Android 虚拟机进化史

从 Android 2.1 版本到现在的 Android 11，中间虚拟机变化过三次：

版本	虚拟机类型	特性
2.1-4.4	Dalvik	JIT+解释器
5.0-7.0	ART	AOT
7.0-11	ART	AOT+JIT+解释器

Dalvik 和 JVM 的主要区别

首先通过介绍 Dalvik 的时候我们就知道 Dalvik 运行的是 dex 文件，而 JVM 运行的是 class 文件。

Dalvik VM 是基于寄存器的架构，而 JVM 是栈机。所以 Dalvik VM 的好处是可以做到更好的提前优化（ahead-of-time optimization）。 另外基于寄存器架构的 VM 执行起来更快，但是代价是更大的代码长度。

Dalvik 在 JVM 上的优化

1.在编译时提前优化代码而不是等到运行时。
2.虚拟机很小，使用的空间也小，被设计来满足可高效运行多种虚拟机实例。
3.常量池已被修改为只使用 32 位的索引，以简化解释器。
4.标准 Java 字节码实行 8 位堆栈指令，Dalvik 使用 16 位指令集直接作用于局部变量。局部变量通常来自 4 位的“虚拟寄存器”区。这样减少了 Dalvik 的指令计数，提高了翻译速度。

Dalivk 进化之 ART

在 Android 5.0 之后完全弃用 dalvik 全部采用 art 为执行环境。ART（ Android Runtime）ART 的机制与 Dalvik 不同。在 Dalvik 下，应用每次运行的时候，字节码都需要通过即时编译器（just in time ，JIT）转换为机器码，这会拖慢应用的运行效率，而在 ART 环境中，应用在第一次安装的时候，字节码就会预先编译成机器码，使其成为真正的本地应用。这个过程叫做预编译（AOT Ahead-Of-Time）。这样的话，应用的启动(首次)和执行都会变得更加快速。

ART 的优缺点

优点：
1.系统性能的显著提升。
2.应用启动更快、运行更快、体验更流畅、触感反馈更及时。
3.更长的电池续航能力。
4.支持更低的硬件。
缺点：
1.机器码占用的存储空间更大，字节码变为机器码之后，可能会增加 10%-20（不过在应用包中，可执行的代码常常只是一部分。比如最新的 Google+ APK 是 28.3 MB，但是代码只有 6.9 MB。）
2.应用的安装时间会变长。

class、dex、odex、ELF 相爱相杀

从执行文件上面进行分析的话，JVM 对应 class 文件，Dalivk 对应 odex 文件，而 ART 对应 oat 文件。

工具：javac，dx

.java ——> .class ——> .dex

.java 文件经过 javac 编译器生成 .class 字节码，再经过 dx 工具生成 .dex 。

为了在 JVM 优化出一个 Dalivk 虚拟机，所以把 JVM 运行的 class 文件进行打包优化为 dex 文件，但其本质还是和 class 文件一样属于字节码文件。但是为了每次启动时都去掉从字节码到机器码的编译过程，Google 又从 Dalivk 中优化出了 ART，在其安装应用的时候将 dex 文件进行预处理生成可执行的 oat 文件。

JIT 的引入

据说 Android 2.2 的虚拟机 dalvik 使用了 JIT 技术，使其运行速度快了 5 倍。

dalvik 解释并执行程序，JIT 技术主要是对多次运行的代码进行编译，当再次调用时使用编译之后的机器码，而不是每次都解释，以节约时间。

每启动一个应用程序，都会相应地启动一个 dalvik 虚拟机，启动时会建立 JIT 线程，一直在后台运行。当某段代码被调用时，虚拟机会判断它是否需要编译成机器码，如果需要，就做一个标记，JIT 线程不断判断此标记，如果发现被设定就把它编译成机器码，并将其机器码地址及相关信息放入 entry table 中，下次执行到此就跳到机器码段执行，而不再解释执行，从而提高速度。

odex (optimized dex)

因为 apk 实际为 zip 压缩包，虚拟机每次加载都需要从 apk 中读取 classes.dex 文件，这样会耗费很多的时间，而如果采用了 odex 方式优化的 dex 文件，他包含了加载 dex 必须的依赖库文件列表，只需要直接加载而不需要再去解析。

在 Android N 之前，对于在 dalvik 环境中 使用 dexopt 来对 dex 字节码进行优化生成 odex 文件最终存在手机的 data/dalvik-cache 目录下，最后把 apk 文件中的 dex 文件删除。

在 Android O 之后，odex 是从 vdex 这个文件中提取了部分模块生成的一个新的可执行二进制码文件，odex 从 vdex 中提取后，vdex 的大小就减少了。

1.第一次开机就会生成在 /system/app/<packagename>/oat/ 下

2.在系统运行过程中，虚拟机将其 从 /system/app 下 copy 到 /data/davilk-cache/ 下

3.odex + vdex = apk 的全部源码 （vdex 并不是独立于 odex 的文件，odex + vdex 才代表一个 apk ）

vdex

dex2oat 工具接受一个 APK 文件，并生成一个或多个编译工件文件，然后运行时将会加载这些文件。文件的个数、扩展名和名称会因版本而异。

在 Android O 版本中，将会生成以下文件：

1..vdex：其中包含 APK 的未压缩 DEX 代码，另外还有一些旨在加快验证速度的元数据。

2..odex：其中包含 APK 中已经过 AOT 编译的方法代码。

3..art (optional) : 其中包含 APK 中列出的某些字符串和类的 ART 内部表示，用于加快应用启动速度。

1.第一次开机就会生成在 /system/app/<packagename>/oat/ 下。
2.在系统运行过程中，虚拟机将其 从 /system/app 下 copy 到 /data/davilk-cache/ 下。

文件格式	用途
.dex	存储java字节码
.odex/.oat	optimized dex，ELF格式
.vdex	verified dex，包含 raw dex + （quicken info)
.art	image文件，存储热点方法string, method, types等

AOT (Ahead-of-time)
ART 推出了预先 (AOT) 编译，可提高应用的性能。ART 还具有比 Dalvik 更严格的安装时验证。在安装时，ART 使用设备自带的 dex2oat 工具来编译应用。该实用工具接受 DEX 文件作为输入，并针对目标设备生成已编译应用的可执行文件。之后打开 App 的时候，不需要额外的翻译工作，直接使用本地机器码运行，因此运行速度提高。

AOT 是 art 的核心，oat 文件包含 oatdata 和 oatexec。前者包含 dex 文件内容，后者包含生成的本地机器指令，从这里看出 oat 文件回会比 dex 文件占用更大的存储空间。

因为 oat 文件包含生成的本地机器指令进而可以直接运行，它同样保存在手机的 data/dalvik-cache 目录下，PMS(PackgetManagerService) —> installd(守护进程) ——> dex2oat(/system/bin/dex2oat) 。注意存放在 data/dalvik-cache 目录下的后缀名都仍为 .dex，前者其实表示一个优化过的 .dex 文件，后者为 .art 文件。

push 一个新的 apk 文件覆盖之前 /system/app 下 apk 文件，会触发 PKMS 扫描时下发 force_dex flag ，强行生成新的 vdex 文件 ，覆盖之前的 vdex 文件。由于某种机制，这个新 vdex 文件会 copy 到 /data/dalvik-cache/ 下，于是 art 文件也变化了。

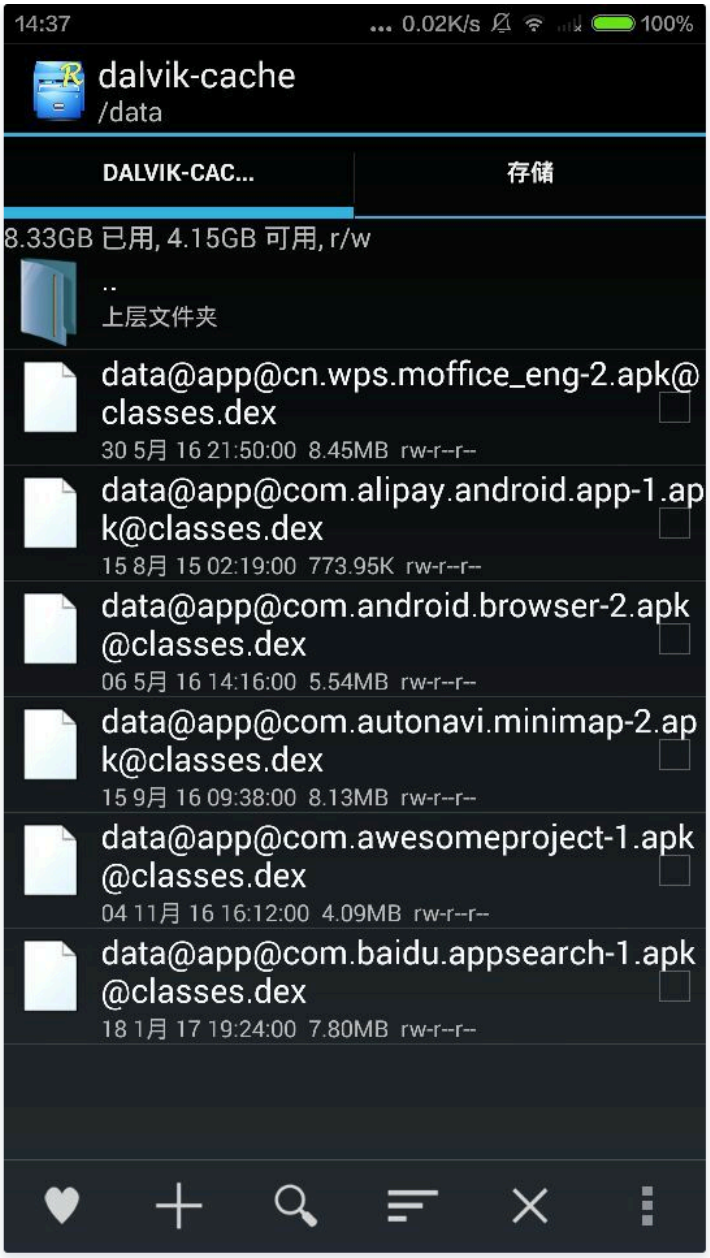
混合运行时
Android N 开发者预览版包含了一个混合模式的运行时。应用在安装时不做编译，而是解释字节码，所以可以快速启动。ART 中有一种新的、更快的解释器，通过一种新的 JIT 完成，但是这种 JIT 的信息不是持久化的。取而代之的是，代码在执行期间被分析，分析结果保存起来。然后，当设备空转和充电的时候，ART 会执行针对 “热代码” 进行的基于分析的编译，其他代码不做编译。为了得到更优的代码，ART 采用了几种技巧包括深度内联。

对同一个应用可以编译数次，或者找到变 “热” 的代码路径或者对已经编译的代码进行新的优化，这取决于分析器在随后的执行中的分析数据。这个步骤仍被简称为 AOT，可以理解为 “全时段的编译” （All-Of-the-Time compilation）。

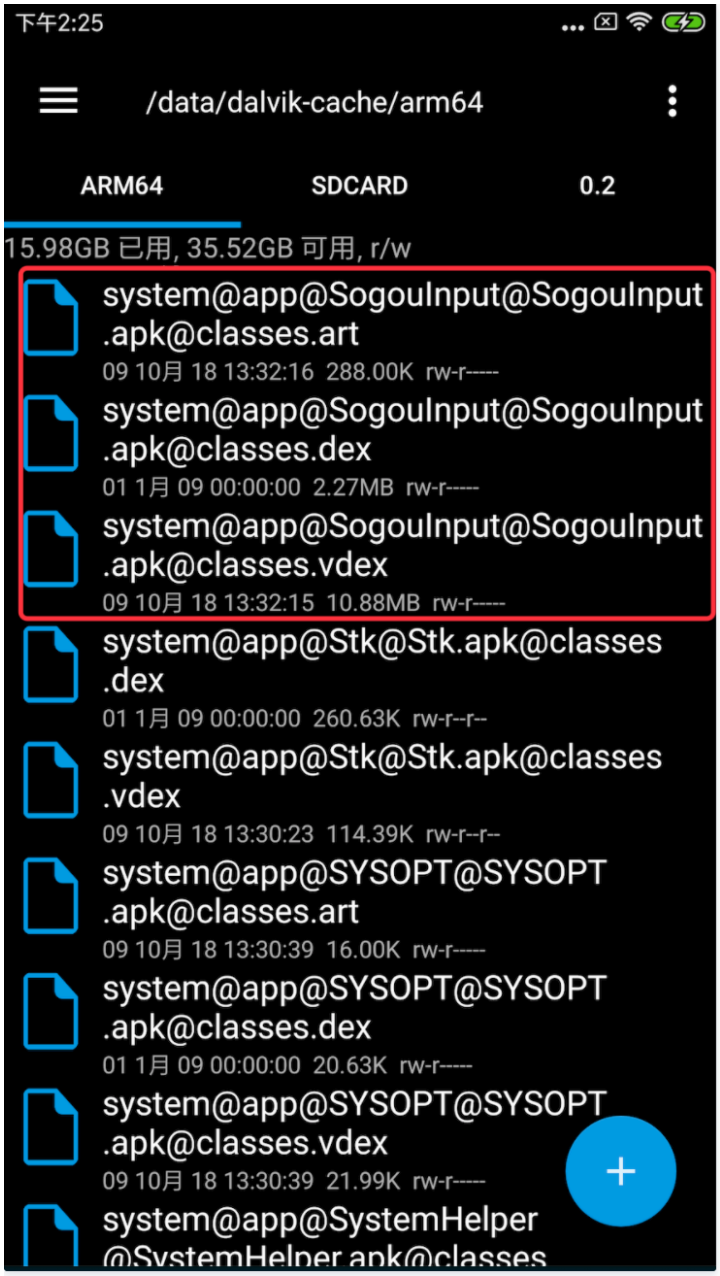
这种混合使用 AOT、解释、JIT 的策略的全部优点如下。
1.即使是大应用，安装时间也能缩短到几秒
2.系统升级能更快地安装，因为不再需要优化这一步
3.应用的内存占用更小，有些情况下可以降低 50%
4.改善了性能
5.更低的电池消耗

apk 安装过程

大家都知道 apk 其实就是 zip 包 apk 安装过程其实就是解压过程。
用户应用安装涉及以下几个目录：
1.data/app 安装目录，安装时会把 apk 文件 copy 到这里
2.data/dalvik-cache 如上述描述中的存放.dex (.odex 无论 davilk 的 dex 还是 art 的 oat 格式)
3.data/data/pkg/ 存放应用程序的数据
Android5.1 版本下 oat 文件都以 .dex 文件在 data/dalvik-cache 目录下：



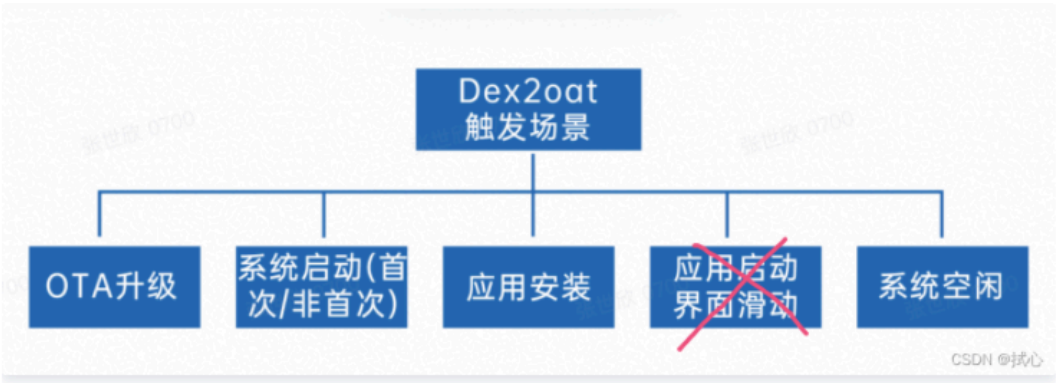
Android 8.0 版本下 dex2oat 工具生成的三个.art, .odex, .vdex 文件都在 data/dalvik-cache 目录下：

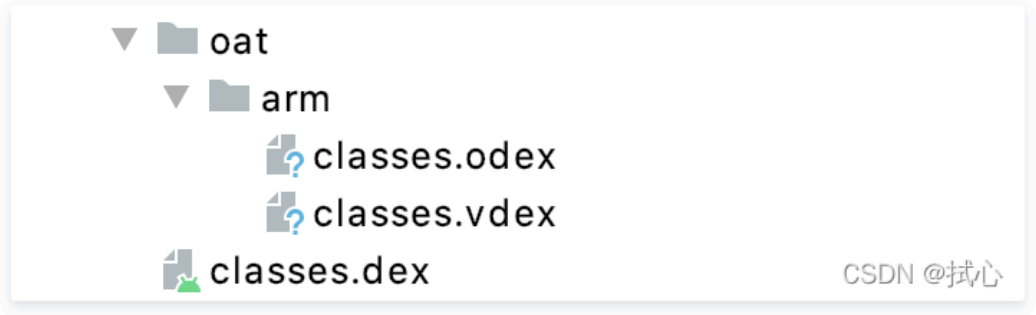


dex2oat 触发场景

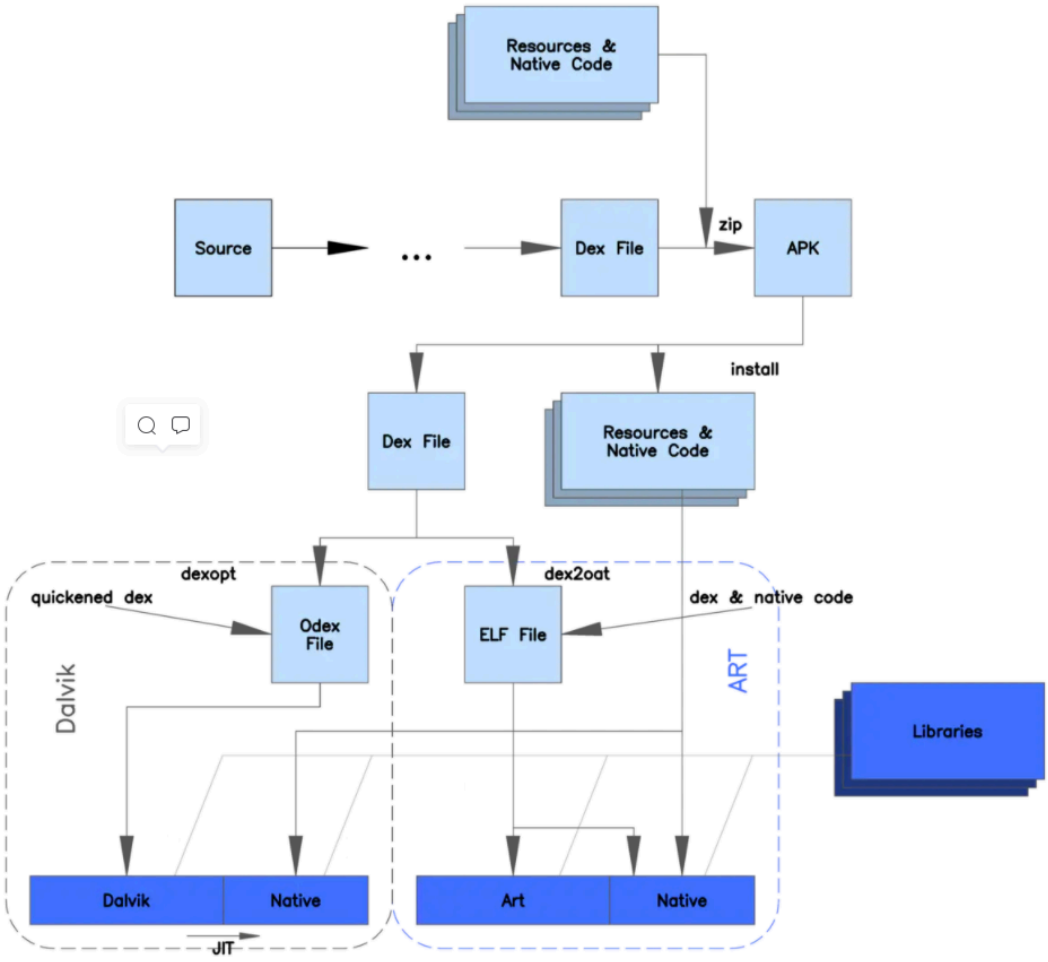
https://cs.android.com/android/platform/superproject/+/master:art/dex2oat/dex2oat.cc

dex2oat 的作用：对 dex 文件进行编译，根据参数，生成 oat vdex art 文件。





dexopt 与 dex2oat 的区别



前者针对 Dalvik 虚拟机，后者针对 Art 虚拟机。
dexopt 是对 dex 文件 进行 verification 和 optimization 的操作，其对 dex 文件的优化结果变成了 odex 文件，这个文件和 dex 文件很像，只是使用了一些优化操作码（譬如优化调用虚拟指令等）。
dex2oat 是对 dex 文件的 AOT 提前编译操作，其需要一个 dex 文件，然后对其进行编译，结果是一个本地可执行的 ELF 文件，可以直接被本地处理器执行。
除此之外在上图还可以看到 Dalvik 虚拟机中有使用 JIT 编译器，也就是说其也能将程序运行的热点 java 字节码编译成本地 code 执行，所以其与 Art 虚拟机还是有区别的。

Android 各版本虚拟机 dexopt 产物的区别

5.0 以下

使用 Dalvik 虚拟机 ，生成 odex 文件 . Dalvik 采用的是 JIT 编译+解释器，也就是即时编译，每次应用运行时会实时将 Dex 翻译成机器码。
优点：安装速度超快 ，占用存储空间小
缺点：
1.由于在 Dex 加载时会触发 dexopt ，导致 Multidex 加载的时候会非常慢
2.由于热点代码的 Monitor 一直在运行 ，解释器解释的字节码会带来 CPU 和时间的消耗，会带来电量的损耗

5.0 - 7.0

使用 ART 虚拟机 ，生成 oat 文件. 在 ROM OTA 或者恢复出场设置后 ，会要进行 dex2oat 根据当前 ROM 进行重新编译生成 .oat 文件。
优点：运行时省电 ，运行速度快
缺点：
1.由于安装 APK 时触发 dex2oat ，需要编译成 native code ，导致安装时间过长
2.由于 dex2oat 生成的文件较大 ，会占用较多的空间

7.0 - 8.0

使用 ART 虚拟机 ，但是在 7.0 之上 ，增加了.vdex 与 .art 机制 ，在 ART 虚拟机再次启动/升级 ，加载 Dex/Oat 文件时 ，会减少 Dex 的校验时间 ，提升加载与运行效率。

9.0

在 ART 虚拟机的基础上 ，增加了 Cdex (Compat Dex)机制。

Compiler-fileter

在 dex2oat 的时候 ，会有一个目标编译类型 ，会有以下几类 ，根据时机不同 dex2oat 的编译方式也会不同。
verify：只运行 DEX 代码验证。
quicken：运行 DEX 代码验证，并优化一些 DEX 指令，以获得更好的解译器性能。
speed：运行 DEX 代码验证，并对所有方法进行 AOT 编译。
speed-profile：运行 DEX 代码验证，并对配置文件中列出的方法进行 AOT 编译。