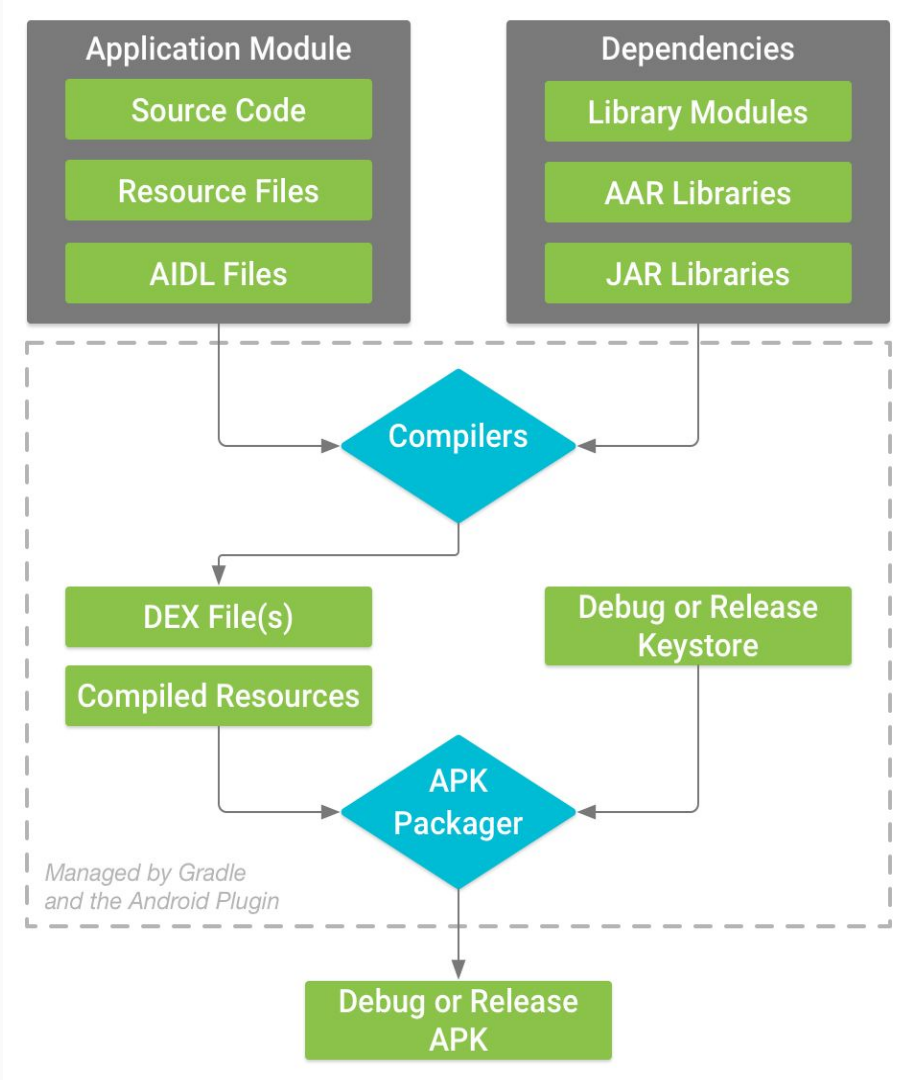
手动打包 apk

生命不息，折腾不止

为了更好的理解 Android 的打包流程，我们决定使用 Android 提供的工具链手动实现打包过程

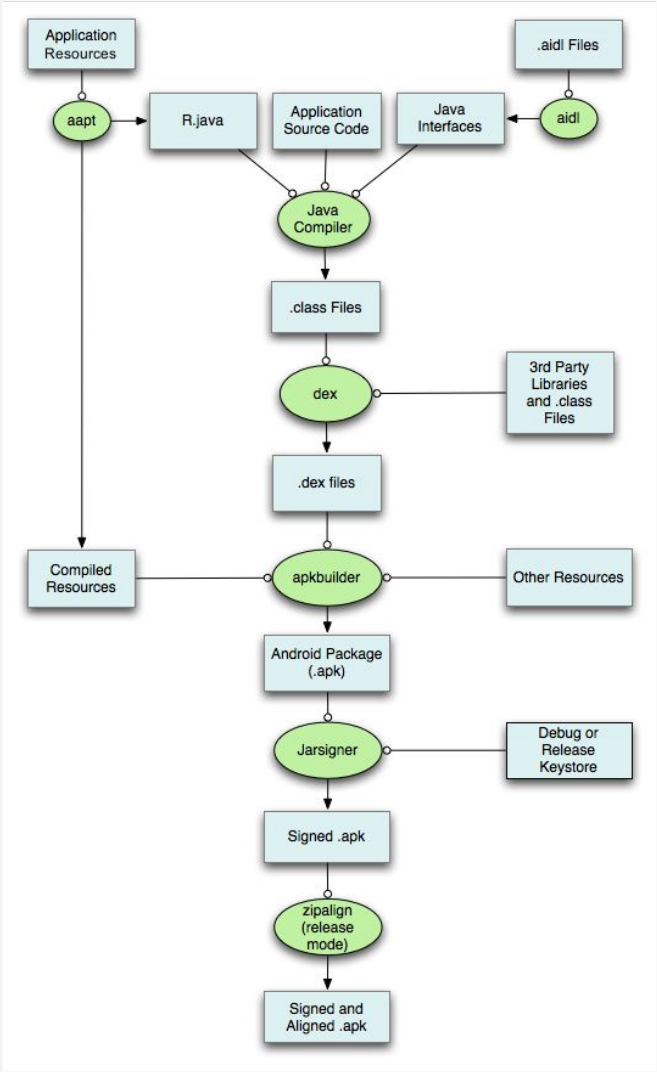
打包流程

一个简单的打包流程如右图



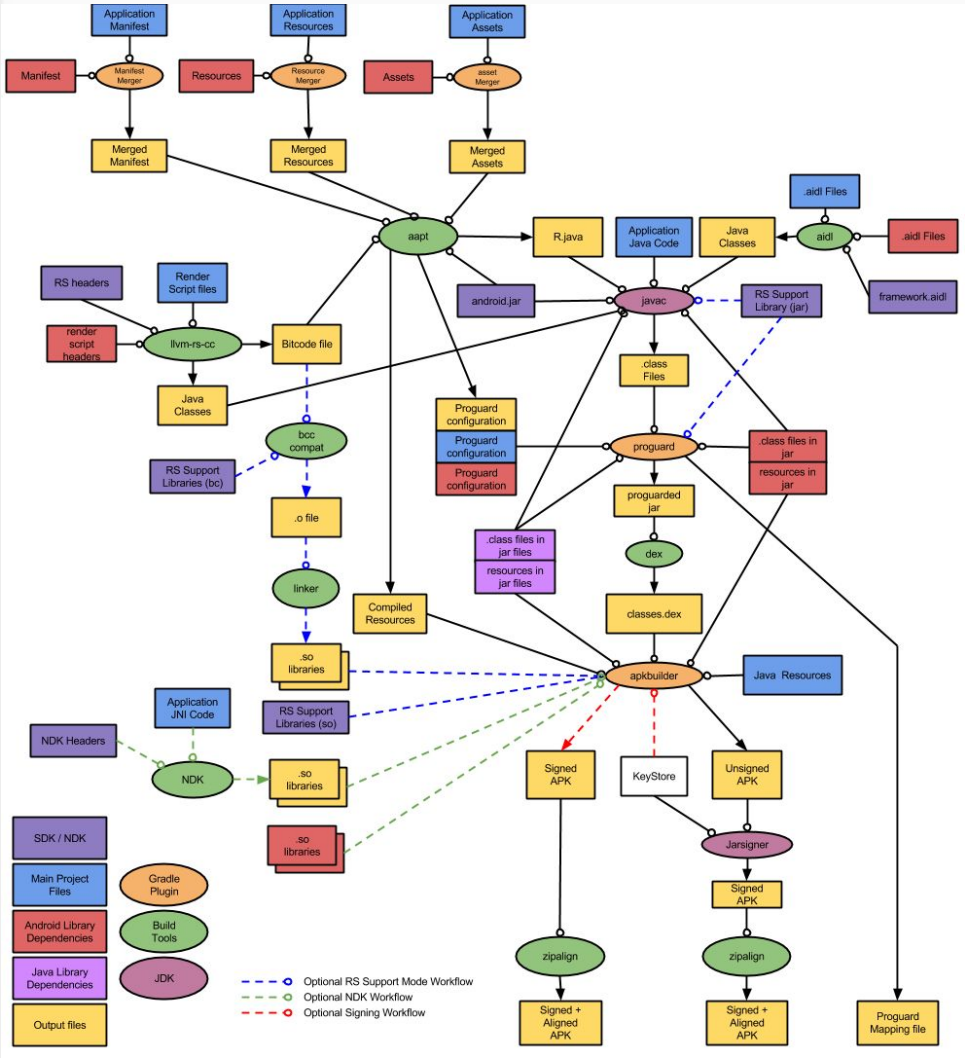
打包流程

稍微详细点的打包流程如右图



打包流程

再详细点的打包流程如右图



打包工具链

aapt2

javac

d8

apkbuilder

zipalign

apksigner

aapt2

https://developer.android.com/studio/command-line/aapt2

从 gradle plugin 3.0 开始默认使用 aapt2 作为资源编译工具

aapt2 相较于 aapt 的优势在于：aapt2 分为 compile 和 link 两个过程，未修改的资源可以跳过 compile，从而提高编译速度

aapt2 compile 参数

-o：指定编译文件输出路径，可以是文件夹也可以是文件（zip），当输入是文件时这里是文件夹，当输入是文件夹时这里是文件

--dir：指定输入资源文件夹，指定 dir 后就不用一个一个输入了

–pseudo-localize：生成伪本地字符串，以用来测试国际化时显示问题
https://developer.android.com/guide/topics/resources/pseudolocales

–no-crunch：禁用 PNG 压缩处理

–legacy：将 aapt 中认为是 warnings 的作为错误抛出，并终止编译，可以认为是严格模式

-v：开启日志输出

aapt2 compile 例子

```
def aapt2_compile():
    zipRes = "{dir}/compile.zip".format(dir=tempDir)
    subprocess.call(["aapt2", "compile", "--pseudo-localize", "-o", zipRes, "--dir", resInputDir])
    zipfile.ZipFile(zipRes, 'r').extractall(resOutputDir)
```

aapt2 link 参数

-o：指定编译文件输出路径，文件类型为 apk，但这个 apk 不能安装，只是资源 apk

--manifest：指定 manifest 文件

-I：指定 android.jar 文件

-A：指定 assets 文件夹

--java：指定生成 R 文件的存放路径

--target-sdk-version：指定 apk target sdk version

Aapt2 link 参数比较多，这里只列出本次打包使用到的参数

aapt2 link 例子

```
def aapt2_link():
    seperator = " {resOutputDir}/".format(resOutputDir=resOutputDir)
    filesStr = resOutputDir + "/" + seperator.join(os.listdir(resOutputDir))
    cmd = "aapt2 link -o {apkOutputDir}/res.apk -I {androidJarFile} --manifest {manifestFile} --java {rOutputDir} --target-sdk-version 26 {filesStr}".format(
        apkOutputDir=apkOutputDir, androidJarFile=androidJarFile, manifestFile=manifestFile, rOutputDir=rOutputDir, filesStr=filesStr)
    subprocess.call(cmd, shell=True)
```

javac 参数

-bootclasspath：指定引导类文件路径，默认使用 jdk，这里使用 android.jar

-d：指定编译文件输出路径

-classpath：设置源码编译依赖类路径，默认是 CLASSPATH 环境变量指定路径

-sourcepath：设置待编译源码路径（不设置时可以使用绝对路径）

关于 classpath 和 sourcepath 区别可参考：https://blog.csdn.net/CaesarZou/article/details/5462189

javac 例子

```
def javac():
    rFile = rOutputDir + "/com/michael/test/R.java"
    javaFile = javaInputDir + "/com/michael/test/MainActivity.java"
    cmd = "javac -bootclasspath {androidJarFile} -d {classesOutputDir} {rFile} {javaFile}".format(
        androidJarFile=androidJarFile, classesOutputDir=classesOutputDir, rFile=rFile, javaFile=javaFile)
    subprocess.call(cmd, shell=True)
```

d8 参数

--lib：指定 android.jar 路径

--output：指定编译文件输出路径

文档：https://developer.android.com/studio/command-line/d8

d8 例子

```
def d8():
    cmd = "d8 {classesOutputDir}/com/michael/test/*.class --lib {androidJarFile} --output {dexOutputDir}".format(
        classesOutputDir=classesOutputDir, androidJarFile=androidJarFile, dexOutputDir=dexOutputDir)
    subprocess.call(cmd, shell=True)
```

apkbuilder 参数

在 Android sdk build tools 22 时做了一些改动，将 apkbuilder 移除了，通过查看老版本 apkbuilder 发现该脚本实际上执行的是 sdklib 中的 ApkBuilder 类，所以将老版本的拷贝一份就可以了

-v：输出日志

-u：创建一个未签名的 apk

-z：指定编译好的资源包，即前边生成的 res.apk

-f：指定 dex 文件路径

ApkBuilder.java：https://github.com/pixnet/android-platform-tools-base/blob/master/sdklib/src/main/java/com/android/sdklib/build/ApkBuilder.java

apkbuilder 例子

```
def apkbuilder():
    cmd = "apkbuilder {apkOutputDir}/unsign_unalign_app.apk -v -u -z {apkOutputDir}/res.apk -f {dexOutputDir}/classes.dex".format(
        apkOutputDir=apkOutputDir, dexOutputDir=dexOutputDir)
    subprocess.call(cmd, shell=True)
```

zipalign 参数

-v：输出详细信息

-f：覆盖输出文件

-c：检查是否对齐

4：对齐 4 个字节

文档：https://developer.android.com/studio/command-line/zipalign

https://developer.android.com/studio/publish/app-signing#signing-manually

zipalign 例子

```
def zipalign():
    cmd = "zipalign 4 {apkOutputDir}/unsign_unalign_app.apk {apkOutputDir}/unsign_aligned_app.apk".format(
        apkOutputDir=apkOutputDir)
    subprocess.call(cmd, shell=True)
```

apksigner 参数

--ks：keystore 文件

--ks-key-alias：keystore 别名

--ks-pass：keystore 密码

--out：签名后输出文件

文档：https://developer.android.com/studio/command-line/apksigner

https://developer.android.com/studio/publish/app-signing#signing-manually

apksigner 例子

```
def apksigner():
    cmd = "apksigner sign --ks ~/.android/debug.keystore --ks-key-alias androiddebugkey --ks-pass pass:android --key-pass pass:android --out {apkOutputDir}/signed_aligned_app.apk {apkOutputDir}/unsign_aligned_app.ap
        apkOutputDir=apkOutputDir)
    subprocess.call(cmd, shell=True)
```

Q&A