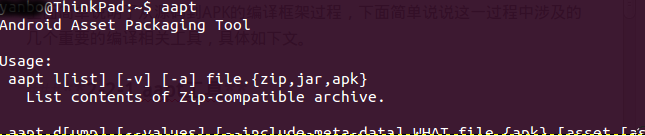
通过这幅图可以明显看出来Android应用程序编译源码到最终的apk产物过程实际是十分复杂的，我们平时只是把这些细则交给了IDE和构建框架而已，所以察觉不到这些细则过程。

## ****2-2 Android应用编译流程相关主要工具****

上面简单说明了从源码到APK的编译框架过程，下面简单说说这一过程中涉及的几个重要的编译相关工具，具体如下文。

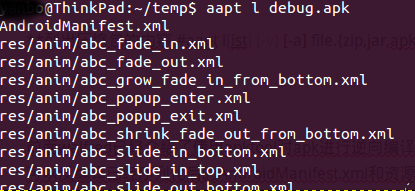
### ****2-2-1 aapt工具****

aapt（Android Asset Packaging Tool）是Android资源打包工具，在SDK的tools目录下，我们可以使用该工具查看、创建、更新ZIP格式的文件（zip、jar、apk等)，也可以将资源文件编译成二进制文件。平时我们很少直接使用这个工具，通常都向前面编译流程图介绍的一样，编译框架会自动帮助我们调运这个工具。下面是aapt命令配置OK后运行的情况：

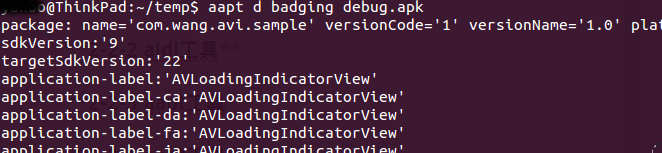


关于aapt命令工具的用法我们直接在终端输入aapt回车即可看见相关help文档，这里只给出几个我们比较常用的命令，其他的参见help文档。

****列出压缩包的内容 #aapt l[ist] [-v] [-a] file.{zip,jar,apk}****



****查看APK包中指定内容 #aapt d[ump] file.{zip,jar,apk}****



这个比较实用，有时候想快速知道一个app的包名等信息时可以通过该命令快捷获取。

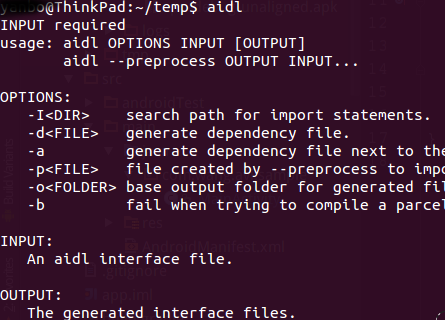
****aapt其他操作命令参数****

aapt命令还可以进行打包生成资源压缩包、给压缩包删除或者添加指定文件等操作，这也是编译构建系统中生成R.java等流程使用的核心工具命令。感兴趣的可以自行google通过aapt命令行去手动打包资源文件。

### ****2-2-2 aidl工具****

AIDL（Android Interface Definition Language）是一种IDL语言，用来生成可以在Android设备上两个进程间通信（IPC）的代码。如果在一个进程中要调用另一个进程的对象则可以使用AIDL生成可序列化的参数进行传递。我们通常开发中都会依据AIDL规则编写相应的aidl文件，然而我们会发现在build目录下会出现我们aidl对应的java接口文件，这就是因为编译框架通过了aidl编译工具将.aidl文件转换为.java接口文件。

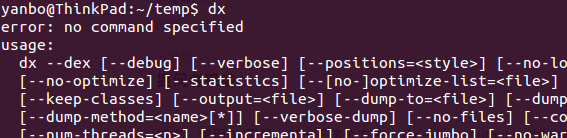
下面就是在SDK的tools目录下将aidl文件转为java文件的aidl工具：



关于aidl工具的用法很简单，帮助文档说的很明白了，详细感兴趣的可以自己去尝试一下，这里不再详细说明；基本不怎么手动去用，我反正没仔细研究过。

### ****2-2-3 dx工具****

我们通过aapt将资源打包同时生成R.java，通过aidl将aidl文件编译生成对应的java文件，再通过Java编译器将R.java、Java源码文件、aidl的Java文件编译为相关的class文件。这时候对于纯Java来说class字节码就是JVM虚拟机可执行的文件了，然而对于Android的Dalvik和ART虚拟机来说class文件是无法直接执行的，他们可执行的文件其实是dex格式的文件，所以我们需要通过SDK/tools目录的dx工具将class字节码编译为dex执行文件。如下是dx工具命令基本情况：



说到了dx工具就不得不再强调几个比较重要的概念和基本原理，具体如下详情。

**apk、dex、odex文件关系：**

dex是Dalvik VM Executes（Android Dalvik执行程序，现在ART也继续兼容使用此格式执行程序）的全称，dex格式文件通常在Android虚拟机中执行时都会先进行优化 ，优化后的文件大小一般都会比原dex文件大。这种优化的发生时机会分两种情况：

* 系统预置应用：在系统编译后直接生成以odex为后缀的优化文件，启动app时不用再解包取dex，效率高，所以在发布系统预置应用时可以看见一个不包含dex文件的apk文件和一个相应的odex文件。
* 非预置应用：编译直接生成包含dex文件的apk，包含在apk文件里的dex文件会在运行时被优化，优化后的文件将被保存在缓存中。

这也就是为何系统编译预置app源码后会看到odex和不含dex的apk文件，而独立app编译后只看见内含dex的apk文件的原因。之前在上家公司做盒子开发时有人曾经有过这个疑惑，当然我也有过！

**dex文件65535异常方法限制原因：**

Android的Dalvik和ART运行时环境都能够执行dx工具生成的.dex文件，也就是说Dalvik和ART使用了同一套Dalvik指令集。通过[相关资料查询](http://source.android.com/devices/tech/dalvik/instruction-formats.html)可以知道Dalvik指令集使用了16位寄存器来保存项目中所有的方法引用，2的16次方是65536，也就是说一个dex文件最多只能引用65536个方法，所以对于Dalvik和ART运行时环境来说都有这个局限性。我勒个去！！！这不就是我们有时候编译项目时抛出Android Dex方法限制异常的原因么（上面也有介绍，不明白上翻回看），也就是说编译时抛出这个异常是因为项目包含的方法太多导致的，好在Google官方也意识到了这个缺陷，所以他们给出了解决方案，如下：

* 使用ProGuard清除项目中无用方法，使用相关脚本对项目中没用到的第三方库中的方法进行清除处理；
* 由于Dalvik运行时环境限制一个apk只能包含一个classes.dex文件，所以我们可以使用Multidex Support Library支持包让一个apk里支持多个.dex文件，这样就可以突破65536的限制。

dx过程中这个错误非常经典，一般都出现在大量使用了第三方库的情况下，所以需要注意一下。

### ****2-2-4 apkbuilder工具****

关于apkbuilder工具这个叫法其实已经有些过时了，因为比较新版本的SDK中已经将apkbuilder工具去掉了，不过apkbuilder工具的实质其实是对/android-sdk-linux/tools/lib/sdklib.jar中ApkBuilderMain等的一个封装而已，所以即使没有了该工具我们也可以自己实现封装，不过新的编译框架会自动帮助我们解决这一过程，我们无需手动处理。该过程的实质其实和压缩工具的性质差不多，只是它将相关资源、dex文件等打包压缩成了一个指定压缩方式和深度等的apk文件而已。

### ****2-2-5 keytool与jarsigner工具****

对apkbuilder打包压缩出来的apk进行签名的实质其实是在应用程序的特定字段写入特定的标记信息，以便用来表示该软件已经通过了签署者审核。签名的作用主要是识别应用的作者、检测应用程序是否已经改变、检测是否为同一个应用等。

一般我们可以通过keytool工具生成签名私钥，然后通过jarsigner工具使用私钥对应用进行签名。不过这一过程非常简单，这里就不再啰嗦了，自行脑补。

### ****2-2-6 zipalign工具****

zipalign工具可以对打包的应用进行优化，优化过的应用在运行时执行效率可以达到最大限度且会占用更少的RAM（Random Access Memory）内存。zipalign对apk文件中数据进行4字节对齐，也就是说编译器把4个字节作为一个单位来进行操作，这样CPU就能对代码进行高效访问，因为对齐后Android系统可以通过调用mmap函数读取文件，也就是说进程可以像读写内存一样操作我们apk中普通文件，所以当对齐的应用在系统中执行时通过共享内存IPC读取资源就能得到较高的性能，如果没有对齐处理则必须显示的调运read等方法去操作数据，也就是说运行过程会比较缓慢且会花费更多的内存，从而导致性能下降。

关于zipalign工具的使用这里也不再啰嗦了，因为通常编译框架允许我们直接配置脚本而不用手动执行命令。

### ****2-2-7 ProGuard工具****

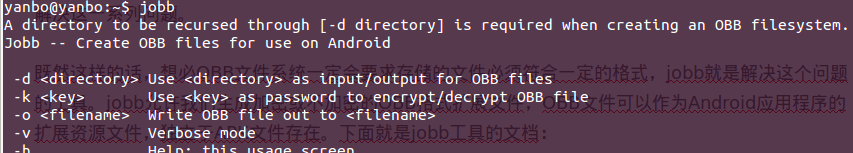
ProGuard是一个压缩、优化和混淆Java字节码class文件的工具，它可以删除无用的类、字段、方法、属性及没用的注释等，最大限度地优化class字节码文件。它还可以使用简短的无意义名称来重命名已经存在的类、字段、方法和属性。我们通常用它来混淆最终的项目，然后稍微增加项目被反编译的难度，当然了，对于现在的技术来说反编译难度这个已经不是问题了，我们还是重点关注他的优化无用资源和简洁替代吧。

关于ProGuard工具的使用这里也不再啰嗦了，因为通常编译框架允许我们直接配置脚本而不用手动执行命令。

### ****2-2-8 jobb工具****

其实这个工具不属于正常编译框架的流程，算是Android的一个拓展特性而已。从Android 2.3版本开始系统增加了一个OBB文件系统（权限访问限制隔离文件系统）和StorageManager类用来管理外部存储上的数据安全。

如果你之前在Android手机上安装过《纪念碑谷》或者《机械迷城》游戏，那你就能对这里讲的jobb工具和OBB文件系统有一个很好的理解。还记不记得在安装几十兆大小的游戏后你还需要下载一个两百多兆的zip压缩包放到文件系统的Android/obb/[GamePackageName]目录下才能正常玩游戏。之所以这么做是因为我们的游戏工程中包含大量的资源（图片、视频、音乐等），直接编译为APK可能会高达好几百兆，系统在安装APK时又会对APK文件大小有一个限制，这么大的APK文件必定会导致Android系统无法正常安装该APK；相信此时机智的你指定会说，我们把这些资源直接放到SD卡上不就完了？哈哈，你想没想过一问题，如果直接放到SD卡，系统的音乐、视频、图片等管理器岂不是直接可以索引到这些东东了，那得多不好（插一句，还可以将这些资源去掉后缀保存，这样这些媒体库就无法索引了，譬如Android系统邮件应用的附件就是这么设计的，真机智！）。好在Android 2.3的OBB文件可以很牛叉的解决这一系列问题。

既然这样的话，想必OBB文件系统一定会要求存储的文件必须符合一定的格式，jobb就是解决这个问题的工具。jobb允许我们生成加密或不加密的OBB格式扩展文件，OBB文件可以作为Android应用程序的扩展资源文件，独立于APK文件存在。下面就是jobb工具的文档：

关于jobb工具这里就不深入说明了，一般游戏等大资源应用开发中才可能会考虑到这种设计，用到时再脑补也不迟，这里知道有这么回事就行了。