内存优化

- **内存泄漏**
  - 定义
    - 大对象被长生命周期的对像持有
    - 堆内存泄露，指的是在程序运行时，给对象分配的内存，当程序退出或者退出界面时，分配的内存没有释放或者因为其他原因无法释放
    - 资源泄露，比如 FD、socket、线程等等，这些在每个手机上都是有数量的限制，如果使用了不释放，就会因为资源的耗尽而崩溃，我们在线上就出现过 FD 的泄露，导致崩溃率涨了 3 倍
  - 泄露的点
    - 匿名内部类、非静态内部类：隐示持有外部类的实例
    - 被静态实例（长生命周期实例）持有（单例持有大对像）
    - 集合：全局性的集合类没有相应的删除机制，导致内存被占用
      - handler
      - threadlocal
      - map，list cacle
    - fd未关闭（file，thread）Native（webview） 泄露
      - webview泄露
      - native持有Activity
      - fd 资源没有关闭
      - thread一直没有结束（持有fd）
      - threadhandler一直没有结束（持有三个fd）
  - 监控手段
    - leakcanary ，Memory Analyzer，MAT
    - memory profile
    - 大图识别，后监控
    - 监控大对像（根对象activity）
    - 监控容器数据（Handler，ThreadLocal，HashMap，List）
    - 监控FD的释放（File， HandleThread thread）

- 监控线程的状态
- Thread 池化管理

内存溢出

- 定义
  - 主要是Java 堆内存溢出，还有部分是超出当前资源（thread，fd）阈值；
  - 分配的内存到达 Java 堆的上限
  - 可用内存很多，因为内存碎片化，没有足够的连续段的空间分配
  - 对象的单次分配或者多次分配累计过大，例如在循环动画中一直创建 Bitmap
  - FD 的数量超出当前手机的阈值
- 可用内存
  - 32 位设备可以使用的虚拟内存大小 3GB
  - 64 位应用可以使用的虚拟内存大小 512GB
- 防止手段-开源
  - 如何解决 Java 堆内存不足的问题：大对象（bitmap，memory cache）移动到 native 内存里去
    - Android 3.0 ～ Android 7.0 上主要将 Bitmap 对象和像素数据统一放到 Java 堆中，Java 堆上限 512MB，而 Native 占用虚拟内存，32 的设备可使用 3GB，64 位的设备更大，因此我们可以尝试将 Bitmap 分配到 Native 上，缓解 Java 堆的压力，降低 OOM 崩溃
  - Native 线程默认的栈空间大小为 1M 左右，经过测试大部分情况下线程内执行的逻辑并不需要这么大的空间，因此 Native 线程栈空间减半，可以减少
  - 系统预分配区域中其中
    [anon:libwebview reservation] 区域占用 130MB 内存，可以尝试释放 WebView 预分配的内存，减少一部分虚拟内存
    虚拟机堆空间减半，在上面提到过有两片大小相同的区域分别 dalvik-main space 和 dalvik-main space 1，虚拟机堆空间减半其实就是减少其中一个 main space 所占用的内存
  - 统一三方图片组件，减少各自的缓存，建立统一的缓存管理组件
- 防止手段-截流
  - 按需加载。so dex （组件化，模块化）
    - 删减代码，减少 dex 文件占用的内存
    - 减少 App 中 dex 数量，非必要功能，可以通过动态下发
    - 按需加载 so 文件，不要提前加载所有的 so 文件，需要使用时再去加载

- 在使用完成后（生命周期结束后）及时释放（注意list，map，handler。。。）
  - 收敛 Bitmap，避免重复创建 Bitmap，退出界面及时释放掉资源（Bitmap、动画、播放器等等资源）
  - 内存回收兜底策略，当 Activity 或者 Fragment 泄露时，与之相关联的动画、Bitmap、 DrawingCache 、背景、监听器等等都无法释放，当我们退出界面时，递归遍历所有的子 view，释放相关的资源，降低内存泄露时所占用的内存
- 收敛线程，祖传代码在项目中有很多地方使用了 new Thread 、 AsyncTask 、自己创建线程池等等操作，通过统一的线程池等手段减少 App 创建线程数量，降低系统的开销
- SparseArray替代HashMap
- 主动GC
- 使用第三方图片库时，需要针对高端机和低端机设置图片库不同的缓存大小，这样我们在高端机上保证体验的同时，降低低端机 OOM 崩溃率
- 针对低端机和高端机采用不同的策略，减少低端机内存的占用
- 线下重复图片检测

- FD泄露问题
  - 同一个问题可能出现不同堆栈， 比较隐晦
  - Fd泄漏时内存可能不会出现不足，就算触发GC也不一定能够回收已经创建的文件句柄
  - 日志关键字： `8`
  - 泄露的方式
    - Resource相关：只要涉及到FileInputStream，FileOutputStream，FileReader，FileWriter api都有可能
    - HandlerThread相关：每一个HandlerThread会产生最少三个fd，不要建立过多的HandlerThread；
    - Thread.start也有产生fd，不过建立过多；
    - Inputchannel也会可能出现fd泄露问题
    - 在Activity中不断弹Dialog
    - bitmap也是需要fd的

- **内存抖动**
  - 定义

- 对象频繁创建，释放，典型的是在执行频率很高的方法里创建对象，用完马上释放了

- 手段
  - 对象池，防抖
  - 注意在执行频率很高的方法里的资源使用
- 常见案例
  - 字符串使用加号拼接
    - 使用StringBuilder替代。
    - 初始化时设置容量，减少StringBuilder的扩容。
  - 资源复用
    - 使用全局缓存池，以重用频繁申请和释放的对象。
    - 注意结束使用后，需要手动释放对象池中的对象
  - 减少不合理的对象创建
    - ondraw、getView中对象的创建尽量进行复用。
    - 避免在循环中不断创建局部变量
  - 使用合理的数据结构
    - 使用SparseArray类族来替代HashMap。

常见问题 16