理解Android虚拟机体系结构

2016-01-27 安卓应用频道

(点击上方公众号，可快速关注)

> 来源：LeoLiang
>
> 链接：http://www.cnblogs.com/lao-liang/p/5111399.html

1 什么是Dalvik虚拟机

Dalvik是Google公司自己设计用于Android平台的Java虚拟机，它是Android平台的重要组成部分，支持dex格式（Dalvik Executable）的Java应用程序的运行。dex格式是专门为Dalvik设计的一种压缩格式，适合内存和处理器速度有限的系统。Google对其进行了特定的优化，使得Dalvik具有高效、简洁、节省资源的特点。从Android系统架构图知，Dalvik虚拟机运行在Android的运行时库层。

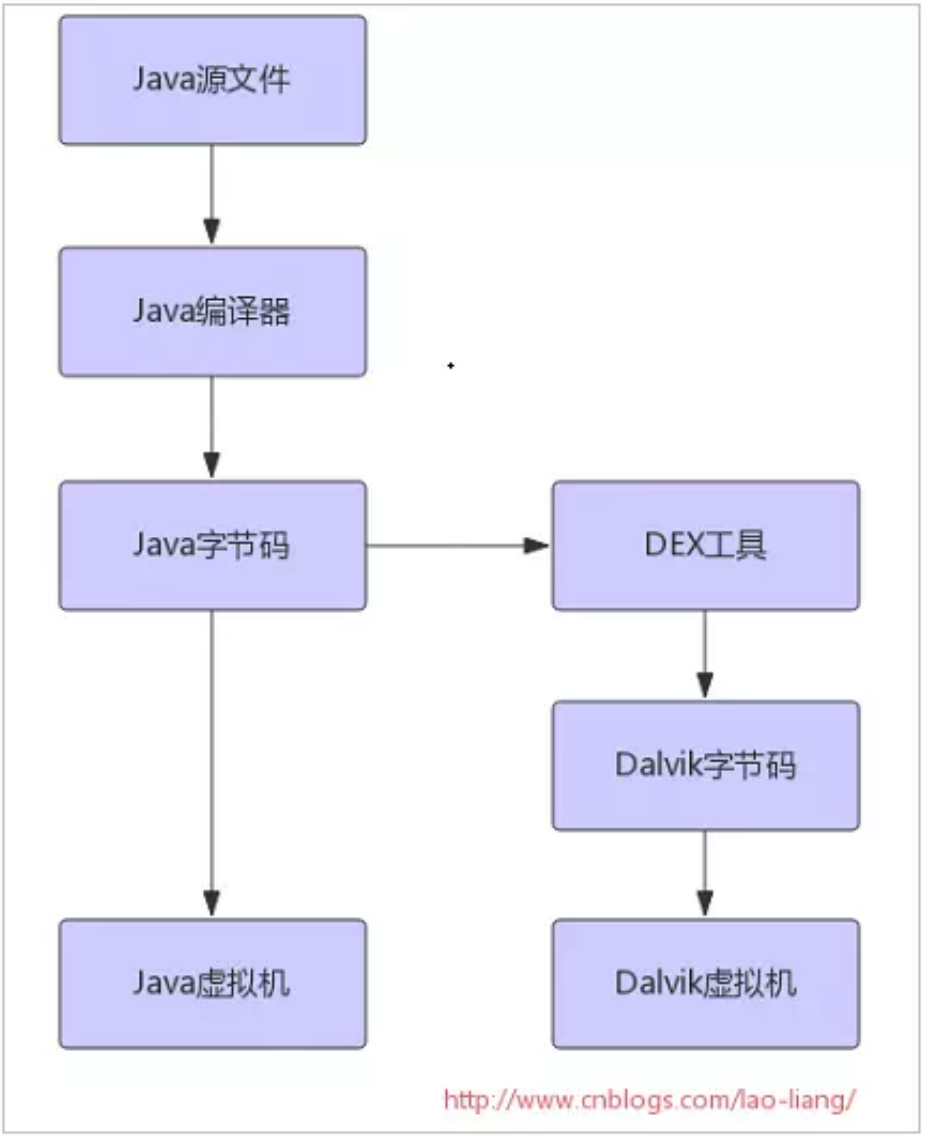
2 Dalvik虚拟机的功能

Dalvik作为面向Linux、为嵌入式操作系统设计的虚拟机，主要负责完成对象生命周期管理、堆栈管理、线程管理、安全和异常管理，以及垃圾回收等。Dalvik充分利用Linux进程管理的特定，对其进行了面向对象的设计，使得可以同时运行多个进程，而传统的Java程序通常只能运行一个进程，这也是为什么Android不采用JVM的原因。Dalvik为了达到优化的目的，底层的操作大多和系统内核相关，或者直接调用内核接口。另外，Dalvik早期并没有JIT编译器，直到Android2.2才加入了对JIT的技术支持。

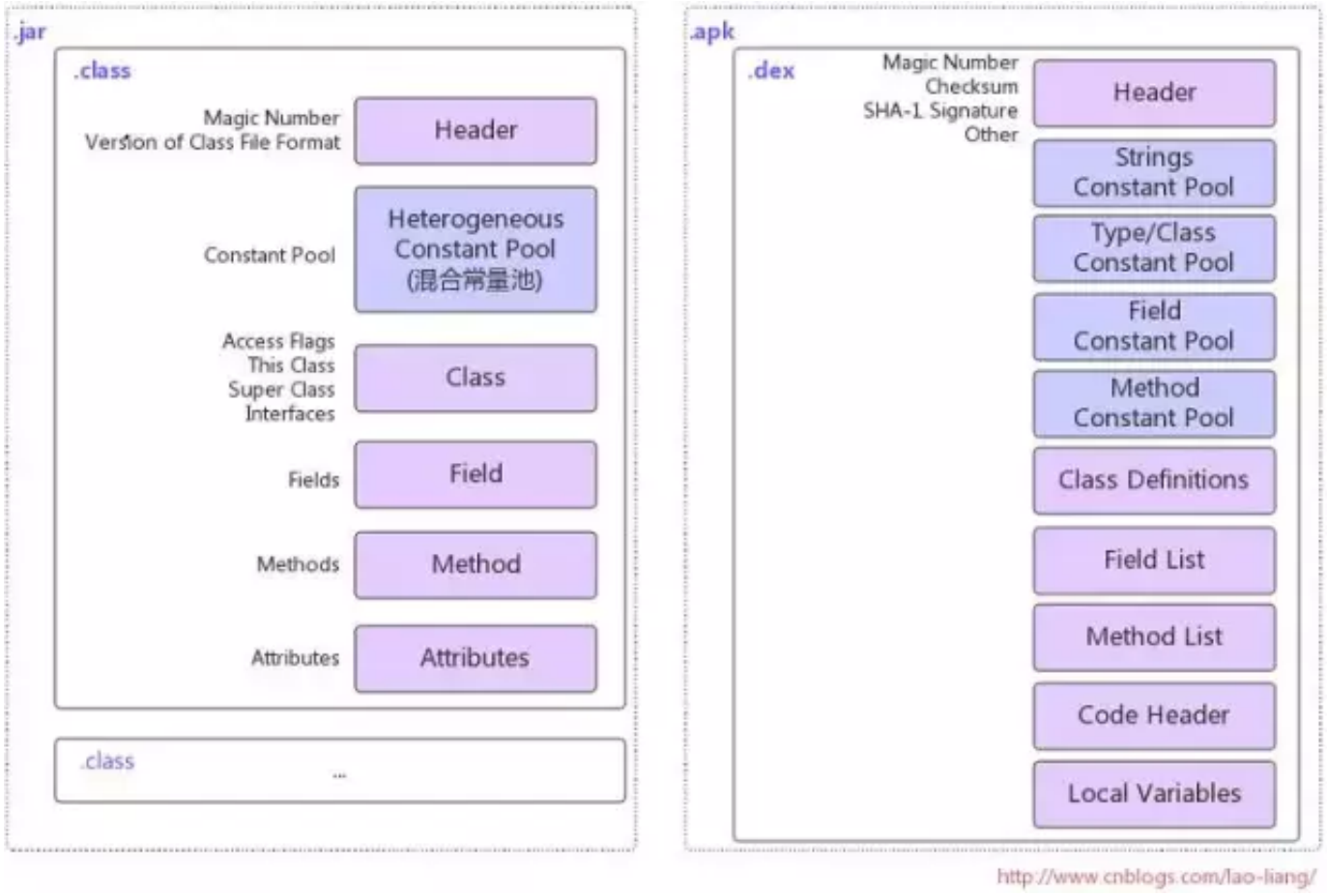
3 Dalvik虚拟机和Java虚拟机的区别

本质上，Dalvik也是一个Java虚拟机。但它特别之处在于没有使用JVM规范。大多数Java虚拟机都是基于栈的结构（详情请参考：理解Java虚拟机体系结构），而Dalvik虚拟机则是基于寄存器。基于栈的指令很紧凑，例如，Java虚拟机使用的指令只占一个字节，因而称为字节码。基于寄存器的指令由于需要指定源地址和目标地址，因此需要占用更多的指令空间。Dalvik虚拟机的某些指令需要占用两个字节。基于栈和基于寄存器的指令集各有优劣，一般而言，执行同样的功能，前者需要更多的指令（主要是load和store指令），而后者需要更多的指令空间。需要更多指令意味着要多占用CPU时间，而需要更多指令空间意味着数据缓冲（d-cache）更易失效。更多讨论，虚拟机随谈（一）：解释器，树遍历解释器，基于栈与基于寄存器，大杂烩 给出了非常详细的参考。

Java虚拟机运行的是Java字节码，而Dalvik虚拟机运行的是专有文件格式dex。在Java程序中，Java类会被编译成一个或多个class文件，然后打包到jar文件中，接着Java虚拟机会从相应的class文件和jar文件中获取对应的字节码。Android应用虽然也使用Java语言，但是在编译成class文件后，还会通过DEX工具将所有的class文件转换成一个dex文件，Dalvik虚拟机再从中读取指令和数据。dex文件除了减少整体的文件尺寸和I/O操作次数，也提高了类的查找速度。



由下图可以看到，jar和apk文件的组成结构，以及class文件和dex文件的差异。dex格式文件使用共享的、特定类型的常量池机制来节省内存。常量池存储类中的所有字面常量，它包括字符串常量、字段常量等值。

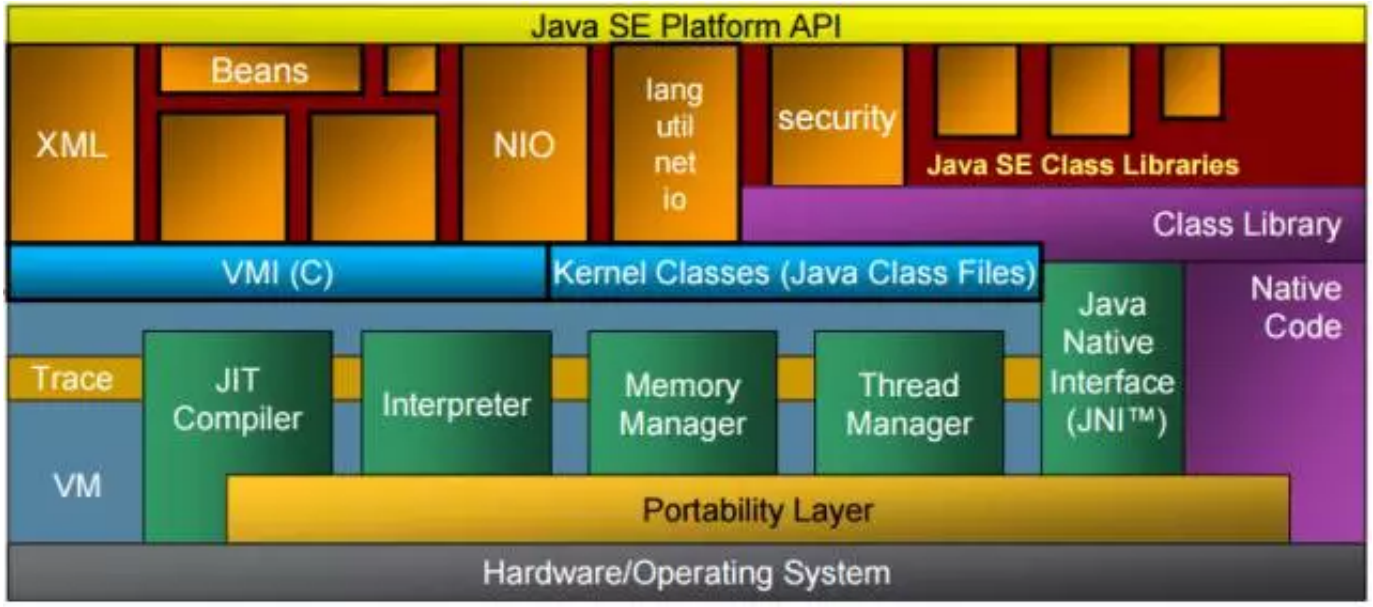


总的来说，Dalvik虚拟机具有以下特点：

- 使用dex格式的字节码，不兼容Java字节码格式
- 代码密度小，运行效率高，节省资源
- 常量池只使用32位的索引
- 有内存限制
- 默认栈大小是12KB（3个页，每页4KB）
- 堆默认启动大小为2MB，默认最大值为16MB
- 堆支持的最小启动大小为1MB，支持的最大值为1024MB
- 堆和栈参数可以通过-Xms和-Xmx修改

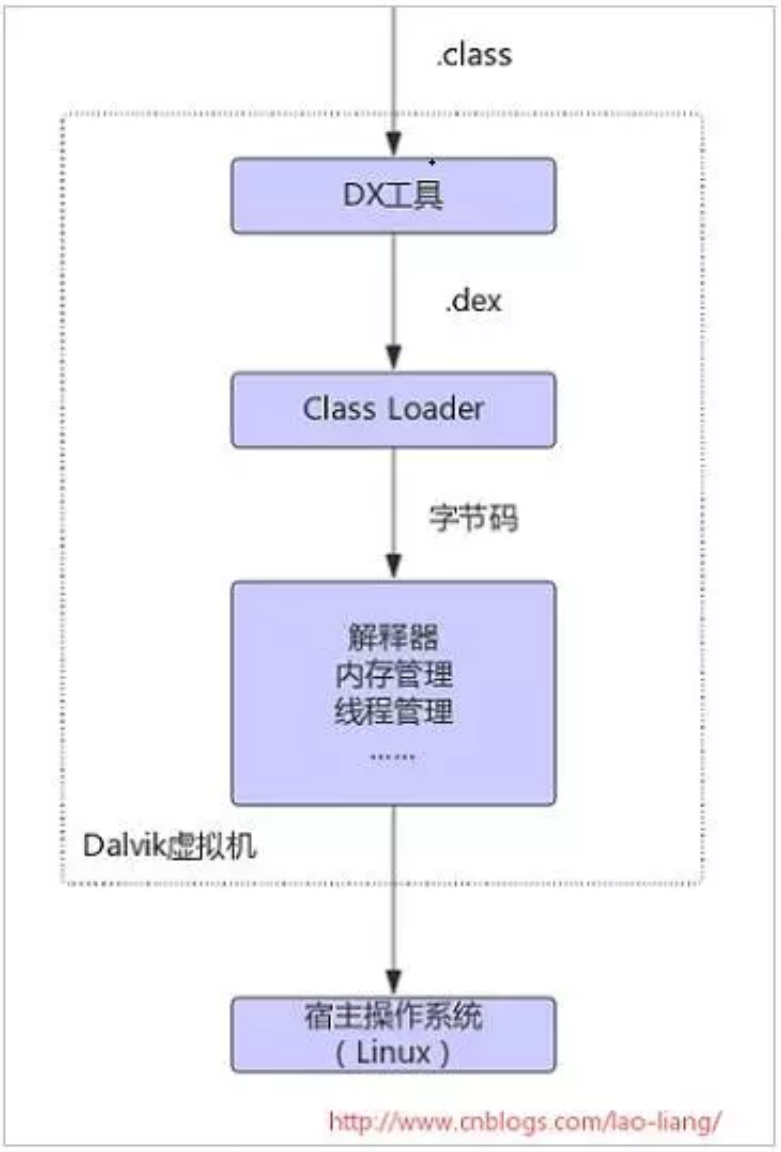
4 Dalvik系统结构

实际上，Dalvik是基于Apache Harmony（Apache软件基金会的Java SE项目）的部分实现，提供了自己的一套库，即上层Java应用程序编写所使用的API。



以上图示来自tech-insider。Apache Harmony大体上分为三个层：操作系统、Java虚拟机、Java类库。它的特点在于虚拟机和类库内部被高度模块化，每一个模块都有一定的接口定义。操作系统层与虚拟机层之间的接口由Portability Layer定义，它封装了不同操作系统的差异，为虚拟机和类库的本地代码提供了一套统一的API访问底层系统调用。虚拟机与类库之间的接口除了Java规范定义的JNI、JVMITI外，还加入了一层虚拟机接口，由内核类和本地代码组成。实现了虚拟机接口的虚拟机都可以使用Harmony的类库实现，并且可以被Harmony提供的同一个Java启动程序启动。

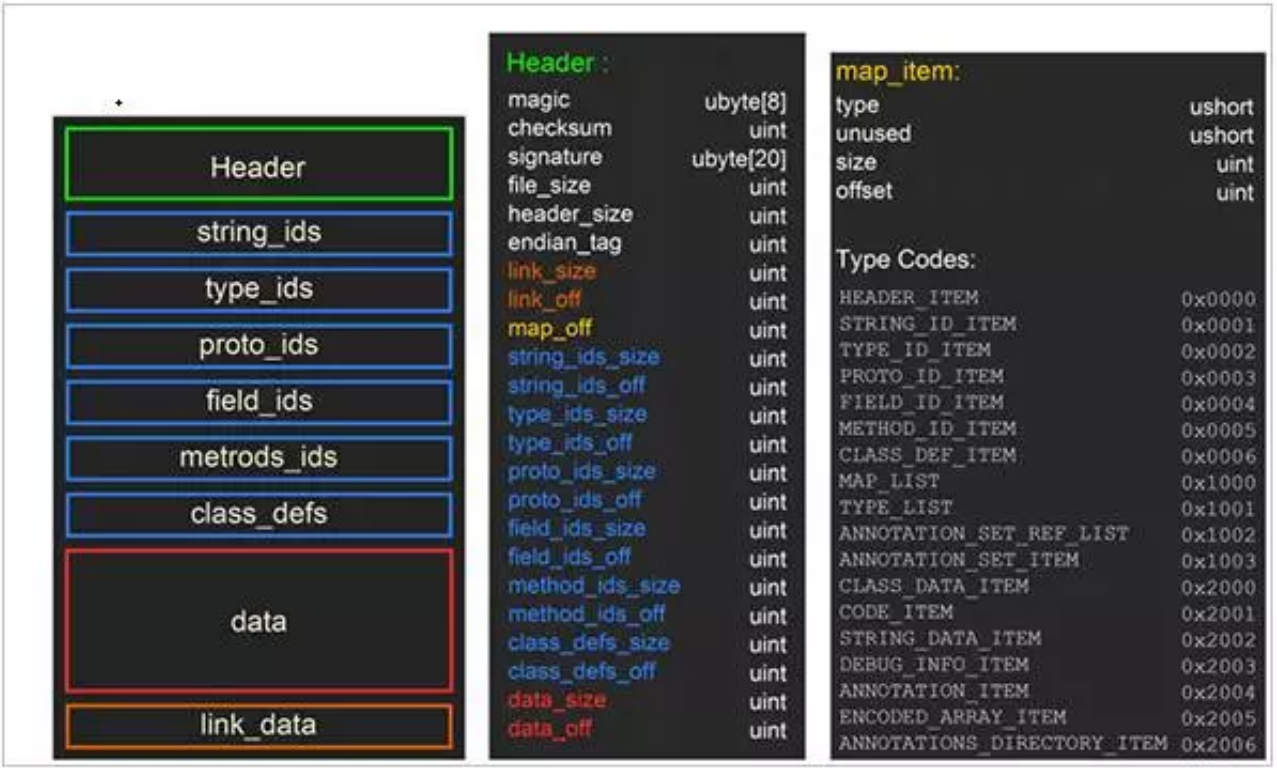
下面是Dalvik虚拟机的结构图：



一个应用首先经过DX工具将class文件转换成Dalvik虚拟机可以执行的dex文件，然后由类加载器加载原生类和Java类，接着由解释器根据指令集对Dalvik字节码进行解释、执行。最后，根据dvm_arch参数选择编译的目标机体系结构。

4.1 dex文件结构

dex文件结构和class文件结构差异的地方很多，但从携带的信息上看，dex和class文件是一致的。



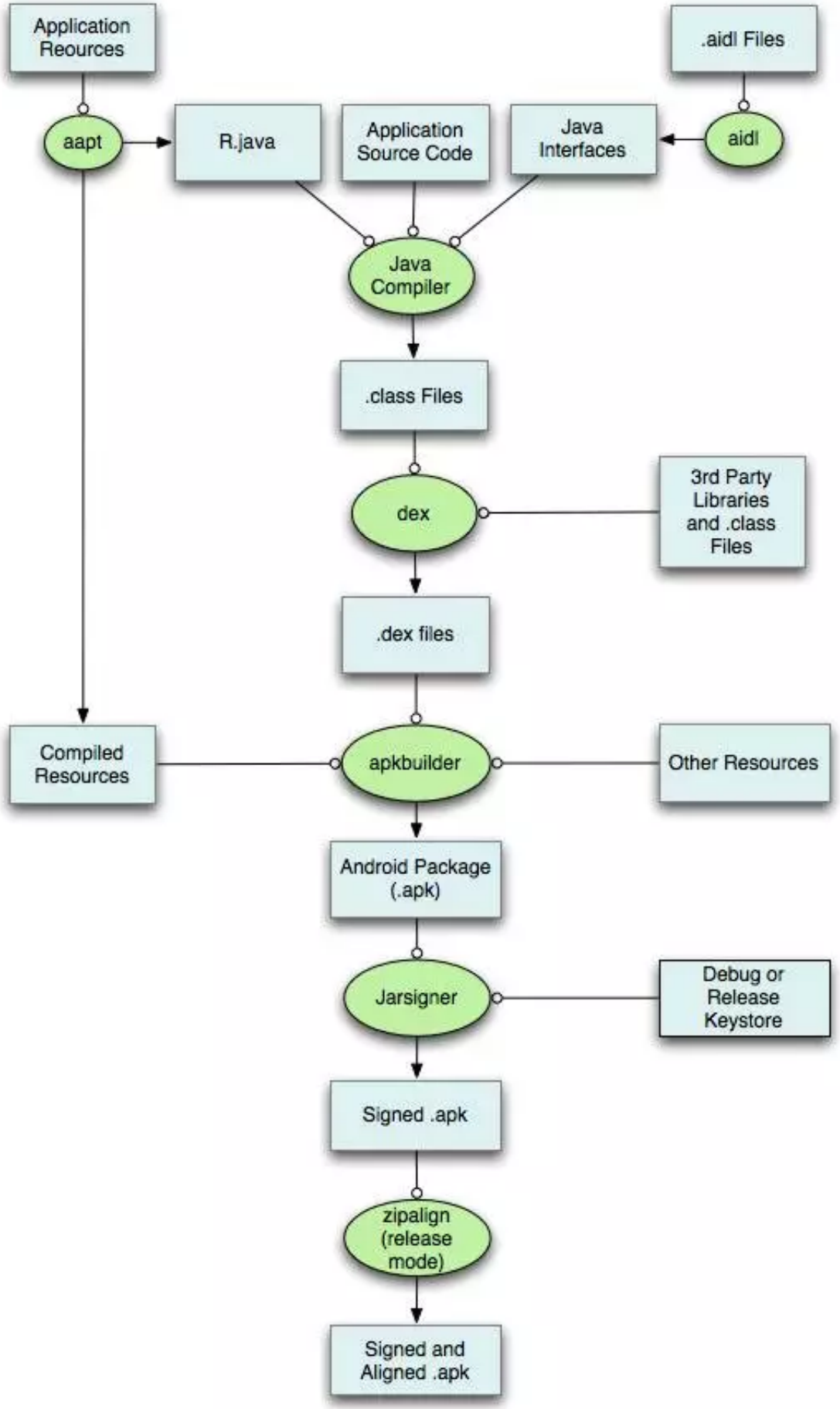
- header：存储了各个数据类型的起始地址、偏移量等信息。
- proto_ids：描述函数原型信息，包括返回值，参数信息。比如“test:()V”
- methods_ids：函数信息，包括所属类及对应的proto信息。

更多dex格式的内容，Android安全–Dex文件格式详解 这篇文章进行了非常详细的介绍。虽然dex文件的结构很紧凑，但想要运行时的性能得到进一步提升，还需要对dex文件进行进一步优化。优化主要针对以下几个方面：

- 调整所有字段的字节序和对齐结构中的每一个域
- 验证dex文件中的所有类
- 对一些特定的类进行优化，对方法里的操作码进行优化

dex文件经过优化后文件大小会膨胀，大约增加到原来的1~4倍。对于内置应用，一般在系统编译后，便会生成优化文件（odex: Optimized dex）。一个Android应用程序，需要经过以下过程才可以在Dalvik虚拟机上运行：

- 把Java源文件编译成class文件
- 使用DX工具把class文件转换成dex文件
- 使用aapt工具把dex文件、资源文件以及AndroidManifest.xml文件（二进制格式）组合成APK
- 将APK安装到Android设备运行



上图（来自网络）详尽地展示了最终签名后的APK是怎么来的。

4.2 Dalvik类加载器

一个dex文件需要类加载器加载原生类和Java类，然后通过解释器根据指令集对Dalvik字节码进行解释和执行。Dalvik类加载器使用mmap函数，将dex文件映射到内存中，通过普通的内存读取操作即可访问dex文件，然后解析dex文件内容并加载其中的类到哈希表中。

4.2.1 解析dex

总的来说，dex文件可以抽象为三个部分：头部、索引、数据。通过头部可以知道索引的位置和数目，以及数据区的起始位置。将dex文件映射到内存后，Dalvik会调用dexFileParse函数对其进行分析，分析的结果放到DexFile数据结构中。DexFile中的baseAddr指向映射区的起始位置，pClassDefs指向class索引的起始位置。为了加快class的查找速度，还创建一个哈希表，对class名字进行哈希并生成索引。

4.2.2 加载class

解析工作完成后就进行class的加载，加载的类需要用ClassObject数据结构来存储。

```
typedef struct Object {
    ClassObject* clazz;  // 类型对象
    Lock lock;           // 锁对象
} Object;
```

其中clazz指向ClassObject对象，还包含一个Lock对象。如果其它线程想要获取它的锁，只有等这个线程释放。Dalvik每加载一个class都会对应一个ClassObject对象，加载过程会在内存中分配几个区域，分别存放directMethod, virtualMethod, sfield, ifield。这些信息从dex文件的数据区中读取。字段Field的定义如下：

```
struct Field {
    ClassObject* clazz;   //所属类型
    const char* name;     // 变量名称
    const char* signature; // 如“Landroid/os/Debug;”
    u4 accessFlags;       // 访问标记

    #ifdef PROFILE_FIELD_ACCESS
        u4 gets;
        u4 puts;
    #endif
};
```

待得到class索引后，实际的加载由loadClassFromDex来完成。首先它会读取class的具体数据，分别加载directMethod, virtualMethod, ifield和sfield，然后为ClassObject数据结构分配内存，并读取dex文件的相关信息。加载完成后，将加载的class通过dvmAddClassToHash函数放入哈希表，以方便下次查找；最后，通过dvmLinkClass查找该类的超类，如果有接口类则加载相应的接口类。

4.3 Dalvik解释器

对于任何虚拟机来说，解释器无疑是核心的部分，所有的Java字节码都经过解释器解释执行。由于Dalvik解释器的效率很重要，Android分别实现了C语言版和各种汇编语言版的解释器。解释器通常是循环执行，需要一个入口函数调用处理程序执行第一条指令，而后每条指令执行时引出下一条指令，通过函数指针调用处理程序。

4.4 内存管理

垃圾收集是Dalvik虚拟机内存管理的核心。此处只介绍Dalvik虚拟机的垃圾收集功能。垃圾收集的性能在很大程度上影响了一个Java程序内存使用的效率。Dalvik虚拟机使用常用的Mark-Sweep算法，该算法分Mark阶段（标记出活动对象）、Sweep阶段（回收垃圾内存）和可选的Compact阶段（减少堆中的碎片）。Android内存管理原理 这篇文章讲解得很详细。

垃圾收集的第一步是标记出活动对象，因为没有办法识别那些不可访问的对象，这样所有未被标记的对象就是可以回收的垃圾。当进行垃圾收集时，需要停止Dalvik虚拟机的运行（除垃圾收集外），因此垃圾收集又被称作STW（stop-the-world）。Dalvik虚拟机在运行过程中要维护一些状态信息，这些信息包括：每个线程所保存的寄存器、Java类中的静态字段、局部和全局的JNI引用，JVM中的所有函数调用会对应一个相应C的栈帧。每一个栈帧里可能包含对对象的引用，比如包含对象引用的局部变量和参数。所有这些引用信息被加入到一个根集合中，然后从根集合开始，递归查找可以从根集合出发访问的对象。因此，Mark过程又叫做追踪，追踪所有可被访问的对象。

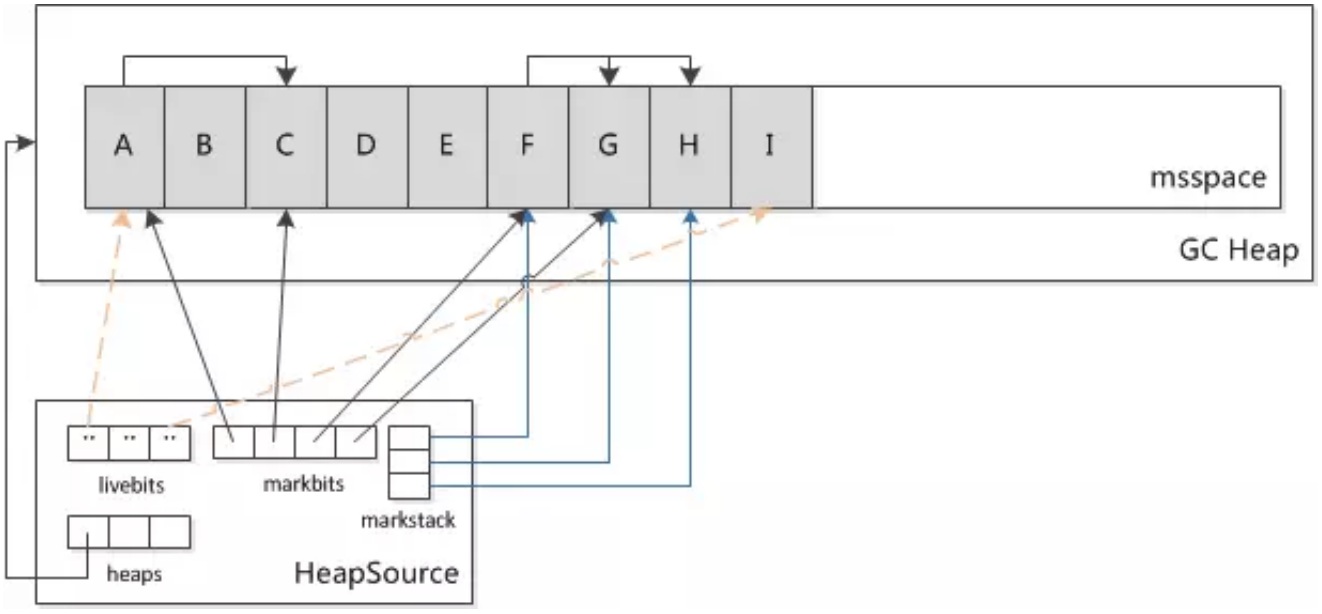
垃圾收集的第二步就是回收内存。在Mark阶段通过markBits位图可以得到所有可访问的对象集合，而liveBits位图表示所有已经分配的对象集合。通过比较liveBits位图和markBits位图的差异就是所有可回收的对象集合。Sweep阶段调用free来释放这些内存给堆。

在底层内存实现上，Android系统使用的是msspace，这是一个轻量级的malloc实现。除了创建和初始化用于存储普通Java对象的内存堆，Android还创建三个额外的内存堆：

- “livebits”（用来存放堆上内存被占用情况的位图索引）
- “markbits”（在GC时用于标注存活对象的位图索引）

- “markstack”（在GC中遍历存活对象引用的标注栈）

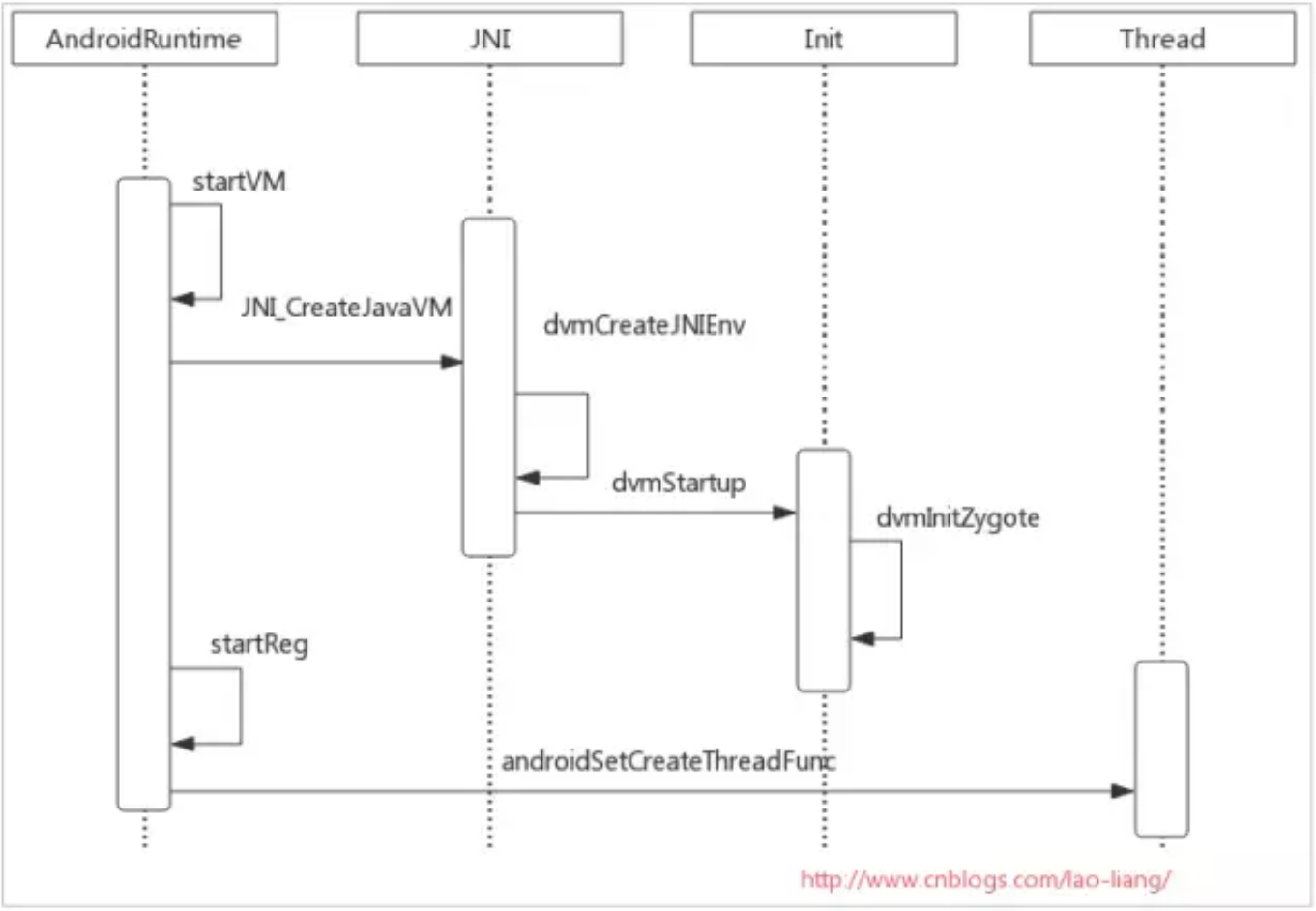
虚拟机通过一个名为gHs的全局HeapSource变量来操控GC内存堆，而HeapSource里通过heaps数组可以管理多个堆（Heap），以满足动态调整GC内存堆大小的要求。另外HeapSource里还维护一个名为”livebits”的位图索引，以跟踪各个堆（Heap）的内存使用情况。剩下两个数据结构”markstack”和”markbits”都是用在垃圾回收阶段。



上图中”livebits”维护堆上已用的内存信息，而”markbits”这个位图索引则指向存活的对象。A、C、F、G、H对象需要保留，因此”markbits”分别指向他们（最后的H对象尚在标注过程中，因此没有指针指向它）。而”markstack”就是在标注过程中跟踪当前需要处理的对象要用到的标志栈，此时其保存了正在处理的对象F、G和H。

4.5 Dalvik的启动流程

Dalvik进程管理是依赖于linux的进程体系结构的，如要为应用程序创建一个进程，它会使用linux的fork机制来复制一个进程。Zygote是一个虚拟机进程，同时也是一个虚拟机实例的孵化器，它通过init进程启动。之前的文章有对此过程有详细介绍：Android系统启动分析(Init->Zygote->SystemServer->Home activity)。此处分析Dalvik虚拟机启动的相关过程。



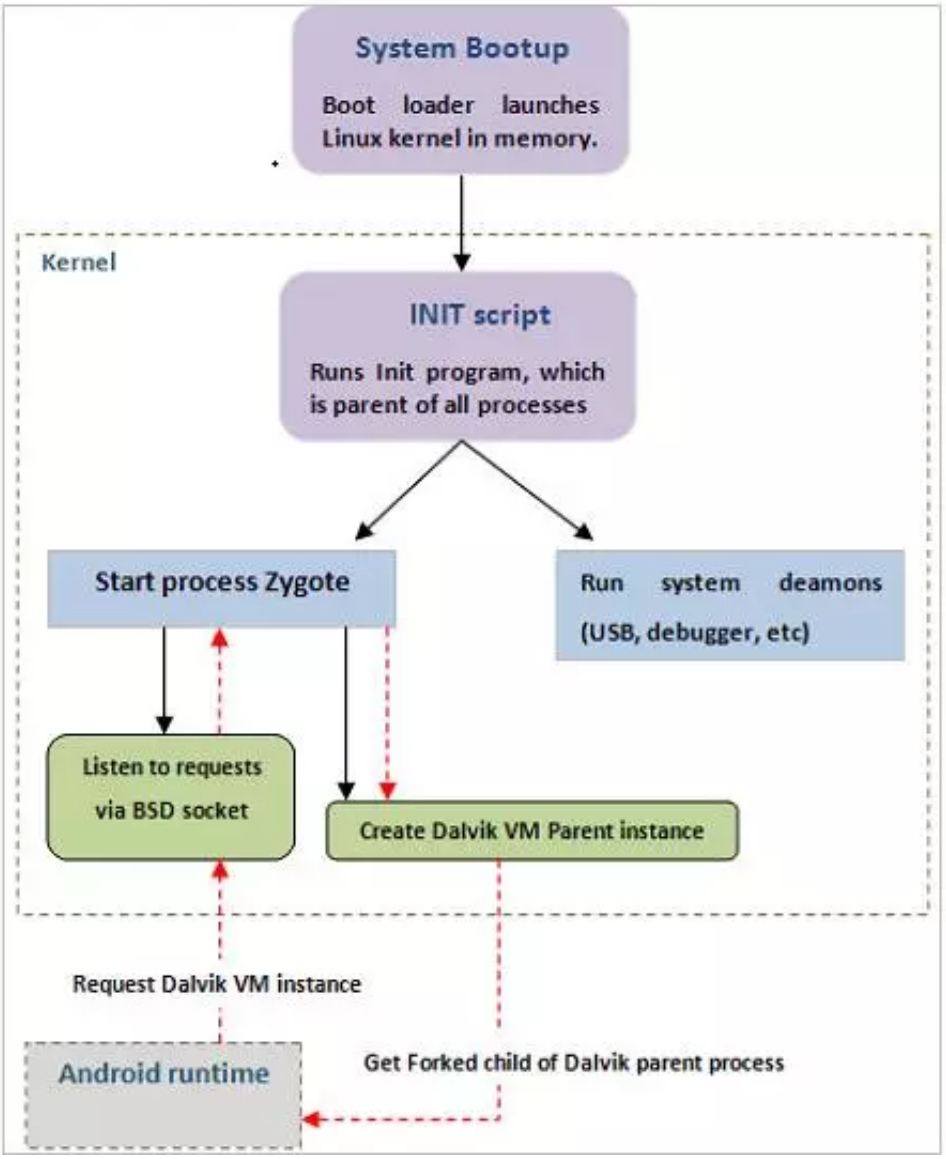
AndroidRuntime类主要做了以下几件事情：

- 调用startVM创建一个Dalvik虚拟机，JNI_CreateJavaVM真正创建并初始化虚拟机实例
- 调用startReg注册Android核心类的JNI方法
- 通过Zygote进程进入Java层

在JNI中，dvmCreateJNIEnv为当前线程创建和初始化一个JNI环境，即一个JNIEnvExt对象。最后调用dvmStartup来初始化前面创建的Dalvik虚拟机实例。函数dvmInitZygote调用了系统的setpgid来设置当前进程，即Zygote进程的进程组ID。这一步完成后，Dalvik虚拟机的创建和初始化工作就完成了。

5 Android的启动

- 启动电源，加载引导程序到RAM
- BootLoader引导
- Linux Kernel启动
- Init进程创建
- Init fork出Zygote进程，Zygote进程创建虚拟机；创建系统服务
- Android Home Launcher启动



参考：

《Android技术内幕》

Dalvik虚拟机简要介绍和学习计划
(http://blog.csdn.net/luoshengyang/article/details/8852432)

深入理解Android（二）：Java虚拟机Dalvik
(http://www.infoq.com/cn/articles/android-in-depth-dalvik)

dalvik虚拟内存管理之二——垃圾收集
(http://www.miui.com/thread-75028-1-1.html)

Dalvik虚拟机的启动过程分析
(http://blog.csdn.net/luoshengyang/article/details/8885792)