# FB App 优化工具 ReDex 优化的 6 点及未优化的 1 点

2016-04-18 Trinea codeKK

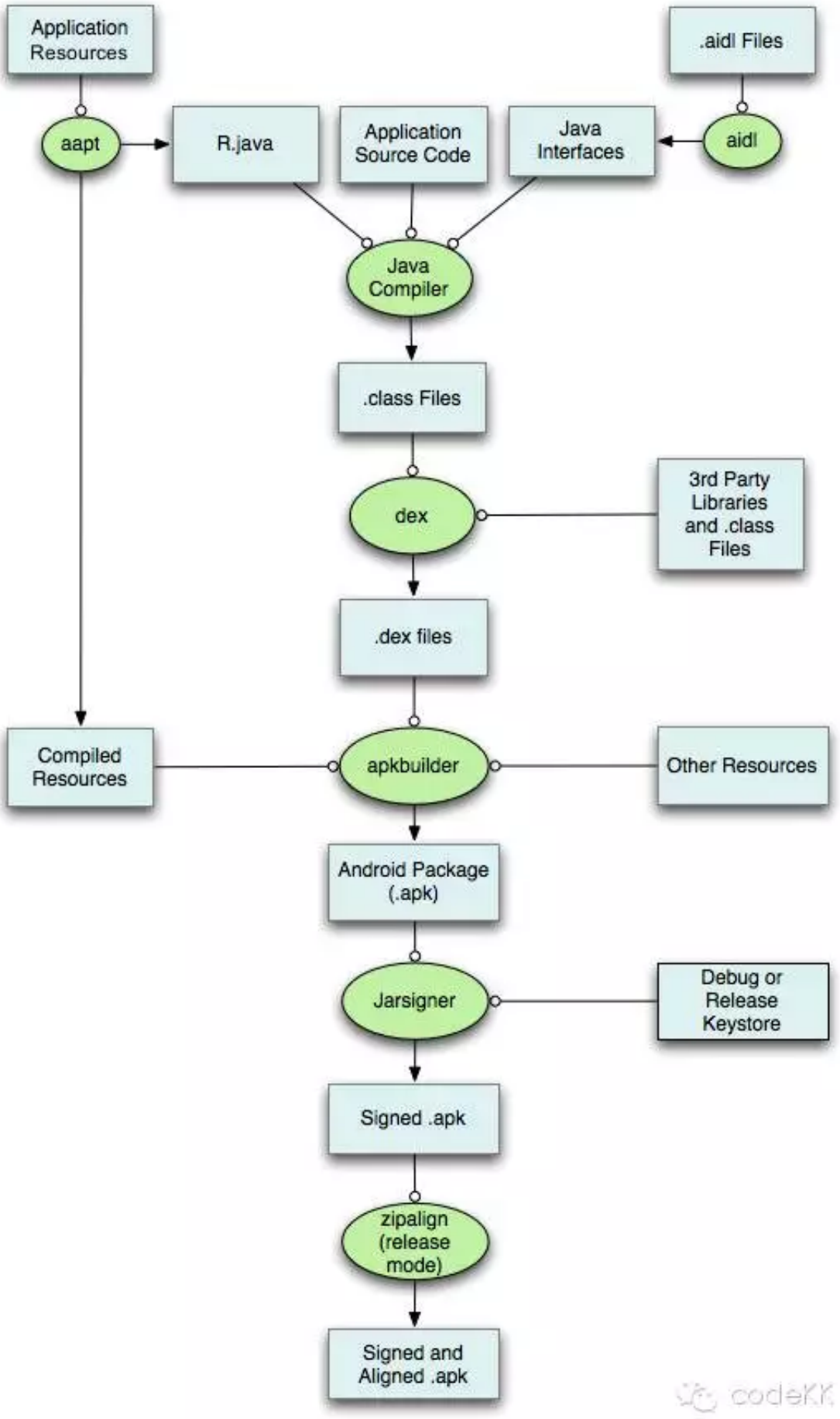
ReDex 是 Facebook 开源的工具，通过对字节码进行优化，以减小 Android Apk 大小，同时提高 App 启动速度。

去年十月 Facebook 就写过一篇文章《Optimizing Android bytecode with ReDex》对其进行介绍，终于在上周 ReDex 开源了。

Facebook 通过线上及线下测试，启动速度提升 20% 以上，Dex 大小减小 25%，对于内存较小的机型启动速度的优化效果尤其明显。

## 设计介绍

1. 基于 Dex 的优化




ReDex 是基于 Dex 文件的字节码进行优化，从上图 Android 编译过程我们可以看出，Java 源文件经过 Java 编译器转换为 .class 文件，再转换为虚拟机字节码和三方库一起转换为 dex 文件。

Dex 优化相比基于源码或者 Java 字节码的好处是：
(1) 可以最大限度的从全局以及类间进行优化；
(2) 可以类似于 C 在编译最后的 Linking 阶段做优化一样进行优化。
PS：Proguard 就是基于 Java 字节码的优化。

### 2. 基于管道的优化过程设计



ReDex 设计时将优化过程分为不同阶段，类似污水处理分为机械处理-生物处理-深度处理等阶段，每个阶段可当做优化插件进行插拔(取舍)，并可以跟在其他不相关的阶段后面，这样的好处有：
(1) 方便多个优化阶段并行开发；
(2) 方便后续添加新的优化阶段以及开源接受更多的布局优化；
(3) 用户可以通过配置方便的决定用哪些优化。

## 6 个优化点介绍

ReDex 主要的优化在于减小 APK 大小以及提高启动速度，分为 6 大点：

### 1. 基于反馈(启动加载顺序测试)的 Class 字节码布局

类字节码在单个 Dex 中的布局是根据编译顺序而不是运行时行为决定，即便 Multidex 会将 App 定义的组件以及部分启动需要的类放在 Main Dex 中，但依然不是根据运行时顺序布局，这会导致程序启动时需要查找随机分布在 Dex 中的类。

ReDex 会将 APK 在 Lab 中试运行，并跟踪启动时哪些类需要加载，然后将这些类字节码放到 Dex 前部，减少启动时从闪存中寻找类的时间，从而提高 App 启动速度。

### 2. 混淆和压缩

跟 JS 的压缩以及 Android 的 Proguard 类似。

### 3. 内联函数

将一些函数直接展开到调用它的函数中，减少函数调用切换的时间消耗。

### 4. 删除无用的 interfaces

删除只有一个实现的接口，用实现直接代替。一定程度加快函数调用时间，并减少内存空间以及函数引用。

### 5. 删除无用代码

通过类似早期内存回收的标记-删除策略，删除无用代码。

### 6. 删除 metadata

metadata 的一些数据在运行中并不需要，用 Dex 中已有的字符串代替源文件引用以及删除无用的注解来减小 Dex 大小。

ReDex 自身的运行速度很快，虽然是针对 Dex 的优化，但接受 APK 参数，方便与已有编译系统集成。

## 注意事项

(1) 对于不能删除的代码需要额外配置，如 JNI 调用、反射、(布局文件中调用？)等特殊情况。

(2) 签名后的 APK ReDex 处理后需要重新签名，因为 Dex 文件已经发生变化。

## 配置

(1) 可通过 -c 指定具体的优化阶段配置文件

redex -c oppass.config -o tmp/output.apk input.apk

其中 oppass.config 可类似于 ReDex 源码下 config/default.config 文件，用于配置采用哪些优化阶段。

(2) 每个优化阶段可以单独做额外的配置

比如上面说的需要反射的类不能删除，文件路径缩略配置表，启动时先后加载的类列表。

(3) Proguard 文件列表

通常情况下，ReDex 会和 Proguard 一起对 APK 进行优化，并且运行在 Proguard 后，通过 Proguard 文件列表的配置 ReDex 就可正确识别混淆后的类。

## 其他优化：

从 Android 编译过程图中我们也可以看出，APK 包除了 Dex 文件外，还包含 Resource 部分，这部分可优化的空间也很大，下次分享相关资料。