Art 上应用启动快，运行快，但是耗费更多存储空间，安装时间长，总的来说 ART 的功效就是"空间换时间"。

ART：Ahead of Time
Dalvik：Just in Time

什么是 Dalvik：Dalvik 是 Google 公司自己设计用于 Android 平台的 Java 虚拟机。Dalvik 虚拟机是 Google 等厂商合作开发的 Android 移动设备平台的核心组成部分之一，它可以支持已转换为.dex(即 Dalvik Executable)格式的 Java 应用程序的运行，.dex 格式是专为 Dalvik 应用设计的一种压缩格式，适合内存和处理器速度有限的系统。Dalvik 经过优化，允许在有限的内存中同时运行多个虚拟机的实例，并且每一个 Dalvik 应用作为独立的 Linux 进程执行。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。

什么是 ART：Android 操作系统已经成熟，Google 的 Android 团队开始将注意力转向一些底层组件，其中之一是负责应用程序运行的 Dalvik 运行时。Google 开发者已经花了两年时间开发更快执行效率更高更省电的替代 ART 运行时。ART 代表 Android Runtime,其处理应用程序执行的方式完全不同于 Dalvik，Dalvik 是依靠一个 Just-In-Time(JIT)编译器去解释字节码。开发者编译后的应用代码需要通过一个解释器在用户的设备上运行，这一机制并不高效，但让应用能更容易在不同硬件和架构上运行。ART 则完全改变了这套做法，在应用安装的时候就预编译字节码到机器语言，这一机制叫 Ahead-Of-Time(AOT)编译。在移除解释代码这一过程后，应用程序执行将更有效率，启动更快。

ART 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

ART 缺点：
1. 更大的存储空间占用，可能会增加 10%-20%
2. 更长的应用安装时间

主要如下:
1、Ahead-of-time (AOT) compilation instead of Just-in-time (JIT)
2、Improved garbage collection
3、Improved memory usage and reduce fragmentation
Ahead-of-time (AOT) compilation instead of Just-in-time (JIT)

在 Dalvik 中(实际为 android2.2 以上引入的技术)，如同其他大多数 JVM 一样,都采用的是 JIT 来做及时翻译(动态翻译)，将 dex 或 odex 中并排的 dalvik code(或者叫 smali 指令集)运行态翻译成 native code 去执行，jit 的引入使得 dalvik 提升了 3~6 倍的性能
而在 art 中，完全抛弃了 dalvik 的 jit，使用了 aot 直接在安装时用 dex2oat 将其完全翻译成 native code。这一技术的引入，使得虚拟机执行指令的速度又一重大提升

Improved garbage collection

首先介绍下 dalvik 的 gc 的过程.主要有有四个过程：
1、当 gc 被触发时候，其会去查找所有活动的对象，这个时候整个程序与虚拟机内部的所有线程就会挂起，这样目的是在较少的堆栈里找到所引用的对象。需要注意的是这个回收动作是和应用程序同时执行(非并发)
2、gc 对符合条件的对象进行标记
3、gc 对标记的对象进行回收
4、恢复所有线程的执行现场继续运行

dalvik 这么做的好处是，当 pause 了之后，gc 势必是相当快速的。但是如果出现 gc 频繁并且内存吃紧势必会导致 ui 卡顿、掉帧、操作不流畅等

后来 art 改善了这种 GC 方式(也是想对 ui 流畅度做贡献，当然关于 ui 流畅，5.0 以上了新的并行 ui 线程)，主要的改善点在将其非并发过程改变成了部分并发。还有就是对内存的重新分配管理

当 art GC 发生时：
1、GC 将会锁住 java 堆，扫描并进行标记
2、标记完毕释放掉 java 堆的锁，并且挂起所有线程
3、GC 对标记的对象进行回收
4、恢复所有线程的执行现场继续运行
5、重复 2-4 直到结束
可以看出整个过程做到了部分并发使得时间缩短，据官方测试数据说 GC 效率提高 2 倍

Improved memory usage and reduce fragmentation

官方把这一点合并到了 Improved garbage collection 这个主题中讲，原因也是和 GC 有很大关系可以对比一下两个虚拟机的内存分配的规则

首先是 Dalvik 他的内存管理特点是：
内存碎片化严重，当然这也是 Mark and Sweep 算法带来的弊端该算法如图

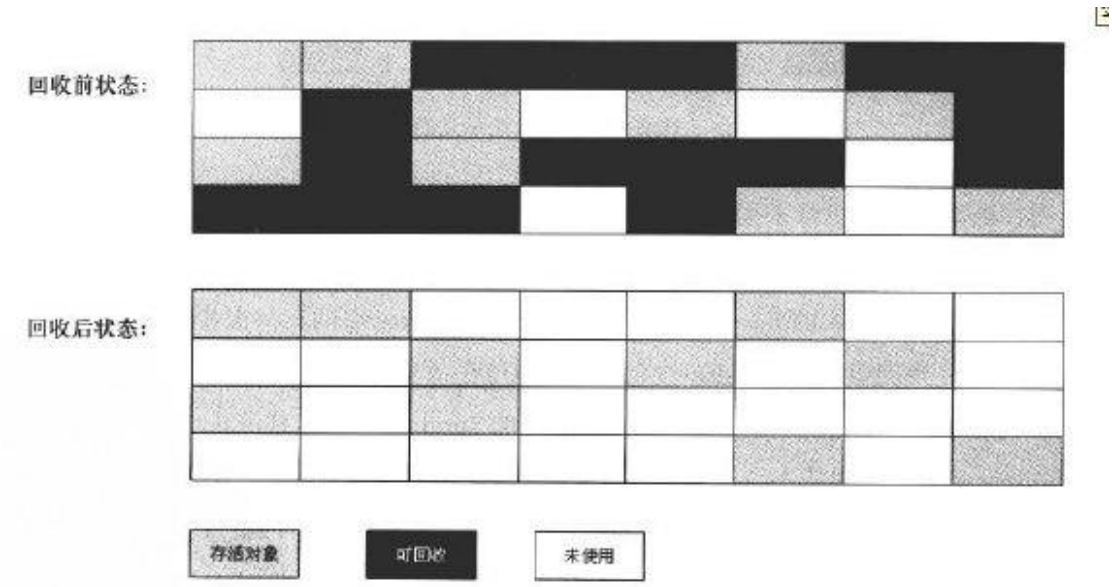

图 3-2 “标记－清除”算法示意图

可以看出每次 GC 后内存千疮百孔，本来连续分配的内存块变得碎片化严重，之后再分配进入的对象再进行内存寻址变得困难

art 的解决：在 art 中，它将 java 分了一块空间命名为 Large-Object-Space，这块内存空间的引入用来专门存放 large object。同时 art 又引入了 moving collector 的技术，即将不连续的物理内存块进行对齐，对齐了后内存碎片化就得到了很好的解决。

Large-Object-Space 的引入一是因为 moving collector 对大块内存的位移时间成本太高，而且提高内存的利用率根官方统计，art 的内存利用率提高 10 倍了左右

要弄清 ART 和 Dalvik 的区别 ，首先要清楚 apk 文件的构成

apk 包中除了一堆资源，还有一个重要文件 classes.dex，此文件由 java 字节码优化打包而成，在 Dalvik 中，每次打开应用的时候，Dalvik 会读取这个 classes.dex 并解释执行（实际情形，还可能先被转换成 odex 文件，在此忽略）；而在 ART 环境下，当你安装 apk 的时候，这个 classes.dex 文件就会被转换成本地机器码——后缀为 oat 的文件，以后打开应用时直接读取 oat 文件执行即可。举一个不是很恰当的例子：目前国际通行的论文语言都是英文（相当于 java 字节码），而你作为一个中国人只会中文（相当于 arm 架构的 Android 手机），而另一个日本人只会日语（相当于 x86 架构的 Android 手机），现在给你一堆英文资料，Dalvik 的做法就是将这些资料整理归档，等你需要查找资料时再从书架上找出需要的部分，然后翻译为中文阅读；而 ART 的做法就是先将所有资料全部翻译为中文单独保存，等你需要时直接找中文资料就可以了。所以，ART 的优点就是大大提高了执行效率，但是缺点也很明显，一是安装耗时更长，一是占用更多磁盘空间。

平台支持差别：

Dalvik Android 4.4 及其以下平台使用的虚拟机

ART Android4.4 以上平台使用的虚拟机技术

工作原理差别：
在应用程序启动时，JIT 通过进行连续的性能分析来优化程序代码的执行，在程序运行的过程中，Dalvik 虚拟机在不断的进行将字节码编译成机器码的工作。

ART 引入了 AOT 这种预编译技术，在应用程序安装的过程中，ART 就已经将所有的字节码重新编译成了机器码。应用程序运行过程中无需进行实时的编译工作，只需要进行直接调用. 因此，ART 极大的提高了应用程序的运行效率，同时也减少了手机的电量消耗，提高了移动设备的续航能力，在垃圾回收等机制上也有了较大的提升。

相对于 Dalvik 虚拟机模式，ART 模式下 Android 应用程序的安装需要消耗更多的时间，同时也会占用更大的储存空间（指内部储存，用于储存编译后的代码）,但节省了很多 Dalvik 虚拟机用于实时编译的时间