android内存检查工具

Last edited by **caoquanli** 1 month ago

Android内存检查工具

在Android生命中，内存占据很大的一部分，内存是处理一切活动的基础，一个简单并且强大的内存检查工具能让我们清晰的了解当前运行程序即应用的内存使用情况，对我们俩了解以及认识应用有很大的帮助。 在当前的开发中，有很多的内存检查工具，下面我一一列出；

1 top/procrank

使用上面的命令必须知道这几个概念

VSS Virtual Set Size 虚集合大小

RSS Resident Set Size 常驻集合大小

PSS Proportional Set Size 比例集合大小

USS Unique Set Size 独占集合大小

一般来说内存占用大小有如下规律：VSS >= RSS >= PSS >= USS

VSS - Virtual Set Size 虚拟耗用内存（包含共享库占用的内存）是单个进程全部可访问的地址空间，是虚拟地址，它上线与进程的可访问地址有关，和当前进程的内存关系不大。

RSS - Resident Set Size 实际使用物理内存（包含共享库占用的内存）是单个进程实际占用的内存大小，对于单个共享库， 尽管无论多少个进程使用，实际该共享库只会被装入内存一次。与 PSS 差不多，包含进程共享内存，RSS没把共享内存大小平分到使用共享的进程上，所以所有进程的RSS相加会超过物理内存。

PSS - Proportional Set Size 实际使用的物理内存（比例分配共享库占用的内存），包含进程间共享的内存，而USS不包括，进程USS相加小于物理内存大小的原因。对于PSS而言，如果A进程和B进程同时共享同一个SO库，那就平分到了A和B上，但启动A进程，B没有启动，则B的PSS曲线图会有较大的阶梯状下滑。对于USS而言，它的坑在Dalvik申请内存会有GC延时及其策略，会影响曲线波动。

USS - Unique Set Size 进程独自占用的物理内存（不包含共享库占用的内存）USS 是一个非常非常有用的数字， 因为它揭示了运行一个特定进程的真实的内存增量大小。如果进程被终止， USS 就是实际被返还给系统的内存大小。是针对某个进程开始有可疑内存泄露的情况，进行检测的最佳数字。怀疑某个程序有内存泄露可以查看这个值是否一直有增加。

2 Android Develpers提供了两个方法来检测应用程序的内存使用情况

dumpsys是一种在Android设备上运行的工具，它提供有关系统服务的信息。dumpsys使用Android调试桥（ADB）从命令行调用， 以获取在连接设备上运行的所有系统服务的诊断输出。此输出通常比您想要的更详细，因此使用下面描述的命令行选项仅为您感兴趣的系统服务获取输出。此页面还描述了如何使用它dumpsys来完成常见任务，例如检查输入，RAM，电池或网络诊断。它提供了两中国方法来实现对应用内存情况的获取

(1)procstats

可以看到应用程序在一段时间内的表现，包括他的后台运行时间以及他在此期间使用的内存两，使用它可以快速的发现应用程序中的低效率和错误的行为，比如内存泄露，这可能会影响其执行方式，尤其是在低内存设备上运行，其状态转储显示u有关每个人应用程序的运行时，比例集大小化然唯一集大小 想获取过去三个小时内的应用程序内存使用情况用下面命令： adb shell dumpsys procstats --hours 3

(2)meminfo

这个可以使用以下命令记录应用程序内存在不同类型的RAM分配之间的划分： adb shell dumpsys meminfo package_name | pid [-d] -d表示的是打印有关Dalvik和ART内存使用的更多信息，这个输出列出了当前应用程序的所有当前分配，以千字节为单位。 私有（干净和脏）RAM RAM是进程使用的内存，这是应用程序进程被销毁是系统可以回收的大部分RAM。 通常，最重要的是Private Dirty RAM ，它仅供当前进程使用，其内容仅存在与RAM中，因此无法分页到存储。所有Dalvik和本机堆分配都将是私有脏RAM， 比例集大小PSS 这个是对应用程序RAM使用情况的衡量，他考虑了跨进程的共享页面，流程独有的任何页面的会直接影响其PSS止，而与其他流程共享的页面仅与共享量成比例的影响PSS止，例如在两个进程之间共享的页面将为每个进程的PSS贡献其一半大小。 PSS测量的特性是可以在所用进程中添加PSS，已确定所有进程使用的实际内存，因此PSS可以很好的衡量进程实际RAM权重，并与其他进程的RAM使用和总可用RAM进行比较。

3 DDMS（Dalvik Debug Monitor Server）

DDMS 的全称是DalvikDebug Monitor Service，是 Android 开发环境中的Dalvik虚拟机调试监控服务。提供测试设备截屏、查看特定进程正在运行的线程以及堆信息、Logcat、广播状态信息、模拟电话呼叫、模拟接收及发送SMS、虚拟地理坐标等服务。 这个是各种调试信息集合，有时延，内存，线程，cpu等各种信息展示，经常使用Heap、Allocation Tracker 和 Dump Hprof file(内存快照)等几个方法来检查内存。 其中heap是DDMS带有的一个内存监测工具，用heap监测应用进程的使用情况操作步骤如下（在Eclipse中）

1. 启动eclipse后，切换到DDMS透视图，并确认Devices视图、Heap视图都是打开的；
2. 将手机通过USB链接至电脑，链接时需要确认手机是处于“USB调试”模式，而不是作为“Mass Storage”；
3. 链接成功后，在DDMS的Devices视图中将会显示手机设备的序列号，以及设备中正在运行的部分进程信息；
4. 点击选中想要监测的进程，比如system_process进程；
5. 点击选中Devices视图界面中最上方一排图标中的“Update Heap”图标；
6. 点击Heap视图中的“Cause GC”按钮；
7. 此时在Heap视图中就会看到当前选中的进程的内存使用量的详细情况。

4 MAT

Memory Analyzer（内存分析），需要抓取Hprof文件，最简单使用ddms抓取，其次就是命令

抓取hprof命令如下(在adb shell模式下)：

am dumpheap pid outfilePath (文件名必须为hprof) 操作：(DDMS抓取的可以直接忽略转格式问题)

1. adb shell am dumpheap com.android.phone /data/local/tmp/test.hprof
2. adb pull /data/local/tmp/test.hprof c:\test.hprof
3. hprof-conv c:\test.hprof c:\testConv.hprof

5 Finder

Activity 获取dump的所有Activity对象

Top Classes 对象数量或对象大小为维度来获取对象降序列表

Compare 对比两个Hprof文件内容的差别

Bitmap 获取dump的所有bitmap对象

Same Bytes 查询dump 中以byte[]类型出现并且内容重复的对象

Singleton 查询dump中的单例

！！！ 能够准确定位80%的泄漏问题

6 LeakCanary

在onDestory 时检查弱引用，使用ReferenceQueue，白名单需要写死在AndroidExcludedRefs.java中。每次得重新编译，区分系统版本。 能识别系统泄漏，能够给出一个分析。 需要人工修复解决内存泄漏问题 开源组件HAHA 分析,返回一条GC链。

7 LeakInspector

在onDestory 时检查弱引用，使用WeakReference,提供回调方法，能增加自定义的LOG，TRACE，DUMPSYS信息，需要在白名单以XML配置的形式存放到服务器上，不区分系统版本。 可以进行预处理，在ondestroy里，通过反射自动修复系统泄漏。 可以对对整个Activity的View遍历一遍，把图片所占用的内存数据释放掉，能够减少对内存的影响采集dump后，自动通过Magnifier上传dump文件，调用MAT命令进行分析，返回GC链。 可以跟自动化测试无缝结合，自动化脚本执行过程中发现内存泄漏，收集dump发送到服务器上，分析，生成JSON结果。

8 JHat

Oracle公司开发的多人协作Hprof分析工具 使用命令：jhat test.prof 解析prof文件，开启httpSrv服务，维护解析后的数据，默认端口7000，直接访问查询。

9 libc_malloc_deBug_leak.so库

Android系统底层调用libc.so申请内存，而libc_malloc_deBug_leak库就是监视libc.so内部接口的调试库 Native Heap 就是类内存申请部分，NDK编译出来的so文件放到系统的 /data 目录中，size字段的值就是内存大小，每一个都拥有一个申请的调用栈，根据栈后的method字段值能够知道该方法的内存偏移，使用addr2line.exe转化方法名称，注需要编译选项中加入“-Wl,-Map=xxx,map -g“

10 APT

腾讯开源的测试工具，是DDMS的插件，能够实时监控多个app的cpu及其内存情况，以图表形式展示出来

11 GC Log

Dalvik GC Log

Logcat输出的log日志，分为Dalvik Log和ART Log两种 ##12 Systrace Android4.1上引入的性能分析工具，能够输出各个线程的当前函数调用状态，并且可以跟当前CPU的线程运行状态、VSYNC、SurfaceFlinger等系统信息在同一时间轴上进行对比。但不是所有手机机型都能支持。

SurfaceFlinger服务在每个VSYNC信号中断时调用一次，那APP显示非常流畅。 如果VSYNC的上升沿SurfaceFlinger服务没有调用，那会导致丢帧。 CPU负载过大，低端机型的单核机型上会有发生。在VSNYC中断信号处，CPU闲暇，没有执行其他进程任务时，那我们需要做进一步分析 应用侧没有完成绘制，应用内部处于繁忙时，查看performTraversals信息，忙于其他业务逻辑没有绘制，那看是否忙于分发响应事件。如果当前窗口视图太多，布局嵌套太深，会导致查找响应输入事件的控件耗时长，没法绘制UI。 对于绘制等问题，Google添加了一些警告：

（1） Inefficient View alpha usage(5.1+以上才有)

（2） Expensive rendering with Canvas.saveLayer() Canvas.saveLayer()会打断绘制过程中的渲染管道，替换使用View.LAYER_TYPE_HEARDWARE或者static Bitmaps,会让离屏缓存服用相邻两帧间的数据，避免渲染目标被切换而打断。

（3） Path Texture Churn

（4） Expensive Bitmap uploads Bitmaps 在硬件加速下，修改和图像的变化都会上传给GPU，如果像素总量大，会消耗GPU的较大资源，所以建议减少每帧中对图片的修改，出现调用setLayerType为LAYER_TYPE_SOFTWARE之后，此时整个屏幕变成一张图。

（5） Inflation during ListView recycling 没用ListView复用，造成inflate的单个Item的getView成本比较高

（6） Inefficient ListView recycling/rebinding 每帧的ListView recycling 耗时较长，那得看下Adapter.getView()绑定数据的时候是否存在问题

（7） Expensive Measure/Layout pass Measure / Layout 耗时导致卡顿，动画时，不要触发Layout

（8） Long View.draw() Draw 本身耗时比较长，避免在View 或 Drawable 的onDraw里面执行任务频繁自定义操作，特别时申请内存和绘制Bitmap.

（9） Blocking Garbage Collection GC导致卡顿，就是GC for Alloc的stop the world，避免在动画的时候生成对象，尽量重用Bitmap能够避免触发GC。

（10） Lock contention UI 线程锁，UI线程去使用其他线程持有的锁，检查现有UI线程锁并确认它锁住的时间长短。

(11) Scheduling delay 网络I/O、磁盘I/O 等线程资源争抢，导致有一定时间的UI线程实际耗时长，而卡顿，检查后台线程是否都云溪nag在低优先级的线程上（是否比Thread_Priority_background还低）。