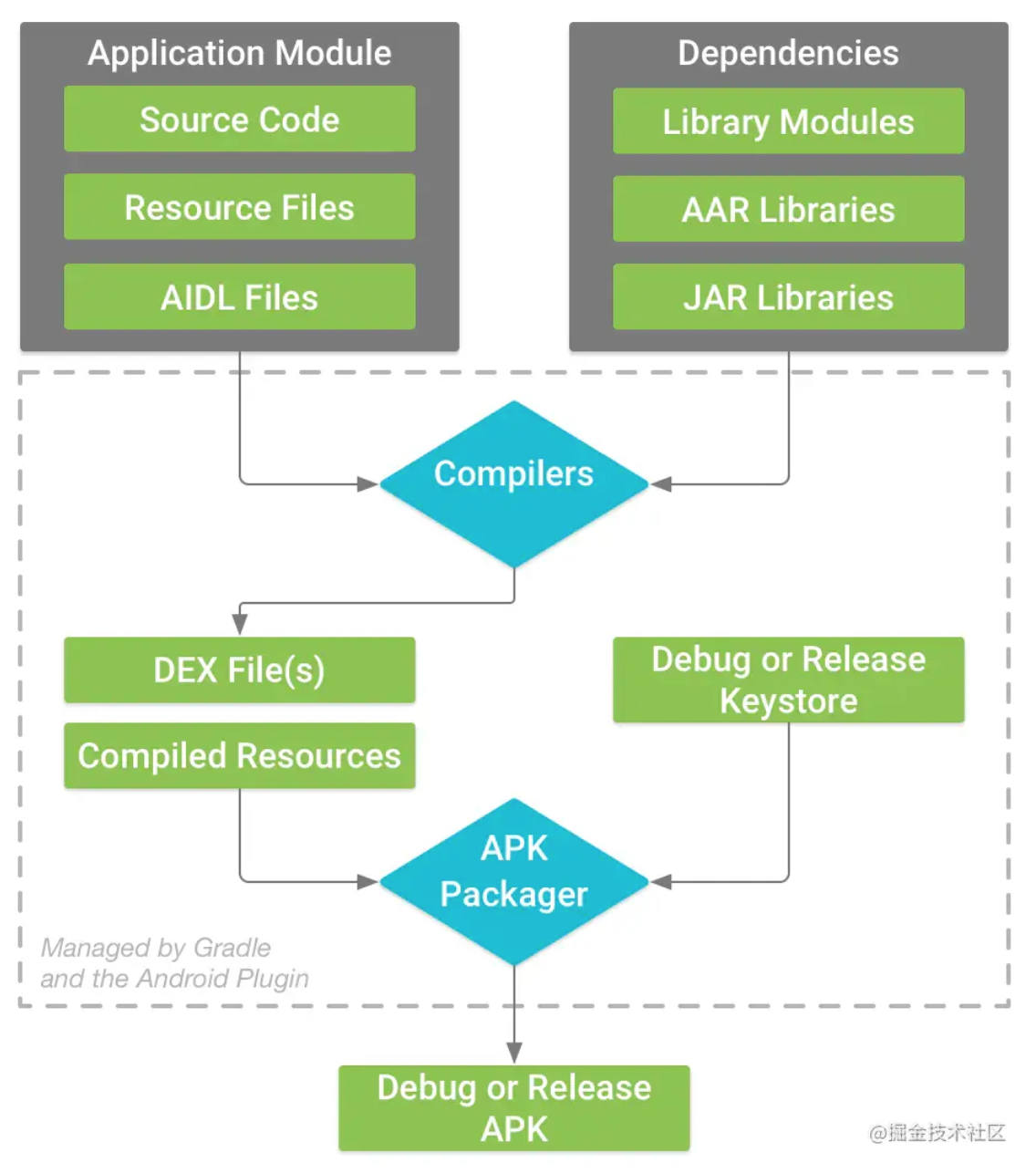
正常情况下，编译流程都是从下图说起的。



 通过 aapt 打包 res 资源文件，生成 R.java、resources.arsc 和 res 文件（二进制 & 非二进制如 res/raw 和 pic 保持原样）；

 处理 .aidl 文件，生成对应的 Java 接口文件；

 通过 Java Compiler 编译 R.java、Java 接口文件、Java 源文件，生成 .class 文件；

 通过 dex 命令，将 .class 文件和第三方库中的 .class 文件处理生成 classes.dex；

 通过 apkbuilder 工具，将 aapt 生成的 resources.arsc 和 res 文件、assets 文件和 classes.dex 一起打包生成 apk；

 通过 Jarsigner 工具，对上面的 apk 进行 debug 或 release 签名；

 通过 zipalign 工具，将签名后的 apk 进行对齐处理。

**从gradle Task看编译流程**

先贴一段gradle打印task耗时的代码

import java.util.concurrent.TimeUnit

// Log timings per task.

class TimingsListener implements TaskExecutionListener, BuildListener {

private long startTime

private timings = []

@Override

void beforeExecute(Task task) {

startTime = System.nanoTime()

}

@Override

void afterExecute(Task task, TaskState taskState) {

def ms = TimeUnit.MILLISECONDS.convert(System.nanoTime() - startTime, TimeUnit.NANOSECONDS);

timings.add([ms, task.path])

task.project.logger.warn "${task.path} took ${ms}ms"

}

@Override

void buildFinished(BuildResult result) {

println "Task timings:"

for (timing in timings) {

if (timing[0] >= 50) {

printf "%7sms %s\n", timing

}

}

}

@Override

void buildStarted(Gradle gradle) {}

@Override

void projectsEvaluated(Gradle gradle) {}

@Override

void projectsLoaded(Gradle gradle) {}

@Override

void settingsEvaluated(Settings settings) {}

}

gradle.addListener new TimingsListener()

当项目运行完之后会输出类似如下的日志，表示一个run执行之后gradle所执行的task的时间以及任务名。

1543ms :compiler:kaptGenerateStubsKotlin

144ms :RouterLib:packageDebugResources

1166ms :compiler:kaptKotlin

816ms :compiler:compileKotlin

401ms :compiler:compileJava

65ms :compiler:jar

122ms :app:mergeDebugResources

56ms :EmptyLoader:compileJava

170ms :app:processDebugManifest

171ms :RouterLib:parseDebugLocalResources

60ms :app:checkDebugDuplicateClasses

2416ms :RouterLib:compileDebugKotlin

122ms :RouterLib:compileDebugJavaWithJavac

124ms :secondmoudle:mergeDebugNativeLibs

1185ms :app:processDebugResources

70ms :secondmoudle:kaptGenerateStubsDebugKotlin

202ms :RouterLib:mergeDebugNativeLibs

350ms :secondmoudle:kaptDebugKotlin

158ms :secondmoudle:compileDebugJavaWithJavac

1108ms :app:kaptGenerateStubsDebugKotlin

91ms :secondmoudle:bundleLibRuntimeToJarDebug

129ms :app:mergeDebugNativeLibs

430ms :app:kaptDebugKotlin

1008ms :app:compileDebugKotlin

120ms :app:compileDebugJavaWithJavac

265ms :app:mergeDebugJavaResource

181ms :app:transformClassesAndResourcesWithAuto\_registerForDebug

7262ms :app:dexBuilderDebug

1308ms :app:mergeProjectDexDebug

344ms :app:packageDebug

从上述Task列表中可以看出，其实最上面这张图所说的编译流程其实并不完整。

## kapt和apt

javaCompiler执行之前会先执行apt，生成java代码，其任务名就是kaptGenerateStubsDebugKotlin。

## compiler

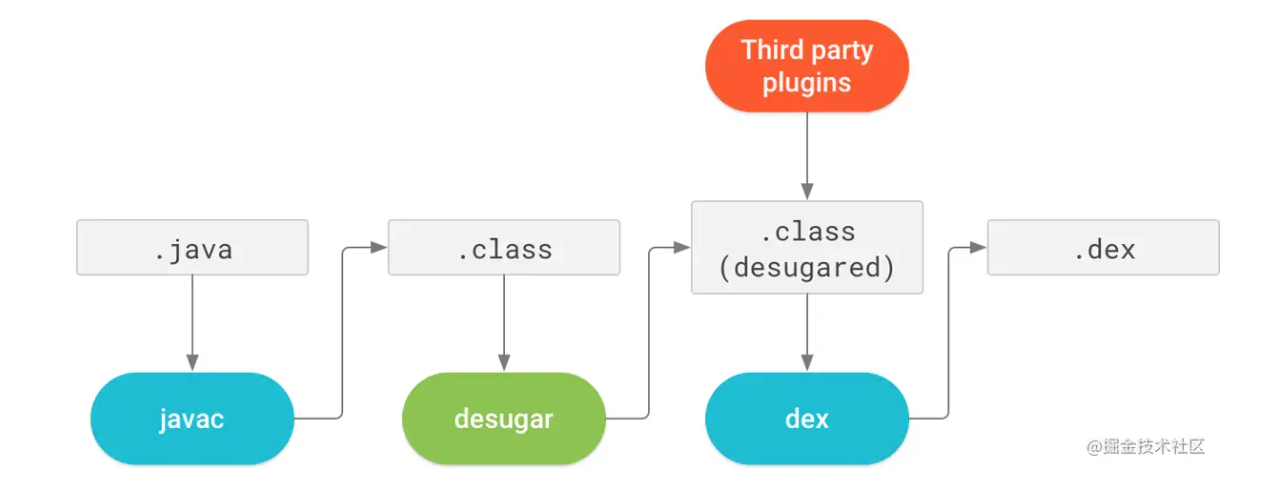
当我们使用字节码插桩之后其实就增加了个transform的流程，也就是这个**transformClassesAndResourcesWithAuto\_registerForDebug**。

## transform

AGP在不同版本的差异还是比较大的。特别是在3.2版本之上的版本被引入了D8编译器之后。

低版本先使用DX编译器将class转化为dex。

而高版本采用**d8**编译器将class转化为dex。

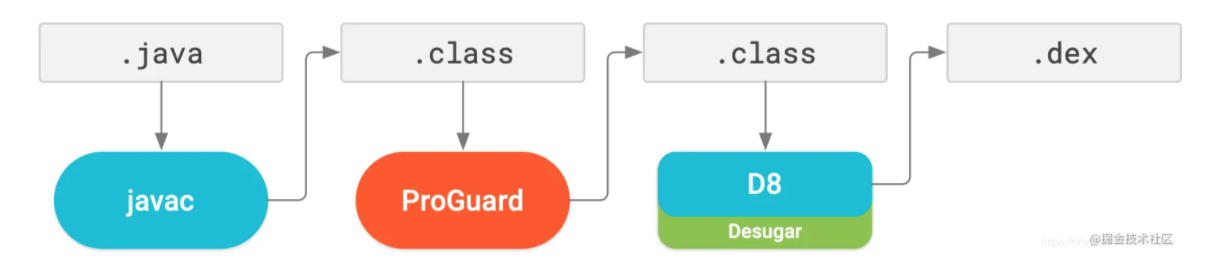


Android Studio 为使用部分 Java 8 语言功能及利用这些功能的第三方库提供内置支持。默认工具链对 javac 编译器的输出执行字节码转换（称为 desugar），从而实现新语言功能。

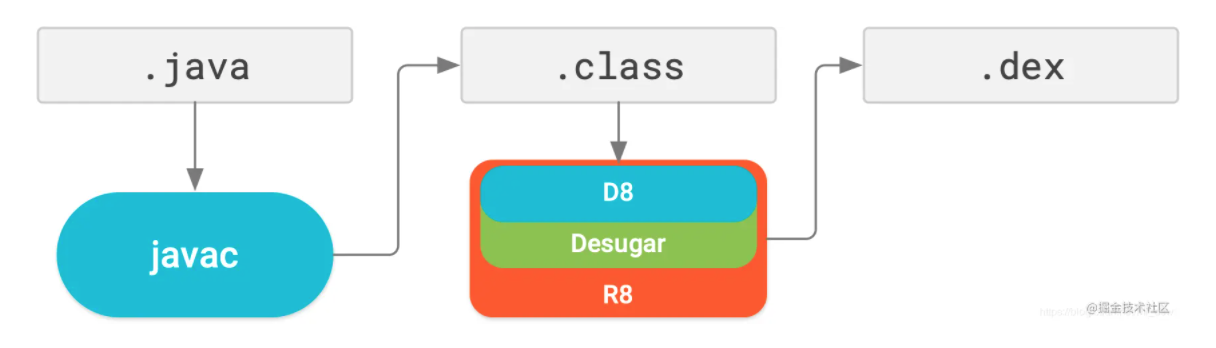
## 混淆

在AGP3.4版本上引入了R8，也就是混淆升级版本。而且在高版本上，整体流程也其实发生了微妙的变更，将原先的流程进行了合并。

1. R8开启前的编译流程



1. R8开启后的编译流程



## 签名

注意：您必须在应用构建过程中的两个特定时间点之一使用 zipalign，具体在哪个时间点使用，取决于您所使用的应用签名工具：

如果您使用的是 apksigner，则只能在为 APK 文件签名之前执行 zipalign。如果您在使用 apksigner 为 APK 签名之后对 APK 做出了进一步更>改，签名便会失效。

如果您使用的是 jarsigner，则只能在为 APK 文件签名之后执行 zipalign。

那么当使用V1签名时，编译流程顺序还是6-7

而当使用的是V2的签名时，则编译流程顺序是7-6