System Performance

性能类型

- 资源类的性能
  - CPU
  - GPU
  - 内存
  - 磁盘I/O
  - 网络传送
  - 异常处理
- 交互类的性能
  - 响应时延（冷起，热起）
  - 流畅度

CPU

- CPU资源冗余使用
  - long -> short / String -> int (CustomTheme themeID)
- CPU资源争抢
  - 很多耗CPU进程 (开机)
  - new Threads （短信、face++）
- CPU资源利用率低
  - 功耗、发热

实例1

- 迅雷 CPU 600%
- CGroups(control groups)、 cpuset
  - bg 0-1
  - third bg 2-3
  - fg 0-6
  - top 0-7
  - dex2oat top 0-4， bg 0-1/2-3
- 后台控制
  - 3 min+high cpu+bg ->kill

实例2

- 应用商店启动慢
  - load 5屏
- 联系人列表启动慢
  - new Threads preload（drawable, layout, 其他资源）
  - lazy load

实例3

- OTA卡顿，游戏卡顿
  - 发热后响应慢
    - Thermal （Odin 1.6G 关5-6）
    - Boost over thermal

内存

毫无疑问内存一致占据着至关重要的地位，任何处于运行过程中的程序或者数据都需要依靠内存作为存储介质，数据在持久化之前暂时居住点。

- LMK（Low Memory Killer）
- GC（Garbage Collection）
  - GC_EXPLICIT (Runtime.gc VMRuntime.gc)
  - GC_FOR_[M]Alloc 没有足够的空间给即将分配的内存时触发。
  - GC_FOR_CONCURRENT 当超过堆占用阀值事会自动触发
  - GC_BEFORE_OOM
  - GC_HPROF_DUMP_HEAP
- 增大free 内存
- zram
- 应用在内存的使用上一定要克制
- 实时内存碎片整理
- 小内存合并

磁盘IO

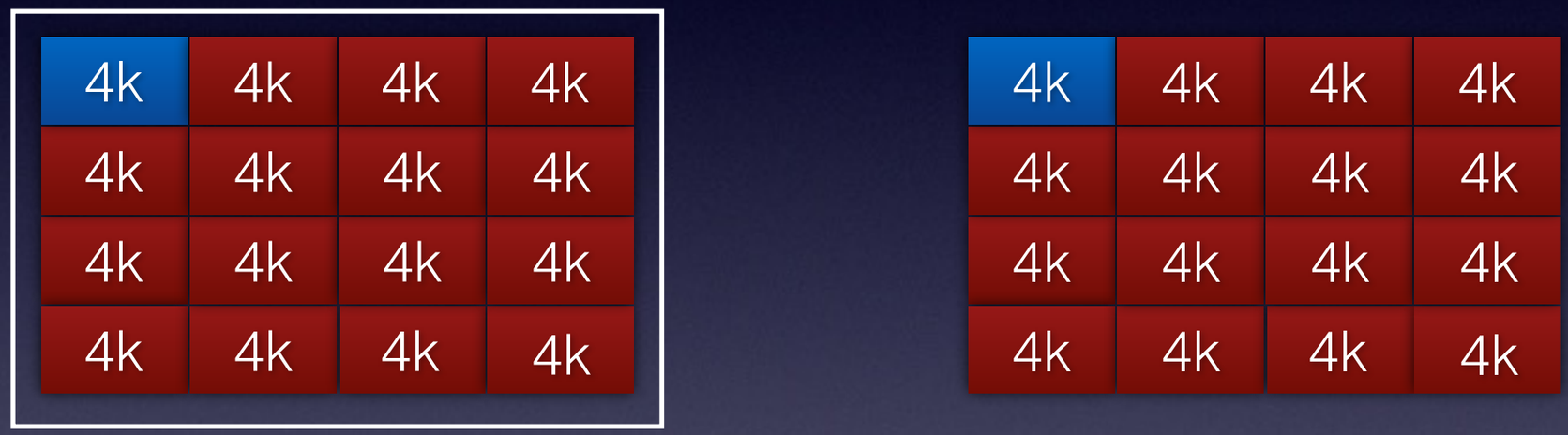
应用程序为了完成一系列工作，可能需要频繁的操作磁盘，无论是数据库还是缓存系统，都需要使用到磁盘，磁盘性能的重要性不亚于CPU和内存。

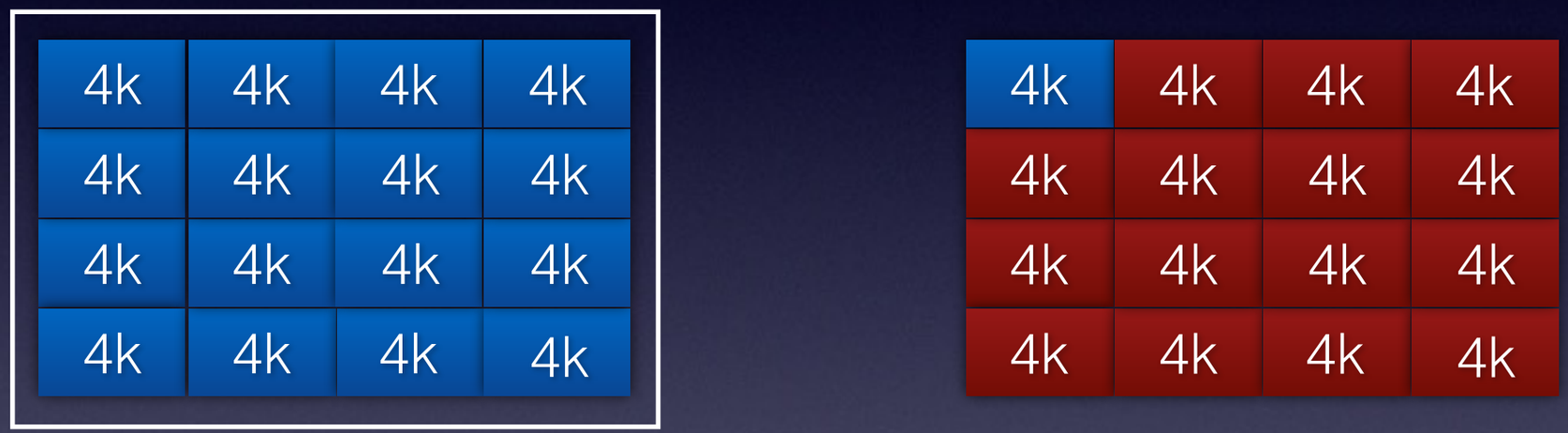
- 随机读写
  - 顺序读写性能进步的非常快，跟SSD的差距已经缩小了很多，但是随机读写的性能依旧很差。
  - 失去预读（read-ahead的）优化效果。
  - 产生大量的失效页面，增大了触发“写入放大”的概率。

数据以4k的
页大小写入

数据以512k的
块大小擦除

数据以4k的
页大小写入


4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k


4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k


4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k
4k 4k 4k 4k


4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k
4k

实例1

- Sqlite 插入数据
  - 逐条插入

```
for (AppInfo appInfo : list) {
    DBUtil.insert(helper, appInfo);
}
```

  - 事务

```
db.beginTransaction();
for (AppInfo appInfo : list) {
    ContentValues values = appInfo.getContentValues();
    db.insert(DBHelper.TABLE_APP, null, values);
}
db.setTransactionSuccessful();
db.endTransaction();
```

  - 开启事务批量插入，使用SQLiteStatement

```
SQLiteStatement stat = db.compileStatement(sql);
db.beginTransaction();
for (AppInfo appInfo : list) {
    stat.binXXX();
    stat.executeInsert();
}
db.setTransactionSuccessful();
db.endTransaction();
```

实例1

- Sqlite 插入数据
  - 逐条插入

```
for (AppInfo appInfo : list) {
    DBUtil.insert(helper, appInfo);
}
```

耗时： 106524ms,也就是106s

  - 事务

```
db.beginTransaction();
for (AppInfo appInfo : list) {
    ContentValues values = appInfo.getContentValues();
    db.insert(DBHelper.TABLE_APP, null, values);
}
db.setTransactionSuccessful();
db.endTransaction();
```

耗时： 2968ms

  - 开启事务批量插入，使用SQLiteStatement

```
SQLiteStatement stat = db.compileStatement(sql);
db.beginTransaction();
for (AppInfo appInfo : list) {
    stat.binXXX();
    stat.executeInsert();
}
db.setTransactionSuccessful();
db.endTransaction();
```

耗时： 1365ms

实例2

Android系统中使用SharedPreferences文件来保存数据非常方便，在需要保存的时候只需要调用commit就可以。

- Commit一次对应一次文件的打开和关闭。
- 可以缓存，延迟写入，从而减少写入次数。保留最后一个commit即可。
- 前提是确保数据安全，不能丢失数据。

实例3

合理的使用ByteArrayOutputStream。

- 使用ObjectOutputStream

```
private void storeData(List<Object> list) {
    oos = new ObjectOutputStream(openFileOutput(file));
    oos.writeObject(list);
    oos.flush();
    oos.close();
}
```

- ByteArrayOutputStream + ObjectOutputStream

```
private void storeData(List<Object> list) {
    baos = new ByteArrayOutputStream();
    oos = new ObjectOutputStream(baos);
    oos.writeObject(list);
    oos.flush();
    oos.close();

    fos = openFileOutput(file);
    baos.writeTo(fos);
    baos.flush();
    fos.flush();
    fos.close();
}
```

实例3

合理的使用ByteArrayOutputStream。

- 使用ObjectOutputStream

```
private void storeData(List<Object> list) {
    oos = new ObjectOutputStream(openFileOutput(file));
    oos.writeObject(list);
    oos.flush();
    oos.close();
}
```

list 的size多大，就会有多少次IO

- ByteArrayOutputStream + ObjectOutputStream

```
private void storeData(List<Object> list) {
    baos = new ByteArrayOutputStream();
    oos = new ObjectOutputStream(baos);
    oos.writeObject(list);
    oos.flush();
    oos.close();

    fos = openFileOutput(file);
    baos.writeTo(fos);
    baos.flush();
    fos.flush();
    fos.close();
}
```

只有一次IO

实例4

XiaoYuan 、Sina SDK频繁的IO操作

- 短信启动后，通过systrace、IO检测发现，XiaoYuan在频繁的做IO操作。
- 天气启动后，sina的SDK也在频繁的做数据库操作。
  - 磁盘碎片化
  - 占用CPU资源

网络传送

- 业务成功率
  - 弱网和拥塞网络（电梯、发布会发照片）
- 网络延时
  - DNS Cache过期时间
  - DNS解析耗时 （IP直连， 域名重用）
- 宽带成本
  - WebP图片压缩、apk瘦身、H264/265视频压缩、gzip文本压缩

实例

- AppStore应用的更新下载
- 弱网下载策略
- 对性能和功耗的影响

异常

- try – catch
- Log.wtf
- 重复kill – restart

工具

CPU相关

- top
- ps
- /proc/[pid]/stat
- dumpsys cpuinfo
- cpu frequency
- systrace
- traceview

top

```
User 3%, System 12%, IOW 0%, IRQ 0%
User 3 + Nice 0 + Sys 10 + Idle 64 + IOW 0 + IRQ 0 + SIRQ 0 = 77

  PID USER     PR  NI CPU% S  #THR     VSS     RSS PCY Name
 1287 shell    20   0  11% R     1  10260K   2220K   fg top
 1797 u0_a34   20   0   1% S    35 2424580K 116896K   fg com.android.systemui
  526 system   -2  -8   1% S    12 138592K   9472K  unk /system/bin/surfaceflinger
```

- PID:进程ID
- CPU% - 当前瞬时所以使用CPU占用率
- #THR - 程序当前所用的线程数
- USER - 运行当前进程的用户id (uid)
- VSS - Virtual Set Size 虚拟耗用内存（包含共享库占用的内存）
- RSS - Resident Set Size 实际使用物理内存（包含共享库占用的内存）

- adb shell top -m 10 -n 1 -t -d 1 -s cpu/vss/rss/thr
- adb shell top | grep -v ‘0%’

ps

adb shell ps -t -p -x -c -P [pid]

```
USER      PID   PPID  VSIZE  RSS  CPU PRIO  NICE  RTPRI SCHED  PCY WCHAN              PC  NAME
u0_a34    1797  579   2432624 125664 2  20    0     0     0     fg  SyS_epoll_ 0000000000 S com.android.systemui (u:102407, s:26529)
u0_a34    1803  1797  2432624 125664 0  29    9     0     0     fg  futex_wait 0000000000 S Jit thread pool (u:12, s:8)
u0_a34    1804  1797  2432624 125664 1  20    0     0     0     fg  do_sigtime 0000000000 S Signal Catcher (u:0, s:0)
u0_a34    1805  1797  2432624 125664 2  20    0     0     0     fg  futex_wait 0000000000 S ReferenceQueueD (u:9, s:1)
u0_a34    1806  1797  2432624 125664 2  20    0     0     0     fg  futex_wait 0000000000 S FinalizerDaemon (u:30, s:0)
u0_a34    1807  1797  2432624 125664 3  20    0     0     0     fg  futex_wait 0000000000 S FinalizerWatchd (u:0, s:1)
u0_a34    1808  1797  2432624 125664 1  20    0     0     0     fg  futex_wait 0000000000 S HeapTaskDaemon (u:185, s:30)
u0_a34    1809  1797  2432624 125664 2  20    0     0     0     fg  binder_thr 0000000000 S Binder:1797_1 (u:559, s:902)
u0_a34    1810  1797  2432624 125664 1  20    0     0     0     fg  binder_thr 0000000000 S Binder:1797_2 (u:526, s:924)
```

- -t 显示该进程下的线程列表
- -p显示进程的优先级和nice等级
- -x显示进程耗费的用户时间和系统时间
- -c显示进程耗费的cpu时间
- -P显示调度策略

dumpsys cpuinfo

adb shell dumpsys cpuinfo

```
CPU usage from 105091ms to 7169ms ago (2017-08-16 23:12:13.077 to 2017-08-16 23:13:50.998):
  22% 1797/com.android.systemui: 18% user + 4.3% kernel / faults: 57 minor
  14% 526/surfaceflinger: 7.5% user + 6.8% kernel
  8% 1294/system_server: 4.2% user + 3.8% kernel / faults: 1326 minor
  1.8% 32134/mdss_fb0: 0% user + 1.8% kernel
  1.2% 1836/kworker/u16:5: 0% user + 1.2% kernel
  1% 2286/kworker/u16:7: 0% user + 1% kernel
  1% 2602/kworker/u16:9: 0% user + 1% kernel
  0.9% 2035/kworker/u16:6: 0% user + 0.9% kernel
  0.6% 17961/com.ss.android.article.news:push: 0.4% user + 0.2% kernel / faults: 247 minor
  0.4% 3141/com.qiyi.video: 0.2% user + 0.2% kernel
  0.4% 10/rcuop/0: 0% user + 0.4% kernel
  0.4% 25/rcuop/2: 0% user + 0.4% kernel
  0.3% 17986/com.ss.android.article.news: 0.2% user + 0.1% kernel / faults: 184 minor
  0.3% 329/mmc-cmdqd/0: 0% user + 0.3% kernel
  0.3% 7/rcu_preempt: 0% user + 0.3% kernel
  0.2% 707/msm_irqbalance: 0% user + 0.2% kernel
  0.2% 408/com.jingdong.app.mall: 0% user + 0.1% kernel / faults: 77 minor
  0.2% 420/logd: 0% user + 0.1% kernel
  0.2% 25079/com.tencent.mobileqq:MSF: 0.1% user + 0% kernel / faults: 188 minor 1 major
  0.2% 32/rcuop/3: 0% user + 0.2% kernel
6.4% TOTAL: 3.6% user + 2.3% kernel + 0.1% iowait + 0.1% irq + 0.1% softirq
```

/proc/[pid]/stat

adb shell cat /proc/1797/stat

```
1797 (ndroid.systemui) S 579 579 0 0 -1 1077936448 236942 0 214 0 81065 21147 0 0 20 0
38 0 2647 2494234624 30892 18446744073709551615 1 1 0 0 0 0 4612 0 34040
18446744073709551615 0 0 17 1 0 0 0 0 0 0 0 0 0 0 0 0
```

- adb shell cat /proc/1797/stat | awk ‘{print $39}’
- adb shell cat /proc/1797/stat | awk ‘{print $14+$15}’

CPU frequency

- adb shell cat /sys/devices/system/cpu/cpu*/cpufreq/scaling_cur_freq
- adb shell cat /sys/devices/system/cpu/online
- adb shell cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
- echo MaxFreq > scaling_min_freq & echo MaxFreq > scaling_max_freq

Systrace

- python systrace.py --list
- systrace_full 4 ~/Desktop/traces/test.html

```
function systrace_full
{
    if [ $# = 2 ]
    then
        python ~/androidSDK/android-sdk-linux/platform-tools/systrace/
systrace.py --time=$1 gfx input view webview wm am sm audio video camera hal
app res dalvik rs bionic power pm ss database network sched irq freq idle
disk mmc load sync workq memreclaim regulators binder_driver binder_lock
pagecache -o $2
    fi
}
```

-b buffer大小

TraceView

- Monitor/DDMS
- Cold launch

```
am start -n com.smartisanos.notes/.NotesActivity -P /data/local/tmp/notes.trace
```

systrace Alert

- Inefficient View alpha usage （alpha 动画）
- Expensive rendering with Canvas.saveLayer （HARDWARE）
- Path Texture Churn （绘制路径）
- Expensive Bitmap uploads （减少图片的修改）
- Inefficient during Lisview recycling/rebinding （复用item）
- Expensive Measure/Layout pass （动画时减少layout，层级简单）
- Long View.draw（尽量避免在onDraw执行耗时操作）
- Blocking Garbage Collection（重用避免垃圾回收）
- Lock contention
- Scheduling delay（网络IO、磁盘IO）

Traceview

列出所有的调用方法，展开可以看到所有parent和children的子项，分别指该方法的调用者和调用方法。

- Incl CPU time 函数占用cpu的时间，包括其调用函数的时间（IO block， wait）
- Excl CPU time 函数自身占用cpu时间，不包含其调用函数的时间（IO block，wait）
- Incl Real time 函数真实执行耗时，包含其调用函数的真实耗时。
- Excl Real time 函数自身真实执行耗时，不包含其调用函数耗时。
- Call + Recur calls/Total 函数调用次数，包含递归调用。
- CPU Time /Call 函数平均占用cpu时间。
- Real Time /Call 函数平均真实耗时。

Memory相关

- top/procrank
- MAT/Monitor
- StrictMode
- meminfo
- libc_malloc_debug_leak.so
- LeakCanary
- GC Log

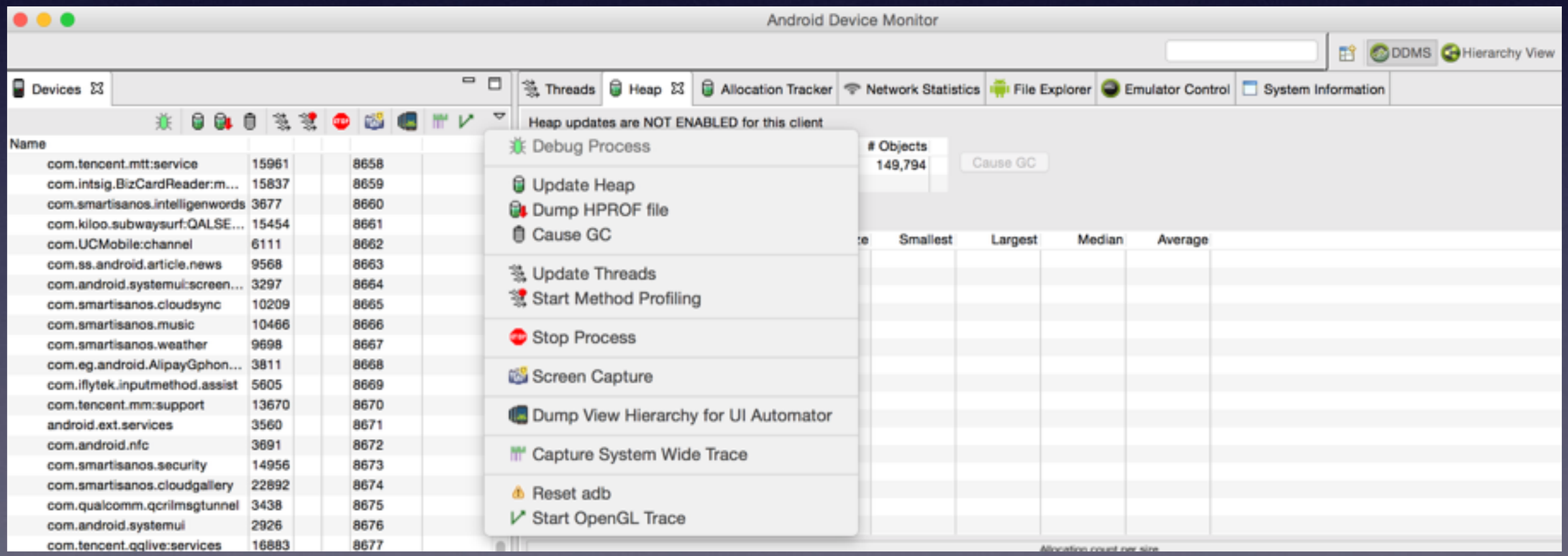
top/procrank

```
adb shell procrank
SWAP offset 393215 is out of swap bounds.
  PID      Vss      Rss      Pss      Uss     Swap    PSwap    USwap    ZSwap  cmdline
14438  2148288K  250836K  188831K  167864K   35936K     658K       0K     226K  com.jingdong.app.mall
 1672  2585980K  135352K   92515K   88904K   54296K   16609K   15744K    5724K  system_server
 2867  1779224K  123108K   71467K   57992K   42284K    4555K    3696K    1570K  com.smartisanos.keyguard
15568  1949048K  106548K   60196K   55652K   47808K   10785K   10080K    3717K  com.taobao.taobao
 3663  1719748K   94292K   57065K   55976K   41652K    3717K    2848K    1281K  com.smartisanos.voice
 3887  2474516K  111496K   54692K   35908K   62416K   24639K   23780K    8492K  com.smartisanos.launcher
 2926  2414528K   96940K   51210K   44788K   53276K   15545K   14688K    5358K  com.android.systemui
```

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
Swap: 虚拟内存
PSwap: Proportional Swap
USwap: Unique Swap
ZSwap: Zram Swap

Monitor – >DDMS

- Monitor是一个调试信息合集，里面包含时延、内存、线程、CPU、文件系统、流量等一系列信息的获取和展示。



MAT (Memory Analyzer Tools）

- 抓取Hprof文件
  - DDMS 最简单，实用性最强
  - adb shell dumpheap pid outfilePath
  - adb shell kill -10 pid （小于android 4.x）
- Hprof – conv工具转换
- MAT 非常灵活且强大，但使用门槛相对较高，不易上手。

StrictMode

- 两种策略
  - 线程策略(ThreadPolicy)
  - 虚拟机策略(VMPolicy)
- ThreadPolicy：
  - 自定义的耗时调用，使用detectCustomSlowCalls()开启
  - 磁盘读取操作，使用detectDiskReads()开启
  - 磁盘写入操作，使用detectDiskWrites()开启
  - 网络操作，使用detectNetwork()开启
- VMPolicy：
  - Activity泄漏，使用detectActivityLeaks()开启
  - 未关闭的Closable对象泄漏，使用detectLeakedClosableObjects()开启
  - 泄露的Sqlite对象，使用detectLeakedSqlLiteObjects()开启
  - 检测实例数量，使用setClassInstanceLimit()开启
- adb logcat | grep StrictMode

dumpsys meminfo [pid]/[procName]

```
dumpsys meminfo com.jingdong.app.mall
Applications Memory Usage (in Kilobytes):
Uptime: 37732157 Realtime: 103856179

** MEMINFO in pid 28817 [com.jingdong.app.mall] **
                   Pss  Private  Private  SwapPss     Heap     Heap     Heap
                 Total    Dirty    Clean    Dirty     Size    Alloc     Free
                ------   ------   ------   ------   ------   ------   ------
  Native Heap    26537    26444        0        0    41472    30729    10742
  Dalvik Heap    88470    88448        0        0   109213    92829
```

native内存占用

java层内存占用

```
Objects
                 Views:      797         ViewRootImpl:        1
           AppContexts:        6           Activities:        2
                Assets:        7        AssetManagers:        7
          Local Binders:       41        Proxy Binders:       31
          Parcel memory:       24         Parcel count:       87
       Death Recipients:        0      OpenSSL Sockets:        9
               WebViews:        0

SQL
         MEMORY_USED:      667
  PAGECACHE_OVERFLOW:      179          MALLOC_SIZE:       62
```

Views、AppContexts、Activities
持续上升有可能发生内存泄漏

数据库使用的内存

dumpsys

```
dumpsys --help
usage: dumpsys
         To dump all services.
or:
       dumpsys [-t TIMEOUT] [--help | -l | --skip SERVICES | SERVICE [ARGS]]
         --help: shows this help
         -l: only list services, do not dump them
         -t TIMEOUT: TIMEOUT to use in seconds instead of default 10 seconds
         --skip SERVICES: dumps all services but SERVICES (comma-separated list)
         SERVICE [ARGS]: dumps only service SERVICE, optionally passing ARGS to it
```

- meminfo　　显示内存信息
- cpuinfo　　显示CPU信息
- SurfaceFlinger　显示 SurfaceFlinger相关信息
- account　　显示accounts信息
- activity　　显示所有的activities的信息
- window　　显示键盘，窗口和它们的关系
- wifi　　显示wifi信息

libc_malloc_debug_leak.so

- NDK使用的内存是透传出Dalvik的，因此在Hprof分析过程中是见不到native的内存分配的。
- native 内存分配所用到的库是libc.so, 而libc_malloc_debug_leak.so 是用来监视libc.so
- 设置libc.debug.malloc 为 1检测内存泄漏 （需重启android框架）

libc.debug.malloc	1	检测内存泄漏
libc.debug.malloc	5	分配的内存用0xeb填充，释放的内存用0xef填充
libc.debug.malloc	10	内存分配打pre-和post- 的桩子，可以检测内存的overruns
libc.debug.malloc	20	SDK模拟器上检测内存用

独立版DDMS可以展示Native Heap

GC log

- Dalvik GC log
  - https://github.com/oba2cat3/logcat2memorygraph

- Art GC log
  - Full
  - Partial
  - Sticky

```
08-20 21:12:00.918 10568 10581 I art     : Background partial concurrent mark sweep GC freed 27770(3MB)
AllocSpace objects, 38(16MB) LOS objects, 13% free, 105MB/121MB, paused 2.666ms total 122.645ms
08-20 21:12:01.326 10568 10581 I art     : Background sticky concurrent mark sweep GC freed 16683(2MB)
AllocSpace objects, 56(1120KB) LOS objects, 0% free, 125MB/125MB, paused 4.703ms total 100.187ms
08-20 21:12:01.494 10568 10581 I art     : Background partial concurrent mark sweep GC freed 15707(1524KB)
AllocSpace objects, 50(19MB) LOS objects, 12% free, 115MB/131MB, paused 3.083ms total 167.050ms
08-20 21:12:01.791 10568 10581 I art     : Background partial concurrent mark sweep GC freed 10619(752KB)
AllocSpace objects, 32(26MB) LOS objects, 12% free, 110MB/126MB, paused 3.357ms total 113.592ms
```

I/O相关

- top IOW%
- vmstat
- iotop
  - https://github.com/laufersteppenwolf/iotop.git
- iodump
  - sudo sh -c 'echo 1 > /proc/sys/vm/block_dump'
  - https://github.com/true/aspersa-mirror/blob/master/iodump

I/O相关

大句柄数。当句柄数目达到限制后，就回出现”too many files open”。查看进程占用的句柄数有几种办法：

1） 通过cat/proc/pid/fd可以查看线程pid号打开的线程；

2） 通过lsof命令， lsof -p proc 命令结果如下

http://mantis.smartisan.cn/view.php?id=222680

流畅性检测

- SurfaceFlinger

adb shell service call SurfaceFlinger 1001

```
08-21 10:23:13.929   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 54.854240
08-21 10:23:15.424   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 20.067972
08-21 10:23:15.970   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 54.888512
08-21 10:23:16.518   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 54.818512
08-21 10:23:18.012   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 20.071144
08-21 10:23:18.560   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 54.799873
08-21 10:23:19.105   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 54.987331
08-21 10:23:20.564   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 20.570780
08-21 10:23:21.110   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 54.874931
08-21 10:23:21.658   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 54.772034
08-21 10:23:23.098   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 20.839397
08-21 10:23:23.644   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 54.876499
08-21 10:23:24.191   544   544 I SurfaceFlinger: #####FPS of last 30 frame: 54.843575
```

流畅性检测

System Monitor

- adb shell service call activity 7668 i32 1打开流畅度log
- /data/syslog/monitor/fluency/
- adb logcat | grep Jank
- reason （cpu， memory， 温度，unknown）

Binder统计

app<->system server

app<->app

adb shell service call activity 1010 i32 3 i32 1

adb shell service call activity 1011 i32 20

adb logcat -s BinderTrans

实例

- settings　0223892: [Odin][4.1.1][量产][高配][性能]在打开应用信息流量使用界面时发生卡顿。
- 浏览器　0195757: [Osborn][DVT2][高配][性能]在浏览器中添加新页面时有帧卡顿现像。
- 相机

Something else

- Bitmap 尽量不要使用decodeFile 而用decodeStream，并且传入的是BufferInputStream，decodeResource同样有性能问题，使用decodeResourceStream代替
- HashMap/HashSet size 最好是2^x， hashCode & (size-1)
- for (int item : arrays)
- 用局部变量取代全局变量

```
private void test() {
    int tempValue = mValue;
    …
}
```

- 动画过程中硬件加速（硬件加速最大支持2048*2048）
  - invalidate只对当前view有效 (微信朋友圈评论点赞的动画)
  - bimap重用将不起作用
  - 内存占用会增加
- 比较耗时的操作是否可以native实现 （String.equals, indexOf, replace）

参考

- https://wiki.cc.gatech.edu/epl/index.php/Android_tools
- http://kernel.meizu.com/zram-introduction.html
- http://bbs.pceva.com.cn/forum.php?mod=viewthread&action=printable&tid=8277
- http://blog.csdn.net/qiiiiiiq/article/details/52621209
- https://baike.baidu.com/item/Swap分区/7613378?fr=aladdin
- http://blog.csdn.net/haima1998/article/details/51508947
- 《大话Java性能优化》

Q/A

Thanks!