#### 阿里巴巴 --接下来说说 Android 虚拟机Dalvik与ART区别在哪里？

本专栏专注分享大型Bat面试知识，后续会持续更新，喜欢的话麻烦点击一个star

Android开发中我们接触的是与Java虚拟机类似的Dalvik虚拟机和ART虚拟机，下面梳理一下三者区别和原理：

一，Dalvik虚拟机 Dalvik虚拟机（ Dalvik Virtual Machine ），简称Dalvik VM或者DVM。Dalvik 发音有道词典并没有收录。说说来历，它是由Dan Bornstein编写的，名字源于他的祖先居住过的名为Dalvik的小渔村。DVM是Google专门为Android平台开发的虚拟机，它运行在Android运行时库中。需要注意的是DVM并不是一个Java虚拟机（以下简称JVM）

##### DVM与JVM的区别

DVM之所以不是一个JVM ，主要原因是DVM并没有遵循JVM规范来实现。DVM与JVM主要有以下区别。

**基于的架构不同** JVM基于栈则意味着需要去栈中读写数据，所需的指令会更多，这样会导致速度慢，对于性能有限的移动设备，显然不是很适合。 DVM是基于寄存器的，它没有基于栈的虚拟机在拷贝数据而使用的大量的出入栈指令，同时指令更紧凑更简洁。但是由于显示指定了操作数，所以基于寄存器的指令会比基于栈的指令要大，但是由于指令数量的减少，总的代码数不会增加多少。

##### 执行的字节码不同

在Java SE程序中，Java类会被编译成一个或多个.class文件，打包成jar文件，而后JVM会通过相应的.class文件和jar文件获取相应的字节码。执行顺序为： .java文件 -> .class文件 -> .jar文件 而DVM会用dx工具将所有的.class文件转换为一个.dex文件，然后DVM会从该.dex文件读取指令和数据。执行顺序为： .java文件 –>.class文件-> .dex文件



img

如上图所示，.jar文件里面包含多个.class文件，每个.class文件里面包含了该类的常量池、类信息、属性等等。当JVM加载该.jar文件的时候，会加载里面的所有的.class文件，JVM的这种加载方式很慢，对于内存有限的移动设备并不合适。 而在.apk文件中只包含了一个.dex文件，这个.dex文件里面将所有的.class里面所包含的信息全部整合在一起了，这样再加载就提高了速度。.class文件存在很多的冗余信息，dex工具会去除冗余信息，并把所有的.class文件整合到.dex文件中，减少了I/O操作，提高了类的查找速度。

##### DVM允许在有限的内存中同时运行多个进程

DVM经过优化，允许在有限的内存中同时运行多个进程。在Android中的每一个应用都运行在一个DVM实例中，每一个DVM实例都运行在一个独立的进程空间。独立的进程可以防止在虚拟机崩溃的时候所有程序都被关闭。

##### DVM由Zygote创建和初始化

在Android系统启动流程（二）解析Zygote进程启动过程这篇文章中我介绍过 Zygote，可以称它为孵化器，它是一个DVM进程，同时它也用来创建和初始化DVM实例。每当系统需要创建一个应用程序时，Zygote就会fock自身，快速的创建和初始化一个DVM实例，用于应用程序的运行。

##### DVM架构

DVM的源码位于dalvik/目录下，其中dalvik/vm目录下的内容是DVM的具体实现部分，它会被编译成libdvm.so；dalvik/libdex会被编译成libdex.a静态库，作为dex工具使用；dalvik/dexdump是.dex文件的反编译工具；DVM的可执行程序位于dalvik/dalvikvm中，将会被编译成dalvikvm可执行程序。DVM架构如下图所示。



è¿éåå¾çæè¿°

从上图可以看出，首先Java编译器编译的.class文件经过DX工具转换为.dex文件，.dex文件由类加载器处理，接着解释器根据指令集对Dalvik字节码进行解释、执行，最后交与Linux处理。

##### DVM的运行时堆

DVM的运行时堆主要由两个Space以及多个辅助数据结构组成，两个Space分别是Zygote Space（Zygote Heap）和Allocation Space（Active Heap）。Zygote Space用来管理Zygote进程在启动过程中预加载和创建的各种对象，Zygote Space中不会触发GC，所有进程都共享该区域，比如系统资源。Allocation Space是在Zygote进程fork第一个子进程之前创建的，它是一种私有进程，Zygote进程和fock的子进程在Allocation Space上进行对象分配和释放。

##### 除了这两个Space，还包含以下数据结构：

Card Table： 用于DVM Concurrent GC，当第一次进行垃圾标记后，记录垃圾信息。 Heap Bitmap： 有两个Heap Bitmap，一个用来记录上次GC存活的对象，另一个用来记录这次GC存活的对象。 Mark Stack： DVM的运行时堆使用标记-清除（Mark-Sweep）算法进行GC，不了解标记-清除算法的同学查看Java虚拟机（四）垃圾收集算法这篇文章。Mark Stack就是在GC的标记阶段使用的，它用来遍历存活的对象。

#### 二.ART虚拟机

ART(Android Runtime)是Android 4.4发布的，用来替换Dalvik虚拟，Android 4.4默认采用的还是DVM，系统会提供一个选项来开启ART。在Android 5.0时，默认采用ART，DVM从此退出历史舞台。

##### ART与DVM的区别

1. DVM中的应用每次运行时，字节码都需要通过即时编译器（JIT，just in time）转换为机器码，这会使得应用的运行效率降低。而在ART中，系统在安装应用时会进行一次预编译（AOT，ahead of time）,将字节码预先编译成机器码并存储在本地，这样应用每次运行时就不需要执行编译了，运行效率也大大提升。
2. ART占用空间比Dalvik大（字节码变为机器码之后，可能会增加10%-20%），这就是“时间换空间大法”。
3. 预编译也可以明显改善电池续航，因为应用程序每次运行时不用重复编译了，从而减少了 CPU 的使用频率，降低了能耗。

##### ART的运行时堆

1. 与DVM的GC不同的是，ART的GC类型有多种，主要分为Mark-Sweep GC和Compacting GC。ART的运行时堆的空间根据不同的GC类型也有着不同的划分，如果采用的是Mark-Sweep GC，运行时堆主要是由四个Space和多个辅助数据结构组成，四个Space分别是Zygote Space、Allocation Space、Image Space和Large Object Space。Zygote Space、Allocation Space和DVM中的作用是一样的。Image Space用来存放一些预加载类，Large Object Space用来分配一些大对象（默认大小为12k）。其中Zygote Space和Image Space是进程间共享的。 采用Mark-Sweep GC的运行时堆空间划分如下图所示。
2. 
3. è¿éåå¾çæè¿°

##### 除了这四个Space，ART的Java堆中还包括两个Mod Union Table，一个Card Table，两个Heap Bitmap，两个Object Map，以及三个Object Stack