Android内存管理机制研究

宋平波，李云，杨豪杰

（中国电信股份有限公司广东研究院，广东 广州 510630）

【摘　要】　本文深入分析了Android操作系统的内存管理机制以及内存回收机制，并针对Android应用开发者提出了有效可实施的建议。

【关键词】　Android　内存管理　内存回收

doi:10.3969/j.issn.1006-1010.2015.07.020　中图分类号：TP311.1　文献标识码：A　文章编号：1006-1010(2015)07-0092-05

引用格式：宋平波,李云,杨豪杰. Android内存管理机制研究[J]. 移动通信, 2015,39(7): 92-96.

Research on Android Memory Management

SONG Ping-bo, LI Yun, YANG Hao-jie

(Guangdong Research Institute of China Telecom Co., Ltd., Guangzhou 510630, China)

[Abstract]　In this paper, the memory management and memory deallocation schemes of Android operating system were analyzed in depth. Then, effective suggestions targeted at Android application developers were put forward.

[Key words]　Android　memory management　memory deallocation

1　引言

伴随国内4G的启动以及各大运营商移动互联网流量经营的不断深入，移动互联网应用将再次迎来飞速增长，移动APP将进一步渗入用户的基本生活，中国电信也在移动互联网应用方面做了大量研发工作。鉴于移动终端操作系统在资源分配、管理以及回收上对应用有着更为严格的限制，为了能够使移动APP高效稳定的运行，APP开发人员有必要深入了解移动终端操作系统的内存管理机制，并根据操作系统的限制，合理使用内存空间，避免被操作系统强杀，提高APP的可靠性。

2　Android内存管理机制

2.1　Android与Linux

Android系统是Google在Linux内核基础上开发，针对移动设备进行了特殊优化的开源操作系统。特别是在内存管理机制上，针对移动终端资源特点及移动APP应用特点进行了优化，当进程活动停止后，Android不会立刻结束这个进程，而是将进程保留在内存中，当用户尝试再次激活此进程时，可以直接复用内存中的数据从而提升进程的启动速度，当系统需要更多内存时，Android才会将这些进程占用的内存释放。

同样，继承自Linux的Android操作系统也可以通过系统命令“cat/proc/meminfo”查看内存使用情况，如图1所示。其中，MemTotal：总的可用内存；MemFree：未被分配空闲的内存；Buffers：buffer的大小；Cached：cache的大小；SwapCached：swap缓存的大小，Android很少使用swap，经常为0。

2.2　Android进程回收（LMK）

（1）进程回收优先级

1）进程优先级及oom_adj

Android是一个多任务系统，即可以同时运行多个程序。操作系统启动运行一个程序是需要一定系统

收稿日期：2014-08-29

责任编辑：刘文竹　liuwenzhu@mbcom.cn

```
shell@android:/ $ cat /proc/meminfo
cat /proc/meminfo
MemTotal:         795932 kB
MemFree:           36868 kB
Buffers:           29796 kB
Cached:           217392 kB
SwapCached:            0 kB
Active:           460508 kB
Inactive:         118688 kB
Active(anon):     336712 kB
Inactive(anon):      308 kB
Active(file):     123796 kB
Inactive(file):   118380 kB
Unevictable:        4644 kB
Mlocked:            3328 kB
HighTotal:        418816 kB
HighFree:           6472 kB
LowTotal:         377116 kB
LowFree:           30396 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                 8 kB
Writeback:             0 kB
AnonPages:        336660 kB
Mapped:           208220 kB
Shmem:               368 kB
Slab:              32276 kB
SReclaimable:      15420 kB
SUnreclaim:        16856 kB
KernelStack:        7144 kB
PageTables:        11280 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:      397964 kB
```

图1　Android手机内存使用情况

开销（CPU、时间等）的。因此，如在2.1节中提到，为了应用再次启动时的启动速度，当用户退出一个程序后，Android操作系统仍然会将这个程序保留在内存中，当程序再次被启动时就可以快速启动。当系统内存出现不足时，Android系统会按照一定的优先级，对运行中的程序进行回收，以释放内存。

如图2所示，Android操作系统将运行中的进程分为6类，并在内存不足时按照空进程（EMPTY_APP）、内容供应节点（CONTENT_PROVIDER）、后台进程（HIDDEN_APP）、次要服务进程（SECONDARY_SERVER）、可视进程（VISIBLE_APP）和前台进程（FOREGROUND_APP）的顺序，逐一对进程进行回收。

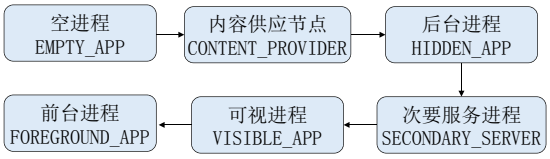

图2　Android进程分类

Android操作系统为每个进程维护一个oom_adj值，一进程的oom_adj值也就代表了它的优先级，oom_adj值越高代表该进程优先级越低。各进程含义及oom_adj值如下：

◆前台进程（FOREGROUND）oom_adj:0

用户在屏幕上可见的应用程序进程，同时也包括（可能不可见的）系统进程或电话服务进程。最为直接的例子就是用户正在使用的应用程序进程就是前台进程，而前台进程的优先级最高，也最不容易被系统回收。

◆可见进程（VISIBLE）oom_adj:1

应用程序仍然可见，但是已经不是前台进程的应用进程。比如：当某个前台应用程序被另外一个应用程序的通知遮盖了一部分（通知通常在屏幕正中，但周边仍然能见到原来的应用程序），这时此应用程序进程由前台进程转为了可见进程。可见进程的优先级仍然较高，仅次于前台进程。

◆次要服务（SECONDARY SERVER）进程oom_adj:2

次要服务指的是系统后台正在运行的服务进程（包括系统桌面、UI等服务，但电话、系统底层服务等主要服务不在此列，因此命名为次要服务）。后台运行（不可见）的包含服务的应用进程是次要服务的主要组成部分，次要服务的优先级低于可见进程。

◆后台进程（HIDDEN）oom_adj:7

后台进程指的是不可见的且不包含服务，但仍在运行的应用进程。当用户返回桌面时，之前用户正在使用的应用程序如果不包含服务，则此应用程序变为后台进程。

◆内容供应节点（CONTENT PROVIDER）oom_adj:14

内容供应节点指为其他应用提供数据内容的进程，联系人、日历等进程属于内容供应节点。

◆空进程（EMPTY）oom_adj:15

空进程指在后台处于挂起状态的应用进程，没有任何代码在执行。空进程的优先级最低，在LMK工作机制中，属于最先被回收的类别。

2）进程回收时机

除了上述进程优先级之外，Android系统还维护着另外一张阀值对应表（终端厂家根据每个型号的手机按照配置和Android版本不同，对具体的阀值做调整），如表1所示（以Google Nexus One为例）：

表1　oom_adj内存阀值表

oom_adj	剩余内存阀值/页（每页大小为4k）
0	1 536
1	2 048
2	4 096
7	5 120
14	5 632
15	6 144

这个表定义了一个对应关系，每一个阀值对应了一个进程优先级（oom_adj），当系统的可用内存低于某个阀值时，就回收大于该阀值对应的进程优先级（oom_adj）的进程。比如，当可用内存小于6 144*4k=24MB时，开始回收EMPTY_APP进程，直至大于24MB；同理当可用内存小于5 632*4k=22MB时，开始回收CONTENT_PROVIDER以及EMPTY_APP进程。

3）操作系统设置及修改

操作系统在init.rc文件中分别定义了各个进程的优先级以及内存阀值，如下方所示：

```
# Define the oom_adj values for the classes of processes
# that can be killed by the kernel
setprop ro.FOREGROUND_APP_ADJ 0
setprop ro.VISIBLE_APP_ADJ 1
setprop ro.SECONDARY_SERVER_ADJ 2
setprop ro.HIDDEN_APP_ADJ 7
setprop ro.CONTENT_PROVIDER_ADJ 14
setprop ro.EMPTY_APP_ADJ 15
# Define the memory thresholds at which the above process
# classes will be killed. In pages (4k)
setprop ro.FOREGROUND_APP_MEM 1536
setprop ro.VISIBLE_APP_MEM 2048
setprop ro.SECONDARY_SERVER_MEM 4096
setprop ro.HIDDEN_APP_MEM 5120
setprop ro.CONTENT_PROVIDER_MEM 5632
setprop ro.EMPTY_APP_MEM 6144
```

同时，也可以通过修改/sys/module/lowmemorykiller/parameters/minfree配置文件来修改各级进程回收的内存阀值，比如通过执行“echo “1836,2048,4096,5120,5360,8192”>/sys/module/lowmemorykiller/parameters/minfree”，可以将操作系统回收EMPTY_APP进程的内存阀值修改为32M。

（2）进程回收（Low Memory Killer，LMK）

在（1）小节中了解到，即使当用户退出应用程序之后，应用程序的进程也还是存在于系统中，这样是为了方便程序的再次启动，但若此，随着打开的程序数量的增加，系统的内存会变得不足，就需要杀掉一部分进程以释放内存空间，至于是否需要杀死一些进程和哪些进程需要被杀死，是通过Low Memory Killer（LMK）机制来进行判定的。

Android的Low Memory Killer（LMK）基于Linux的OOM机制，在Linux中，内存是以页面为单位进行分配的，当申请页面分配时如果内存不足会通过以下流程选择bad进程来杀掉从而释放内存。

在Low Memory Killer中通过进程的oom_adj与占用内存的大小决定要杀死的进程，oom_adj越小越不容易被杀死。参考Andoird源码中lowmemorykiller.c的lowmem_shrink函数，计算哪些进程该回收，并发送SIGKILL信号将该进程杀死。

3　Android应用内存管理

3.1　操作系统限制

绝大多数Android应用程序本质上是一个java进程，运行在Linux内核之上的dalvik（可以理解为Android的JVM）虚拟机中，每个Android应用的虚拟机均为各自独立的，因此Android又类似于Java的内存管理。

每个虚拟机所能使用的内存按照Android版本以及手机配置各有不同，可以通过下面的系统命令查看每台手机上每个应用能使用内存数量的限制：

```
getprop | grep dalvik
```

[dalvik.vm.heapgrowthlimit]: [64m]

[dalvik.vm.heapsize]: [128m]

3.2 垃圾收集GC（Garbage Collector）

Java语言引入了垃圾收集机制，程序员不需要像在C/C++中一样，手工释放不需要的对象或内存区域，Java虚拟机可以跟踪内存中的所有对象并可以发现和回收不再使用的对象。从而极大地减少了内存泄漏发生的可能性。

垃圾收集算法的核心思想是：虚拟机对内存中的对象进行监控，当对象不再被使用（引用数降为0）时，将对象标为垃圾对象，并在合适的时候将对象释放并回收其占用的内存空间。垃圾回收对于系统性能有着较高的影响，因此开发人员需要对垃圾回收的算法、回收时机以及相关系统参数做深入的了解

3.3 位图（Bitmap）资源

客户端应用为了良好的视觉体验，经常使用大量的图片进行美化，图片在内存中以位图（Bitmap）对象来存放。使用Bitmap对象，需要紧记以下2点：

（1）Bitmap对象占内存量仅与图片的长宽像素数有关，与原图的图片格式或压缩比例无关。因此，压缩JPG图片的质量，而不减少图片尺寸，是无法降低该图片占用的内存量的。

（2）根据Android SDK文档：“Bitmap data is not allocated in the VM heap. There is a reference to it in the VM heap (which is small), but the actual data is allocated in the Native heap by the underlying Skia graphics library … The Native heap is shared between running applications, so the amount of free space depends on what other applications are running and their bitmap usage …”，可知Bitmap对象的数据并非存放在VM Heap中，而是存放在Native Heap中，而Native Heap的空间是与其他应用程序共享的，因此即使应用程序申请的内存在3.1小节所述限额之内，也有可能申请不到内存并产生OutOfMemoryError错误。

4 Android应用开发建议

在第2和第3部分中，描述了Android操作系统的内存管理机制以及Android操作系统对于Android应用的各种限制，为了让应用能够更加稳定地运行在Android操作系统之上，应用开发者需要根据Android操作系统的特性，有针对性地优化自己的应用。

4.1 针对LMK机制

首先需要明确一点，没有任何办法能够保证第三方应用不被LMK机制回收。但当应用不希望被操作系统LMK机制主动回收时，建议采用一些优化机制：

（1）尽量提高进程的优先级。在应用中实现一个Service，可以将进程变为次要服务（SECONDARY SERVER），在LMK机制中拥有较高的优先级，可以降低LMK命中的几率。

（2）应用进入后台运行时，尽量在应用中保留一个Activity（界面），因为在相同优先级的情况下，所有Activity均已关闭的进程会优先被回收。

（3）依赖于其他优先级高的进程。当一个进程被其他更高优先级的进程所依赖或者为其他更高优先级进程提供服务时，系统会让此进程享有与高优先级进程同样的优先级。例如：当某个通信录应用正在向另外一个前台进程（通过CONTENT PROVIDER提供）提供通信录内容时，即使这个通信录应用没有任何UI在屏幕上显示、所有Activity也已经关闭、且CONTENT PROVIDER进程的优先级不高，操作系统仍然不会回收这个通信录应用，因为此时通信录应用享有与前台进程相同的优先级。

4.2 针对GC机制

根据3.2小节中对GC机制的描述，不良的代码会导致GC机制对程序性能产生影响，并有可能导致内存不足等。因此程序员有必要在了解GC机制的基础上，针对GC特点进行编码，具体包括：

（1）尽量不使用静态变量

作为全局变量，静态变量会一直占用内存，不会被GC回收，有可能导致应用内存不足。

（2）尽快释放不使用的对象

尽快将不使用的对象设为null，可以使得该对象尽快被GC回收，提高了GC的效率。

（3）避免集中创建或者释放大量的对象

在短时间内大量创建对象，或者大量申请内存空间，会导致JVM进行GC以回收内存来应对应用的需

求。而大量的GC请求会使得应用运行迟缓，导致应用用户体验下降。

（4）采用StringBuilder处理字符串累加

String是固定长的字符串对象，因此累加String对象会导致String对象的频繁创建，尽量采用StringBuilder来处理字符串累加。

（5）尽量使用基本类型变量

基本类型变量比对象变量占用内存更小，因此应该尽量使用基本类型变量。

4.3 针对位图（Bitmap）资源

由3.3小节可知，Android中存储Bitmap数据的内存区域是有限的，而且需要与其他应用程序共享，即使申请的内存总量仍在虚拟机的限额之内，也可能因为其他应用产生了过多Bitmap数据而导致内存不足，因此小心地使用Bitmap对象，不仅可以使自己的应用程序更加稳定，也会使得在Android运行环境上执行的其他应用程序更加稳定。

（1）及时回收

当应用不再使用某个Bitmap对象时，尽量不要等待系统自动释放，应该尽快调用“recycle()”释放Bitmap占用的内存空间。

（2）捕获异常

在产生Bitmap对象的地方捕捉OutOfMemoryError错误，能够有效地减少因为内存不足而造成的应用程序崩溃。

（3）同样的图片避免生成多个Bitmap对象

很多情况下，可能需要在一个Activity里多次用到同一张图片（如好友列表中的默认头像等）。此时应该尽量使用同一个Bitmap对象，不应新建多个Bitmap对象，避免内存的浪费。

（4）尽量避免使用大图片

Bitmap对象占用内存的空间=图片长边像素数*图片宽边像素数*颜色深度（也就是每位像素占用的内存数）。可见Bitmap对象占用的内存空间与图片的像素成正比，当图片太大时，很容易引发内存不足。根据图片显示区域的大小，使用BitmapFactory.Options设置inSampleSize缩小图片，可以有效地减少大图片对内存的影响。

5 结束语

移动互联网正在飞速发展，Android设备正在飞速普及，本文有针对性地介绍了Android操作系统内存管理方面的机制，并针对Android应用开发者提出了有效可实施的建议。

参考文献：

[1] Google Inc. Android源代码[EB/OL]. [2014-08-27]. http://code.google.com/p/android/.

[2] Google Inc. Android SDK文档[EB/OL]. [2014-08-27]. http://developer.android.com.

[3] 池炜成. Java垃圾收集的机制及调优[J]. 计算机应用研究, 2004(3): 144-148.

[4] 柯元旦. Android内核剖析[M]. 北京: 电子工业出版社, 2011.

[5] Cay S Horstmann, Gray Cornell. Java核心技术[M]. 周立新,陈波,叶乃文,等,译. 北京: 机械工业出版社, 2014. ★

作者简介

宋平波：硕士毕业于英国纽卡斯尔大学通信及信号处理专业，现任职于中国电信股份有限公司广东研究院，主要研究方向为移动互联网应用、电子商务等电信级平台开发。

李云：学士毕业于哈尔滨工业大学电子与通信工程系，现任职于中国电信股份有限公司广东研究院，主要研究方向为移动互联网应用、搜索引擎、电子商务等电信级平台开发。

杨豪杰：硕士毕业于中山大学软件学院，现任职于中国电信股份有限公司广东研究院，主要研究方向为移动互联网应用平台开发。