手Q iOS客户端性能监控和优化实践

罗鑫（rosen）

Geekbang. 极客邦科技 InfoQ

极客邦科技
会议推荐2019

5月

QCon 北京
全球软件开发大会
大会：5月6–8日
培训：5月9–10日

QCon 广州
全球软件开发大会
培训：5月25–26日
大会：5月27–28日

6月

GTLC GLOBAL TECH LEADERSHIP CONFERENCE 上海
技术领导力峰会
时间：6月14–15日

GMTC 北京
全球大前端技术大会
大会：6月20–21日
培训：6月22–23日

7月

ArchSummit 深圳
全球架构师峰会
大会：7月12–13日
培训：7月14–15日

10月

QCon 上海
全球软件开发大会
大会：10月17–19日
培训：10月20–21日

11月

GMTC 深圳
全球大前端技术大会
大会：11月8–9日
培训：11月10–11日

AiCon 北京
全球人工智能与机器学习大会
大会：11月21–22日
培训：11月23–24日

12月

ArchSummit 北京
全球架构师峰会
大会：12月6–7日
培训：12月8–9日

Geekbang 极客邦科技 | TGO 鲲鹏會

TGO鲲鹏会

汇聚全球科技领导者的高端社群

全球12大城市

850+高端科技领导者

使命 Mission

为社会输送更多优秀的科技领导者

愿景 Vision

构建全球领先的有技术背景优秀人才的学习成长平台

扫描二维码，了解更多内容

自我介绍

罗鑫（rosen），来自腾讯 QQ研发中心。2015年加入腾讯手Q团队，主要工作是负责手Q iOS性能监控体系的建设，在卡顿、内存、发热耗电和Crash等移动终端的性能问题优化有丰富的实践经验，目前是手Q iOS客户端性能负责人。

目录

目录

手Q在发展中遇到的性能挑战

用户反馈

性能问题主要有三类：

1.随机卡顿

场景不固定，一般没有规律复现定位困难

2.爆内存闪退问题

内存使用不当引发的爆内存闪退，无法被传统手段捕获

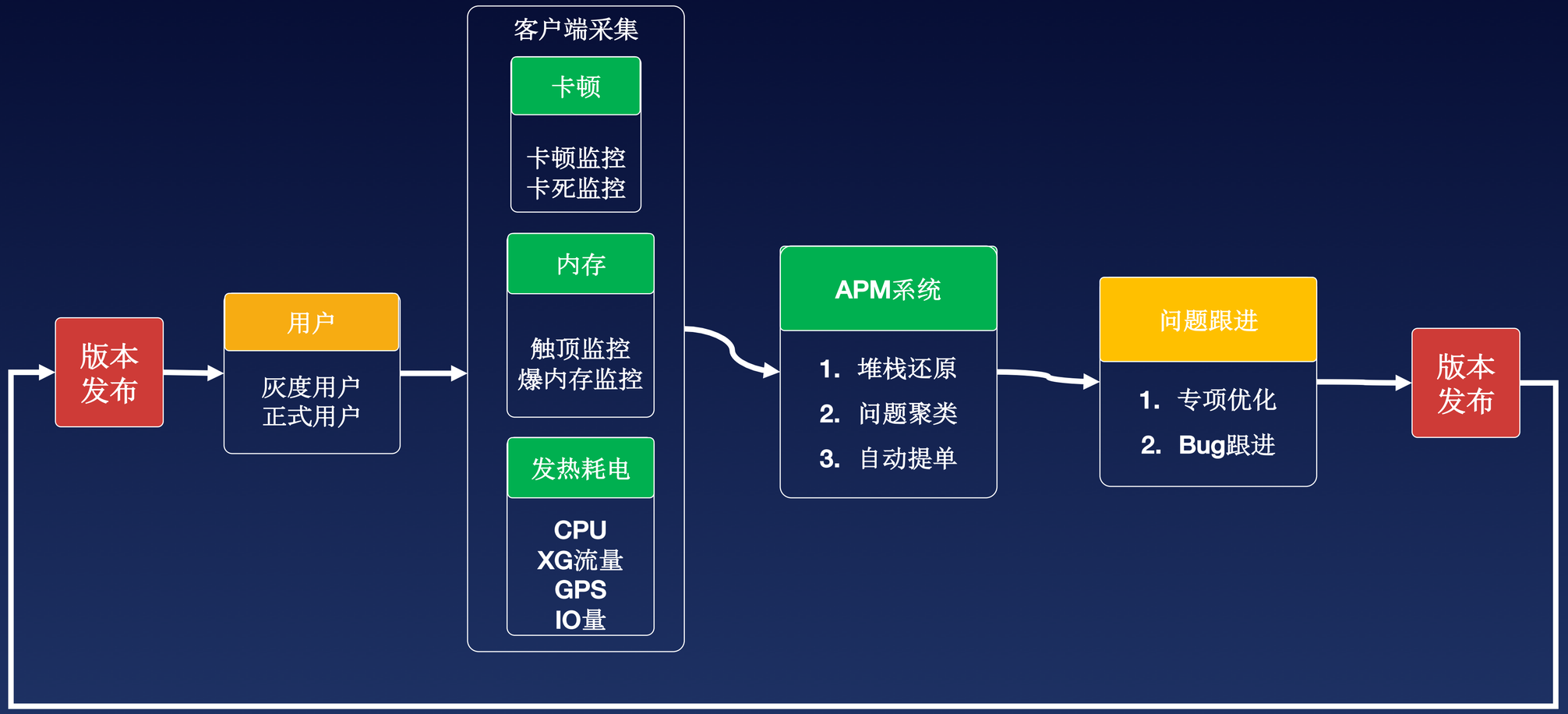
3.发热耗电

不当逻辑引发的耗电太多或者手机发热严重

思考：

要有完善的线上工具监控和定位相关性能问题

手Q APM线上性能监控体系



目录

从手Q早期典型卡顿案例说起

14年开始就有用户反馈手Q使用中经常卡顿

问题特点：

难复现：场景不固定，问题发生无规律

难定位：大部分不必现，日志无法准确打点

问题原因：

当NSUserDefault组件在系统低内存时会把内存中的数据进行置换，在执行的过程中会加锁，此时在主线程访问NSUserDefault会导致长时间卡顿

卡顿堆栈

```
Thread 0:",
    "1 CoreFoundation -[CFPrefsSearchListSource generationCountFromListOfSources:count:] (in CoreFoundation) + 76",
    "2 CoreFoundation -[CFPrefsSearchListSource alreadylocked_copyDictionary] (in CoreFoundation) + 164",
    "3 CoreFoundation -[CFPrefsSearchListSource alreadylocked_copyValueForKey:] (in CoreFoundation) + 52",
    "4 CoreFoundation ___CFPreferencesCopyAppValueWithContainer_block_invoke (in CoreFoundation) + 72"
```

手Q卡顿监控方案演进史

第一版

OC入口方法Hook

Hook objc_msgsend、NSNotificaiton和NSTimer和perforselector OC函数入口，打印方法耗时

第二版

基于RunLoop observer 抓栈监控方案

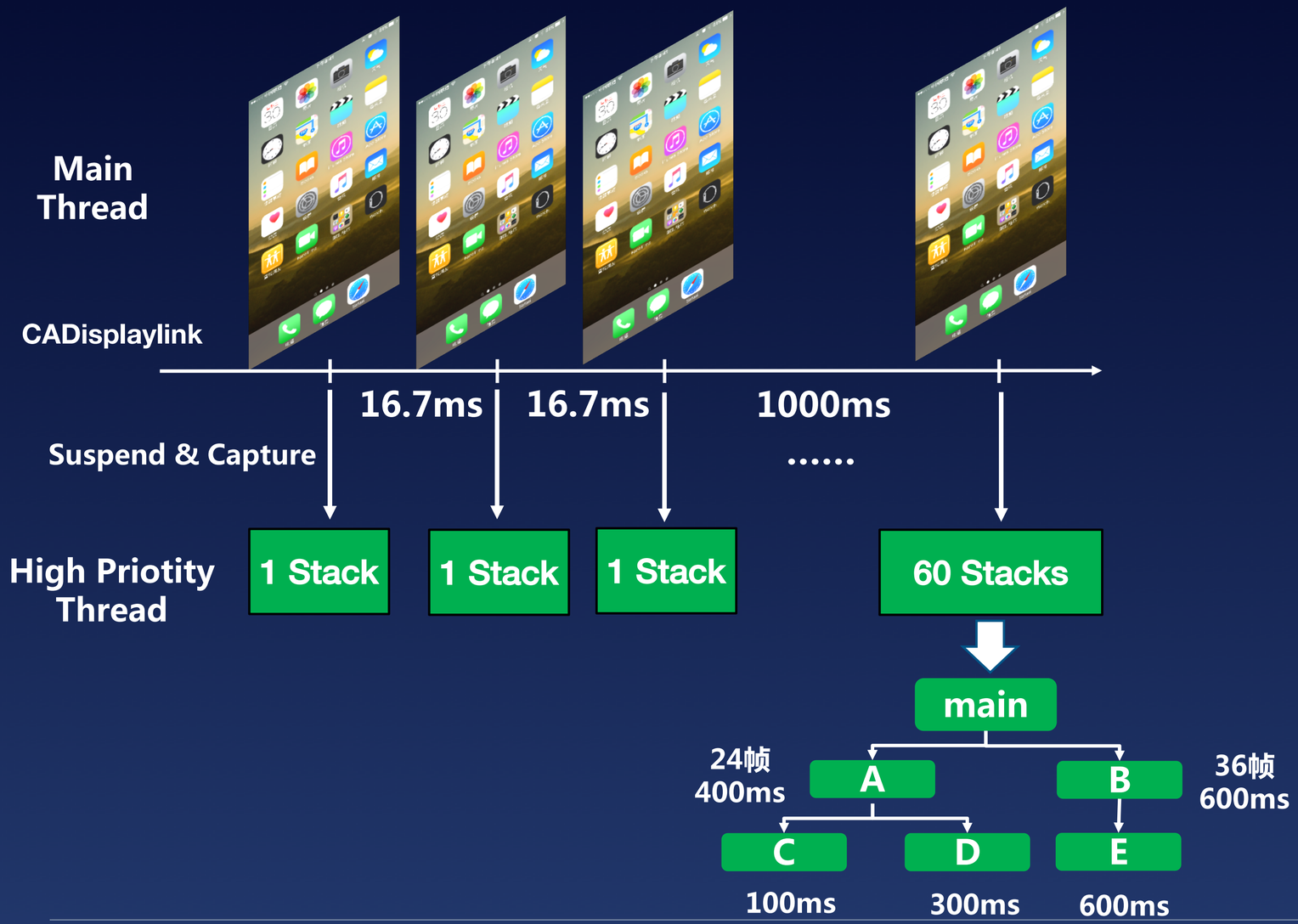
主线程监听NSRunloop observer，子线程利用NSTimer定时抓取堆栈

第三版

基于CADisplayLink 回调的抓栈监控方案

主线程监听CADisplayLink屏幕刷新回调，子线程利用CADisplayLink回调抓取堆栈

基于CADisplayLink的卡顿监控方案



方案概述：

利用CADisplayLink提供的屏幕刷新回调，在主线程两次刷新之间通过高优先级子线程持续抓取主线程堆栈

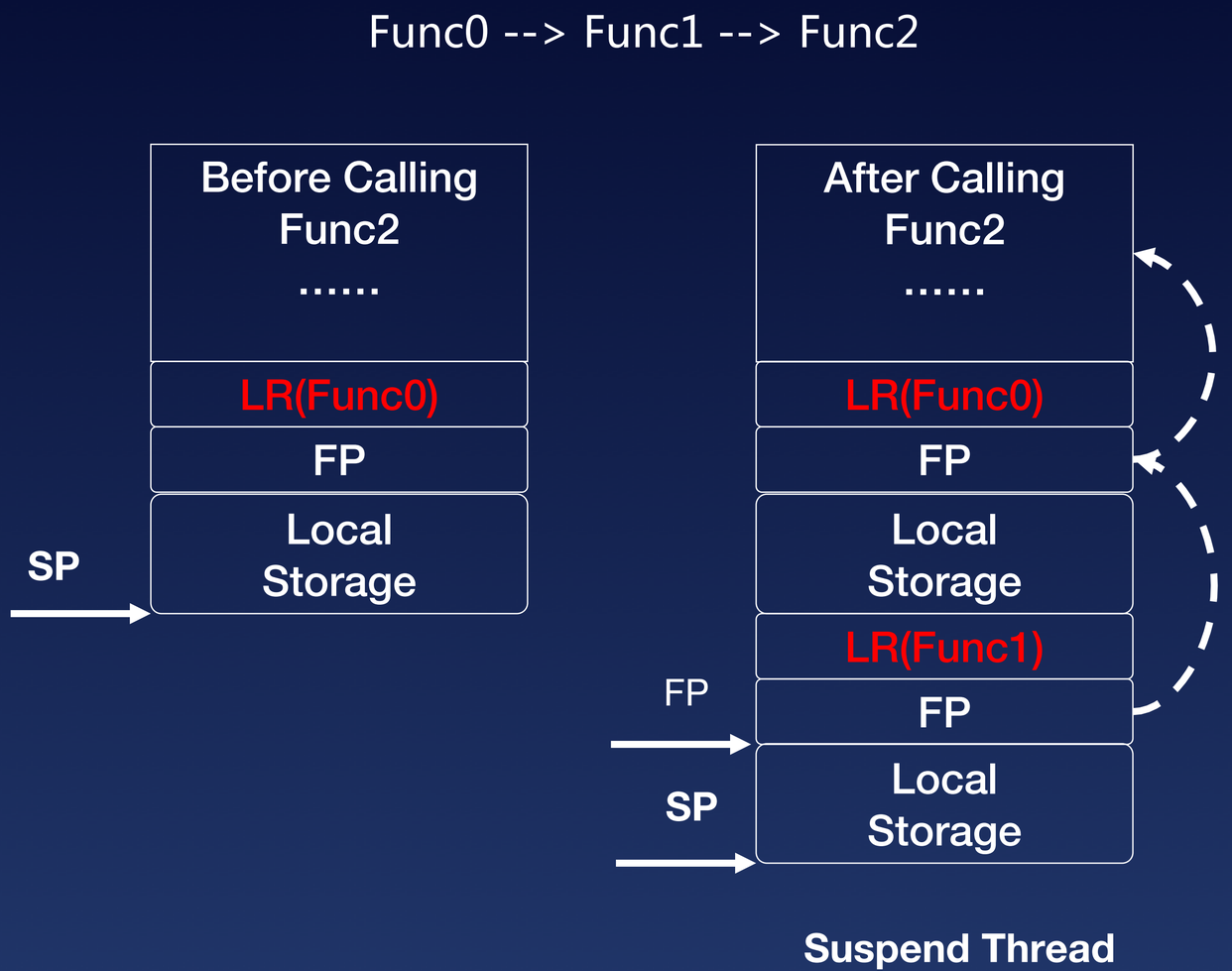
方案优势：

1.屏幕刷新回调，保证所有卡顿都不会遗漏

2.CADisplayLink硬件事件回调，相比NSTimer更加精准无延迟

3.子线程直接抓取堆栈，抓取效率高

堆栈抓取原理



方案优点：

1.单次采集耗时0.2ms，性能开销小

2.整体CPU消耗<2%左右

技术要点：

- 判断Frame Pointer合理性，避免栈帧逆向增长
- 提前分配内存，避免死锁
- 卡顿一般只抓主线程堆栈，超过5秒抓一次全部线程堆栈

iOS常见卡顿问题总结

1.NSUserDefaults：低内存的情况下，系统拷贝内存数据到磁盘

2.I/O：文件读写、数据库操作

3.系统接口：获取mac地址、切换音频设备、读取桌面icon未读数

4.死锁以及锁的滥用

5.排版绘制： sizeWithFont、 [EAGLContext setCurrentContext:]

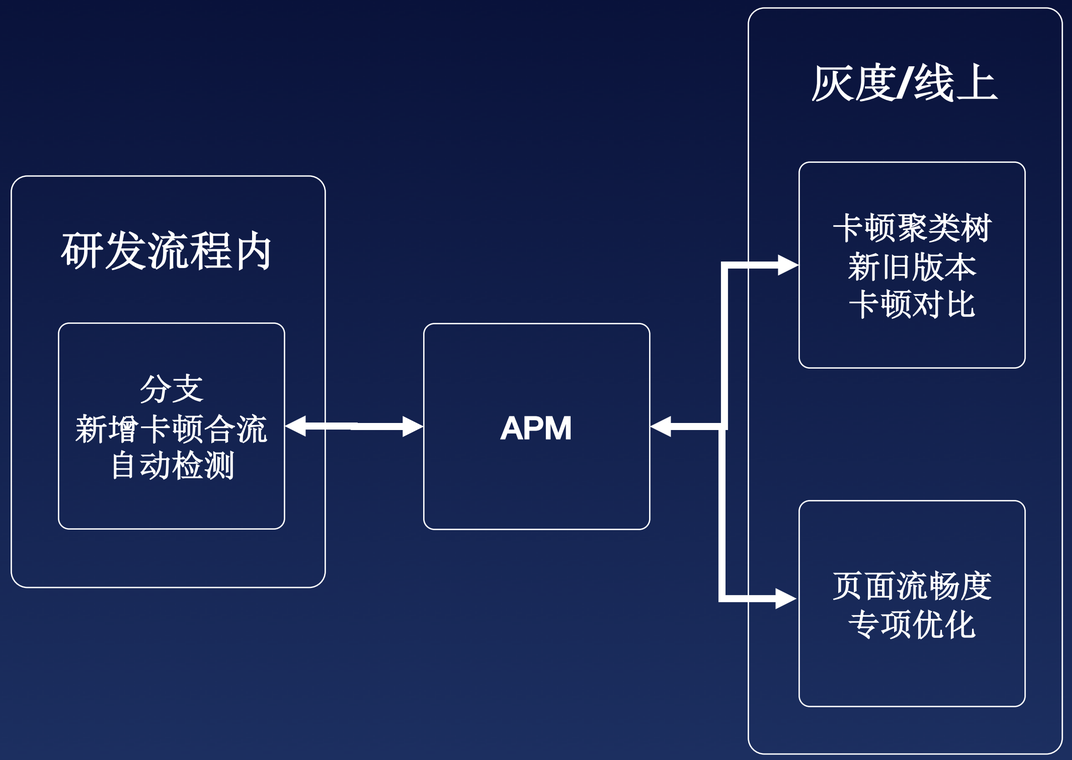
6.GCD并发队列短时间创建大量任务

7.UIWebview初始化、销毁（使用性能更优的WKWebView）

如何在快速的版本迭代中保证App性能不恶化？

三大机制为App性能保驾护航：

1. **分支新增卡顿：**基于分支代码提交记录和卡顿堆栈精准匹配
2. **卡顿聚类树版本卡顿对比：**利用大盘用户的大数据，与历史版本进行对比分析发现新增问题，需要说明的是占比或者卡顿耗时变化较大的也是新增卡顿
3. **页面流畅度专项优化：**建立页面性能负责人机制，对于新版本流畅度下降的页面专项优化



目录

从爆内存典型案例说起

有部分用户反馈升级ios10后手Q启动10秒内必现闪退

问题特点：

难发现：爆内存属于SIGKILL闪退，不可捕获

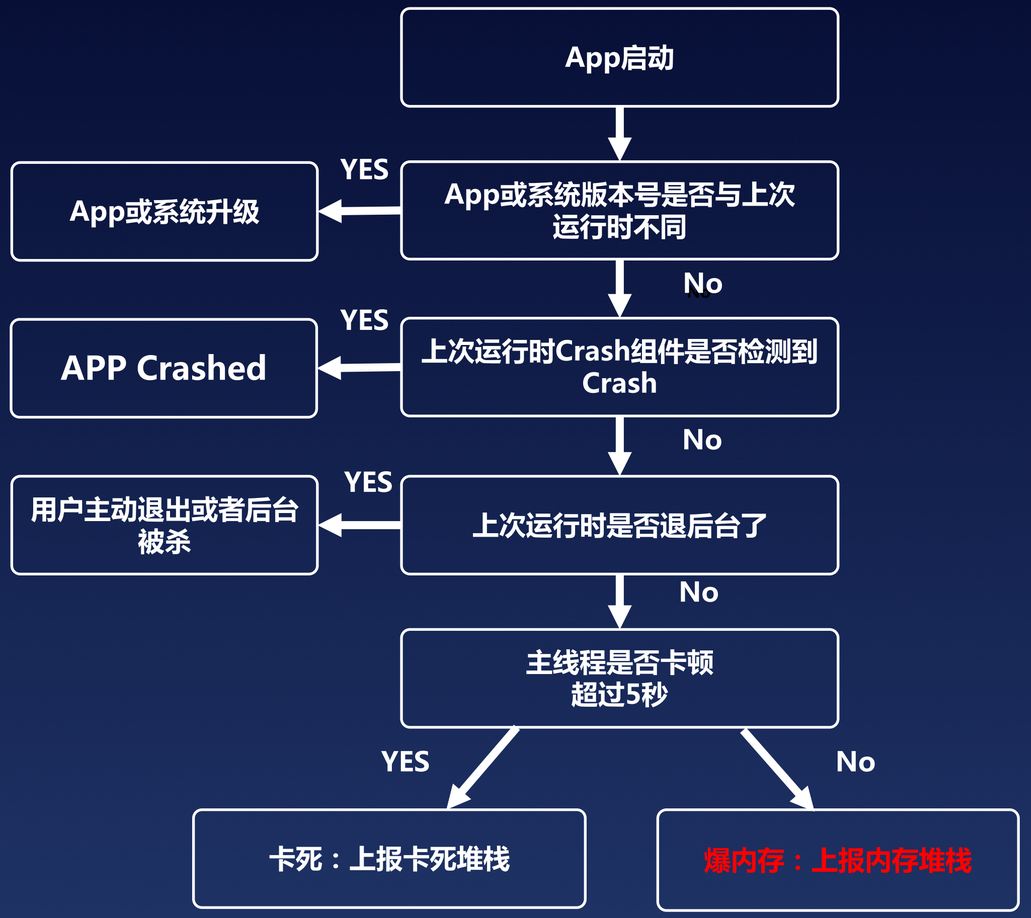
难复现：特定用户+特定场景，内网无法复现

难定位：系统jetsam日志无定位堆栈，无迹可寻

问题原因：

手Q通讯录的循环遍历逻辑中没有加autoreleasepool, 当用户通讯录较多时会积累大量autorelease对象从而引发内存暴涨最终被系统kill

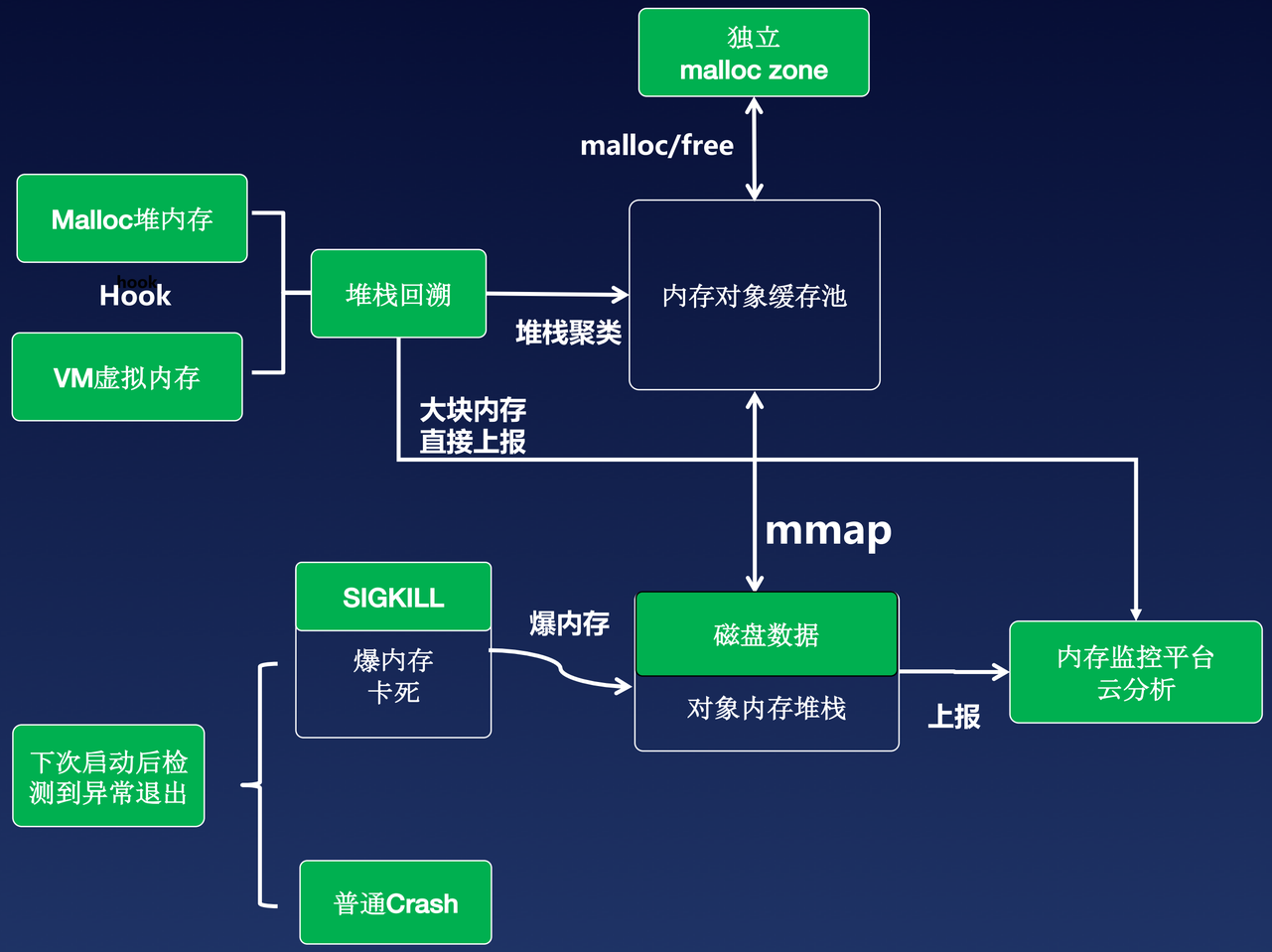
发现爆内存问题：SIGKILL闪退识别



基于Facebook方案，结合实践经验改进点：

1. 优化IO失败导致的误报
2. 优化Crash捕获组件，避免Crash组件内部逻辑引发的二次Crash和内部内存分配导致的卡死
3. 结合卡顿组件，区分卡死情况，上报卡死堆栈定位问题

让爆内存问题“有的放矢”：爆内存堆栈监控方案



方案概述：

Hook所有内存分配方法，记录并抓取所有内存分配的堆栈和内存大小

实现中遇到的问题：

1. 如何Hook内存分配方法？
- 替换全局函数指针malloc_logger

2. 如何解决工具内部内存分配递归死循环问题？
- 使用独立malloc zone，隔离内部内存分配

3. 如何保证堆栈数据及时落地？
- 使用mmap保证数据不丢失

4. 如何保证工具开启不影响用户体验？
- 深度优化工具内存和性能

如何解决内存监控工具自身性能问题？

工具性能优化关键点：

1.使用自旋锁代替互斥锁，提升对象缓存池的访问效率

2.对相同堆栈进行合并压缩，对于内存占用不高的堆栈，只存储压缩后的摘要信息；只有当堆栈内存超过一定阀值时，才存储堆栈的原始信息

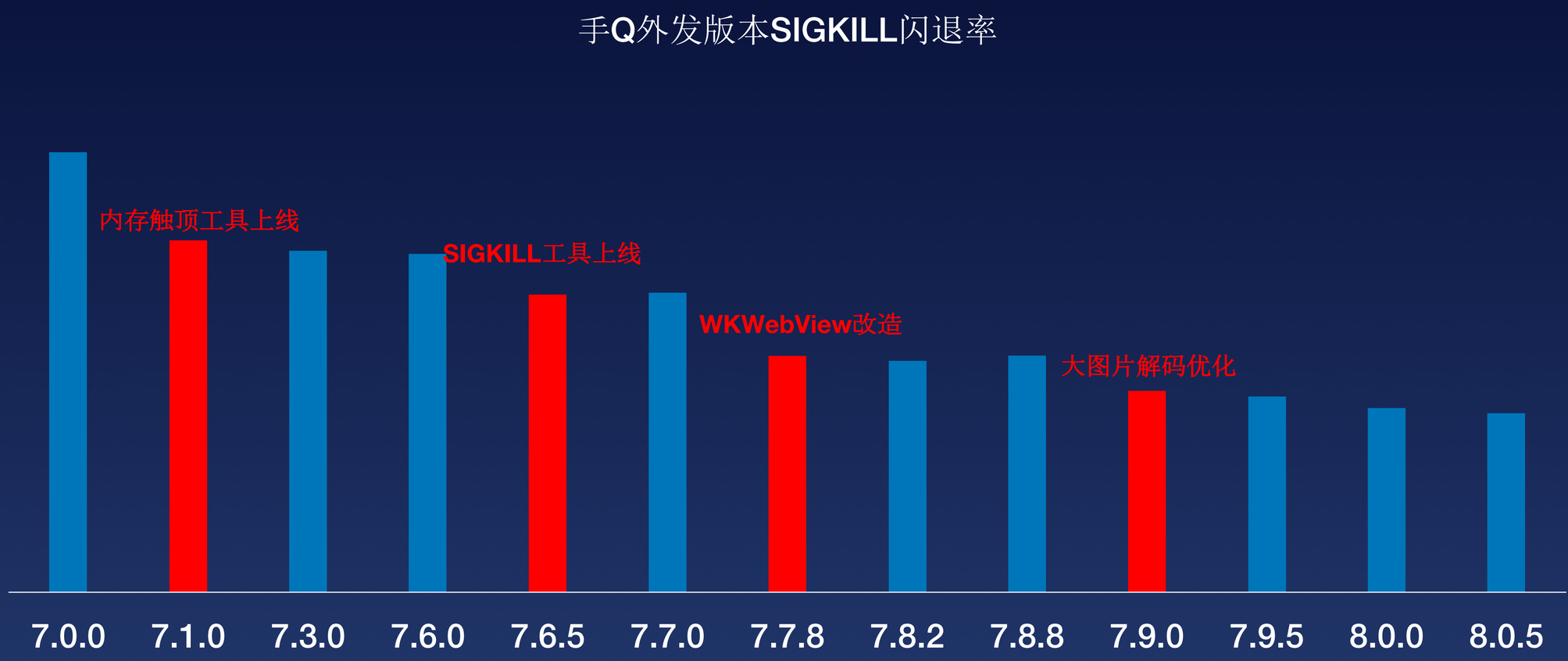
3.优化摘要算法，一开始使用的是md5算法，后来优化为性能和内存都更优的crc64算法

4.抽样策略：对于小块内存抽样采集，大块内存全量采集

iOS 常见内存问题总结

1. **UIWebView**：UIWebView的内存消耗很大，是内存问题的“重灾区”，优化方案是用WKWebView替换
2. **大界面渲染**：指View的size过大或者渲染结构过于复杂的界面渲染，优化方案是降低界面复杂度
3. **Autorelease对象**：由于autorelease对象积累过多导致峰值内存增大爆内存，常见于大循环和GCD串行队列中
4. **大图片解码**：这是一类比较常见的问题，图片size过大导致，优化方法是适当降低图片清晰度和限制大图缓存数量
5. **内存泄漏**：OC对象、malloc内存块泄漏或者C++对象泄漏经过一定时间积累都会导致爆内存

手Q 爆内存闪退率优化效果



目录

影响发热耗电的因素总结

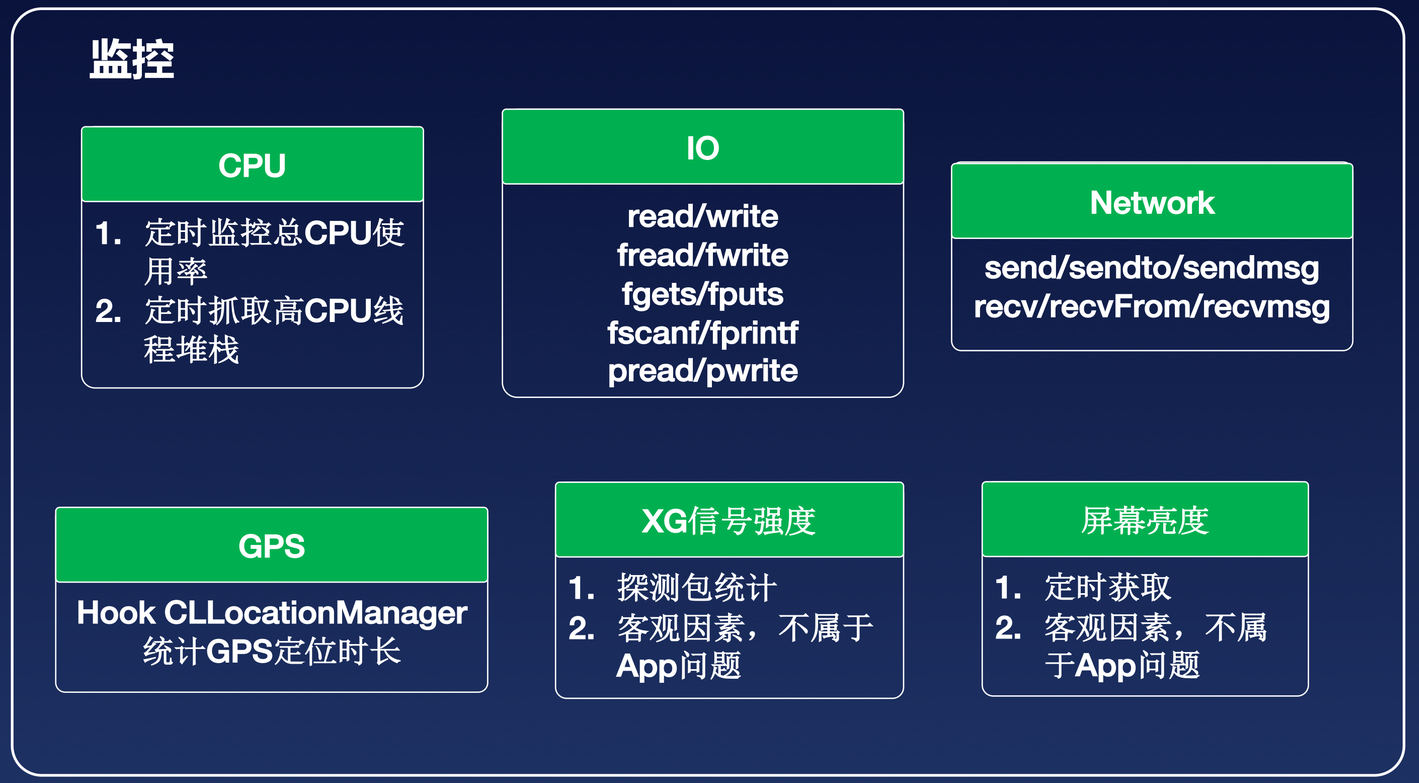
手机由各种电子元器件组成，这些电子元器件在耗电的同时，也会发热，与耗电主要因素总结如下：

1. **CPU使用率**：CPU的功耗是各种元器件中最高的，是引发发热耗电的首要因素
2. **网络流量**：信号较好时，Wifi和XG耗电量基本一致；信号较差时，XG耗电量可达Wifi的12倍
3. **GPS**：GPS定位模块也是耗电大户，持续使用GPS定位1分钟可引发发热
4. **IO总量**：IO设备的频繁使用会一定程度增加系统耗电
5. **屏幕亮度**：屏幕亮度和耗电量成正比，当屏幕亮度最大时耗电量是普通情况下的4倍

硬件模块	耗电1%花费时长	发热时长
CPU(200%)	2.5min	40s
CPU(100%)	5min	2min
CPU(50%)	8min	5min
GPS(搜星过程)	1min	1min
WIFI	5min	-
xG(信号良好)	5min	-
xG(信号差)	40s	1min
屏幕(0.3亮度)	6min	-
屏幕(1.0亮度)	1.5min	-

电量测试数据

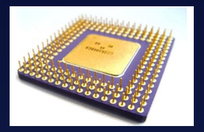
手Q iOS 发热耗电监控方案



监控方案总结：

1. 流量监控：在信令通道和Http通道建立统一监控机制，记录各业务流量
2. GPS监控：GPS模块的权限统一管控，记录各业务使用时长
3. CPU监控：CPU使用超过阀值时通过定时抓取线程堆栈辅助分析
4. IO监控：大文件可直接抓取堆栈，频繁文件读取结合CPU异常辅助分析
5. 其他：XG信号弱和屏幕亮度过亮导致的发热问题属于客观因素，这类问题一般无需跟进

CPU/IO异常堆栈监控策略

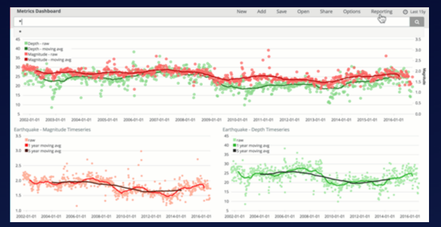


Thread 1	Usage 1
Thread 2	Usage 2
Thread 3	Usage 3
…	…
Thread n	Usage n

CPU使用率

两分钟内平均CPU使用率大于60%

调用栈



辅助定位 | 堆栈抓取 → 调用栈

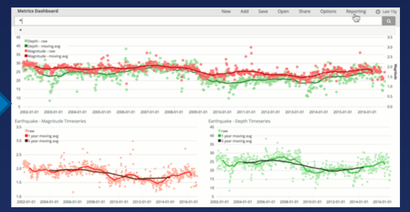
每分钟读写文件总量大于100Mb

HOOK

- open/close
- read/write
- fread/fwrite
- fgets/fputs
- ……

大文件IO调用栈

总IO操作量



CPU问题定位策略：

1.实时统计总CPU使用率

2.CPU总量异常时定时抓取高CPU线程堆栈辅助分析

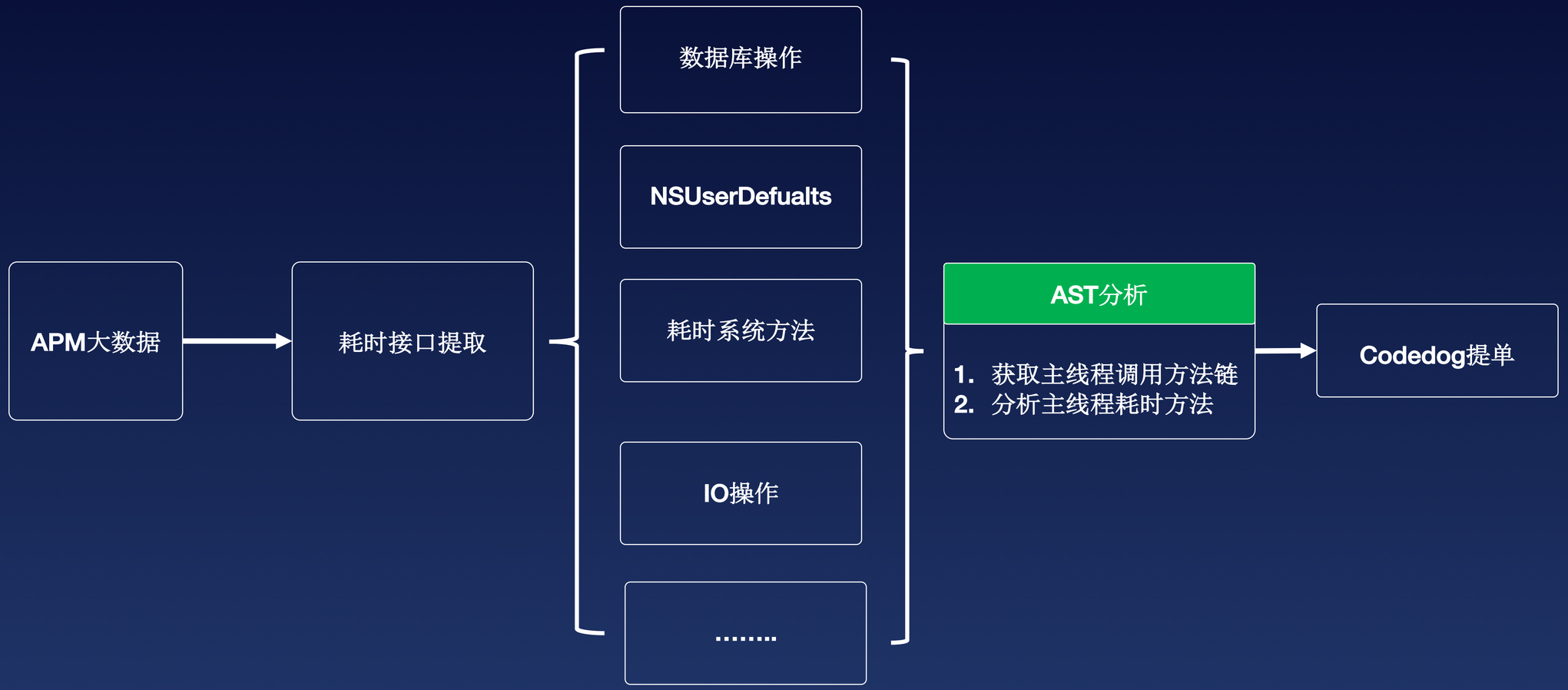
IO问题定位策略：

1.大文件IO直接抓取堆栈

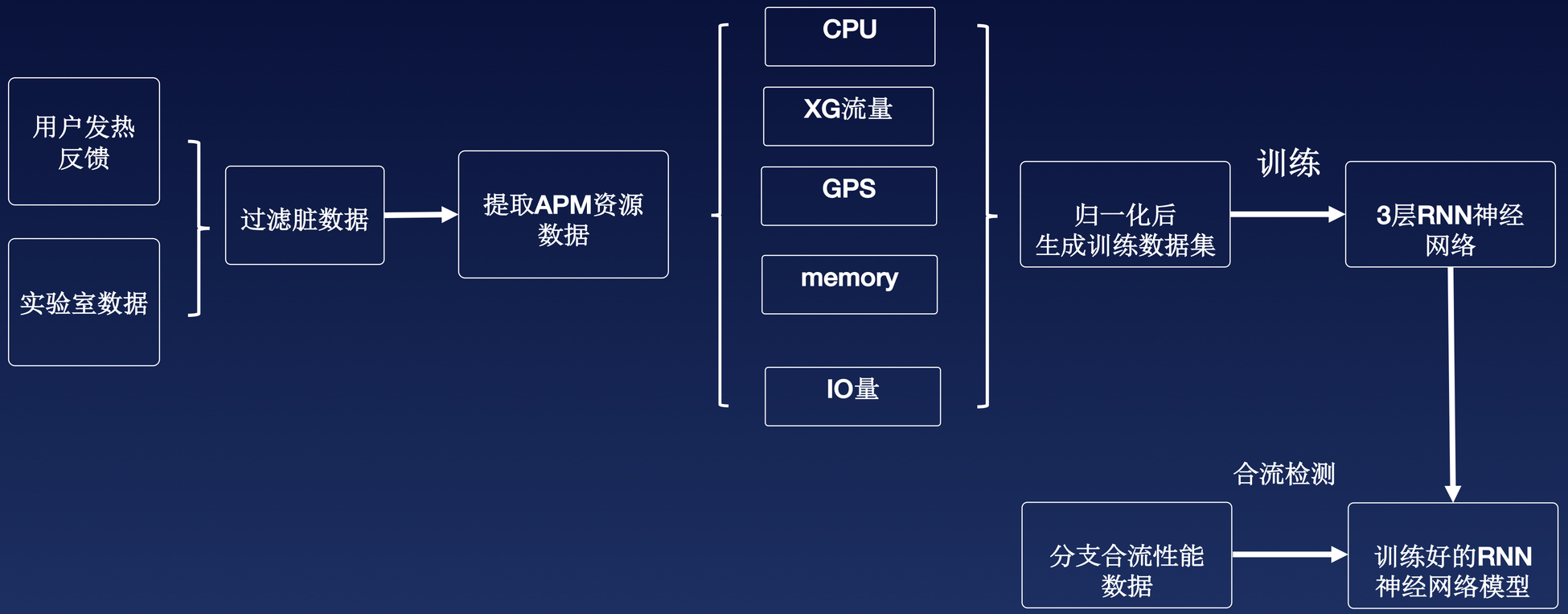
2.IO总量异常定时抓取高CPU线程堆栈辅助分析

目录

应用大数据静态检测代码耗时



应用机器学习检测发热耗电问题



组件开源：更极致的性能优化

爆内存监控组件：

已开源 (https://github.com/Tencent/OOMDetector)

卡顿+发热监控：

Coming Soon.....

Geekbang. 极客邦科技 | InfoQ 极客传媒

InfoQ官网 全新改版上线

促进软件开发领域知识与创新的传播

关注InfoQ网站
第一时间浏览原创IT新闻资讯

免费下载迷你书
阅读一线开发者的技术干货

THANKS

GMTC
全球大前端技术大会