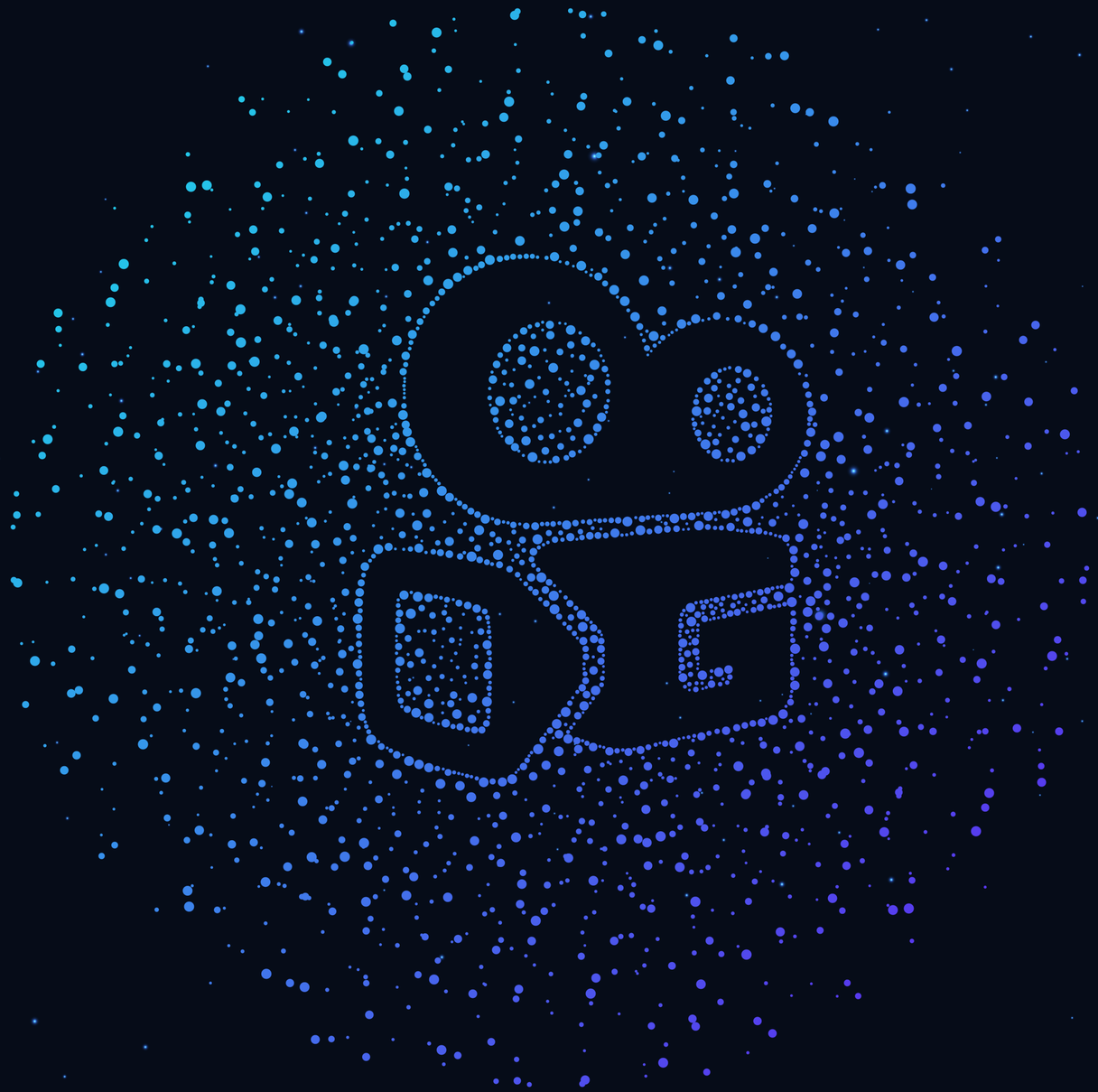
快手稳定性体系建设



薛秋实

Android工程师

退出率监控

什么是稳定性?

- Android： Java crash & Native crash
- iOS：NSException & BSD Signal & Mach EXC
- 指标：session crash率 / 设备crash率

案例

iOS大V主播反馈频繁崩溃，无法正常开播，无崩溃上报，
内存卡顿等数据也无异常，怎么办?

WAKEUPS是什么?

```
Wakeups: 45001 wakeups over the last 142 seconds
(316 wakeups per second average), exceeding limit
of 150 wakeups per second over 300 seconds
```

官方解释

- **WAKEUPS**是EXC_RESOURCE的子类，表示线程唤醒过于频繁，增加耗电
- EXC_RESOURCE是以进程为单位的资源不足异常
- 如果Exception Note field包含**NON-FATAL CONDITION**则不会引起崩溃
- 通常是频繁的**线程间通信**引起的，可以通过日志中的堆栈定位

定位 & 解决

- 通过系统日志中的堆栈定位到粉丝给大V发私信太频繁
- 优化线程间通信不反馈、不上传日志怎么办?
- 优化磁盘读写

监控WAKEUPS

- 定时获取task_interrupt_wakeups
- Hook ulock_wake 抓取调用栈

启发

Java crash & Native crash

NSException & BSD Signal & Mach EXC

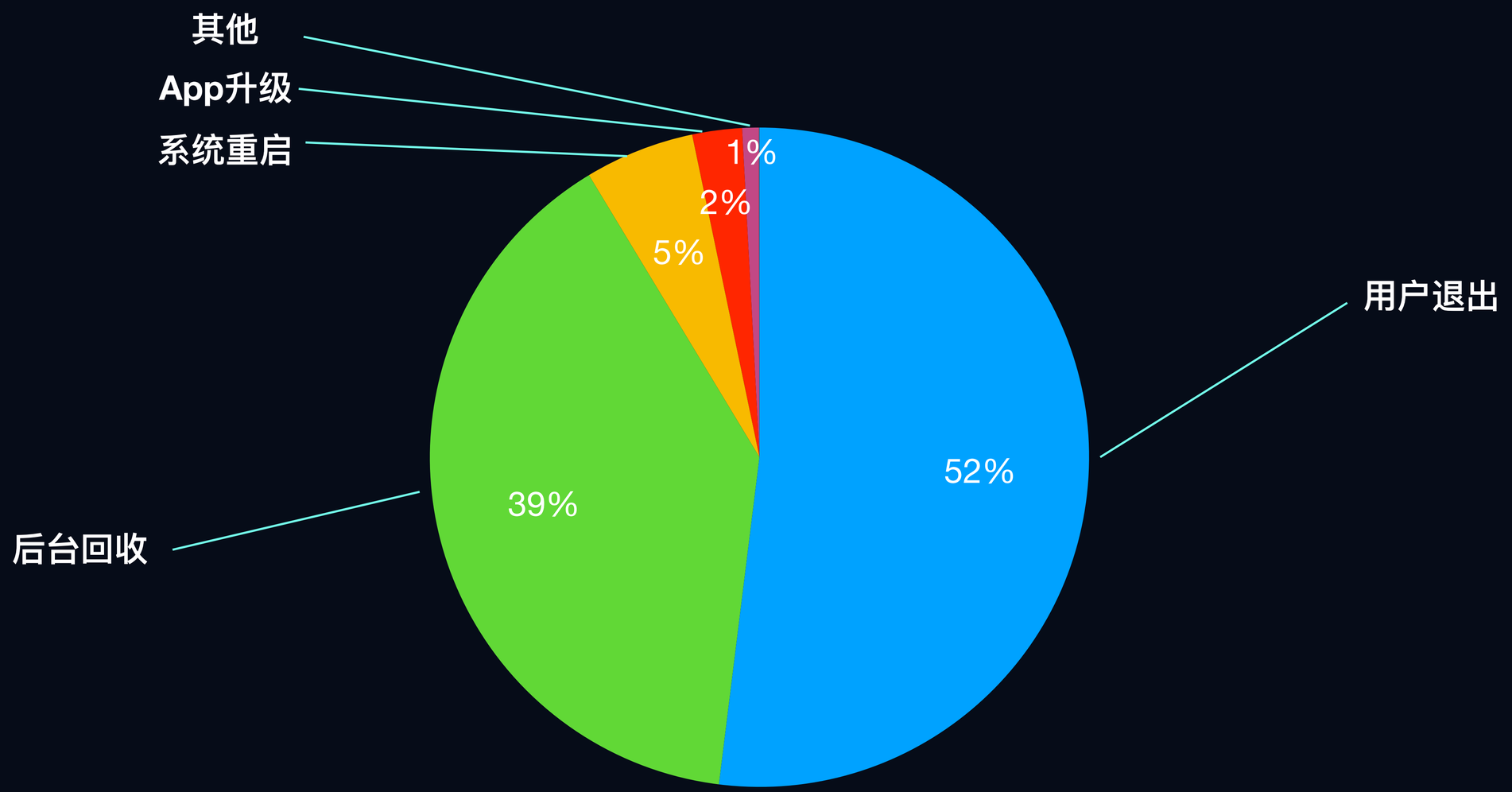
???

退出率定义

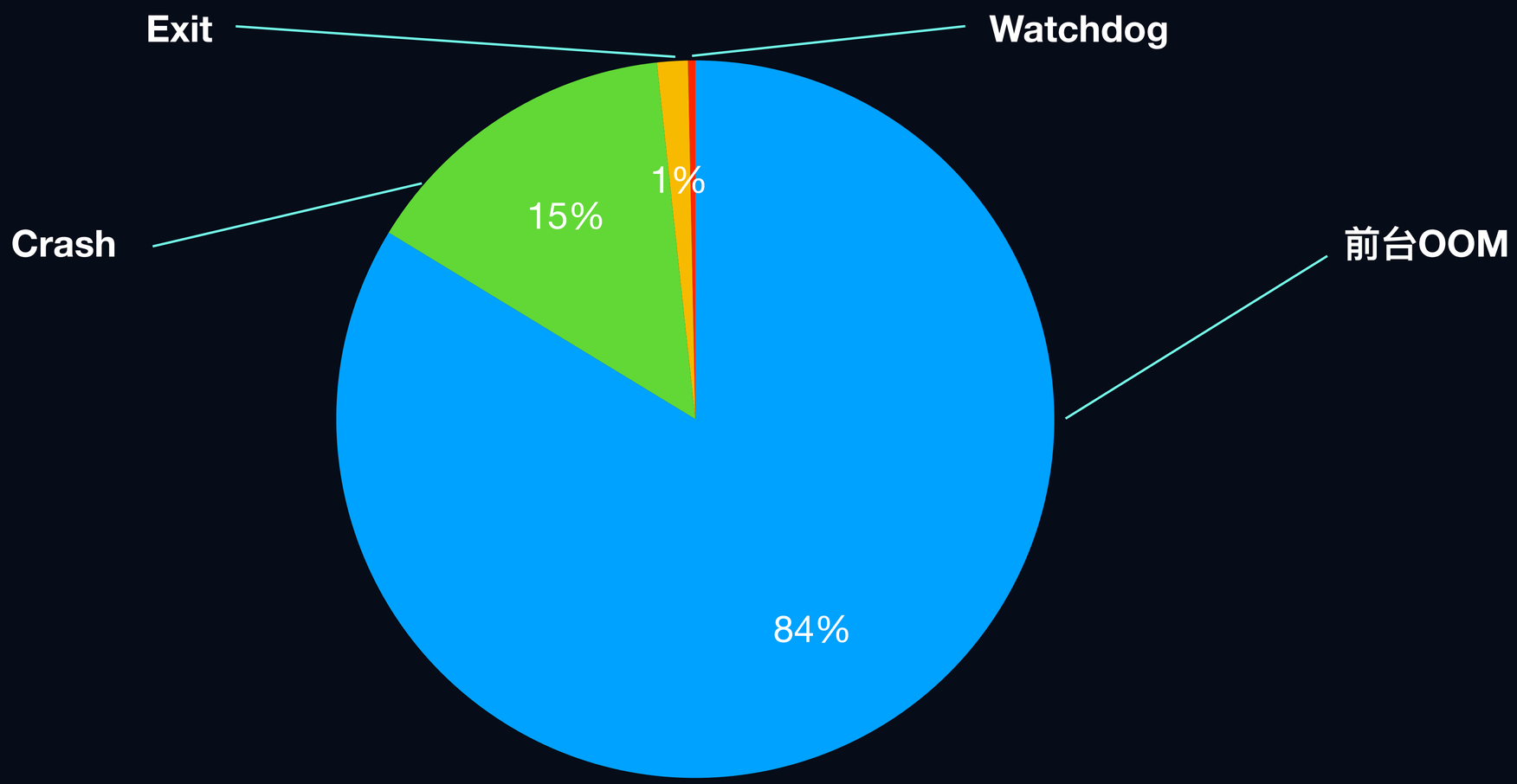
应用退出次数 / 启动数

- Crash
- 前台OOM
- WatchDog
- 主动Exit
- 系统重启
- 用户强杀
- 后台回收
- 系统升级
- App升级
- 其他原因

退出类型占比

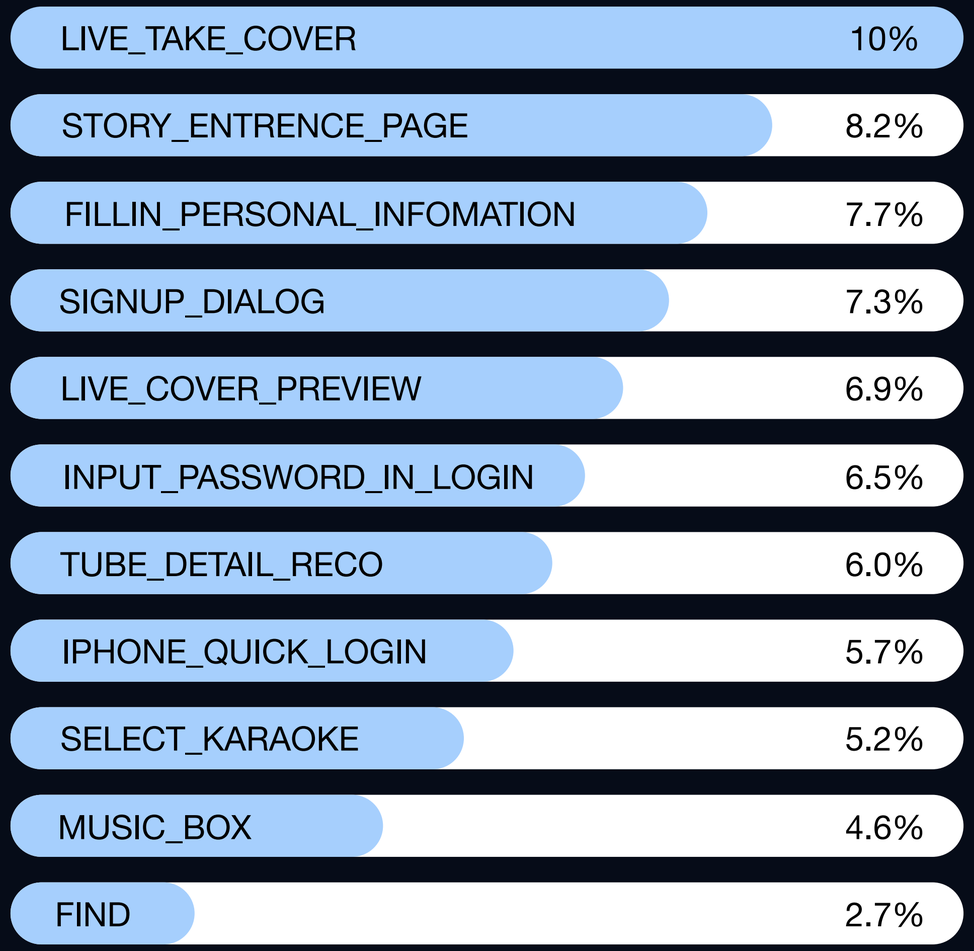


退出类型占比

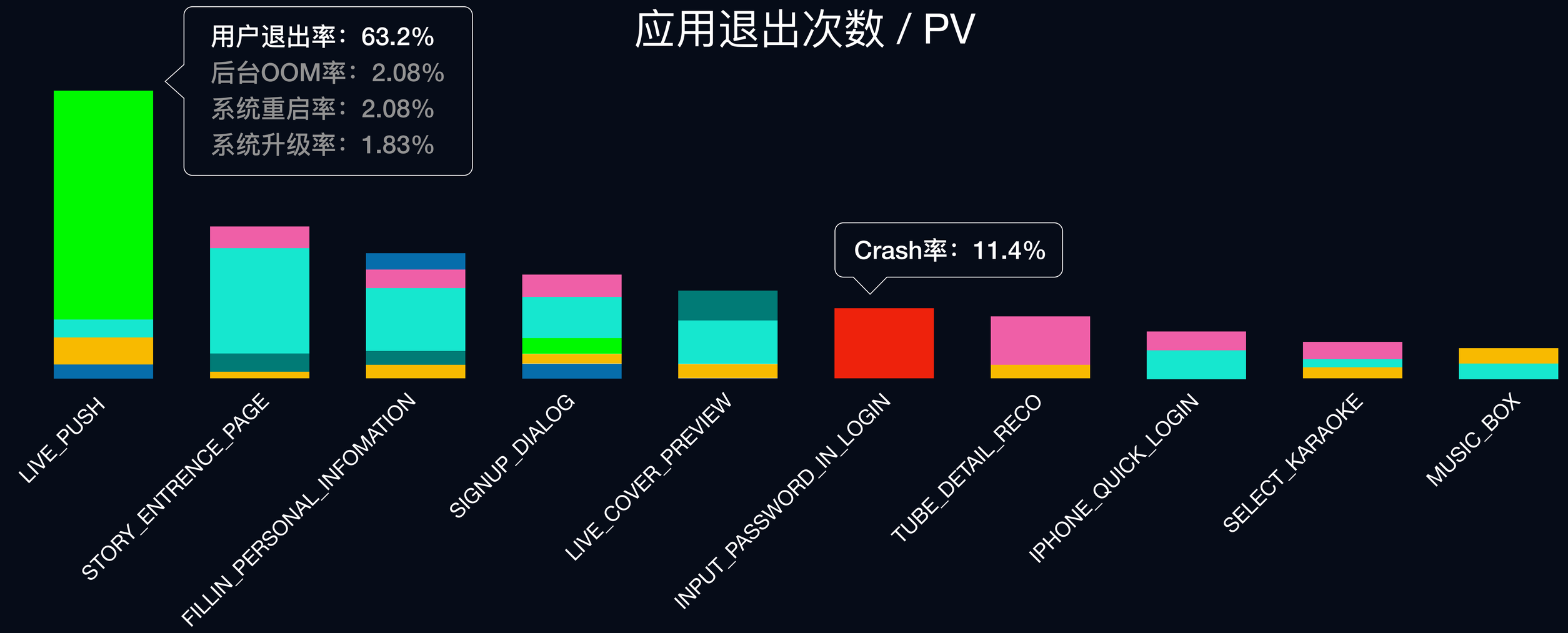


分页面退出率统计

应用退出次数 / PV



分页面退出类型分布


用户退出率：63.2%
后台OOM率：2.08%
系统重启率：2.08%
系统升级率：1.83%
应用退出次数 / PV
Crash率：11.4%
LIVE_PUSH
STORY_ENTRENCE_PAGE
FILLIN_PERSONAL_INFOMATION
SIGNUP_DIALOG
LIVE_COVER_PREVIEW
INPUT_PASSWORD_IN_LOGIN
TUBE_DETAIL_RECO
IPHONE_QUICK_LOGIN
SELECT_KARAOKE
MUSIC_BOX

小结

- WAKEUPS
- 退出率
- 分页面退出率

Android OOM 治理

什么是OOM?

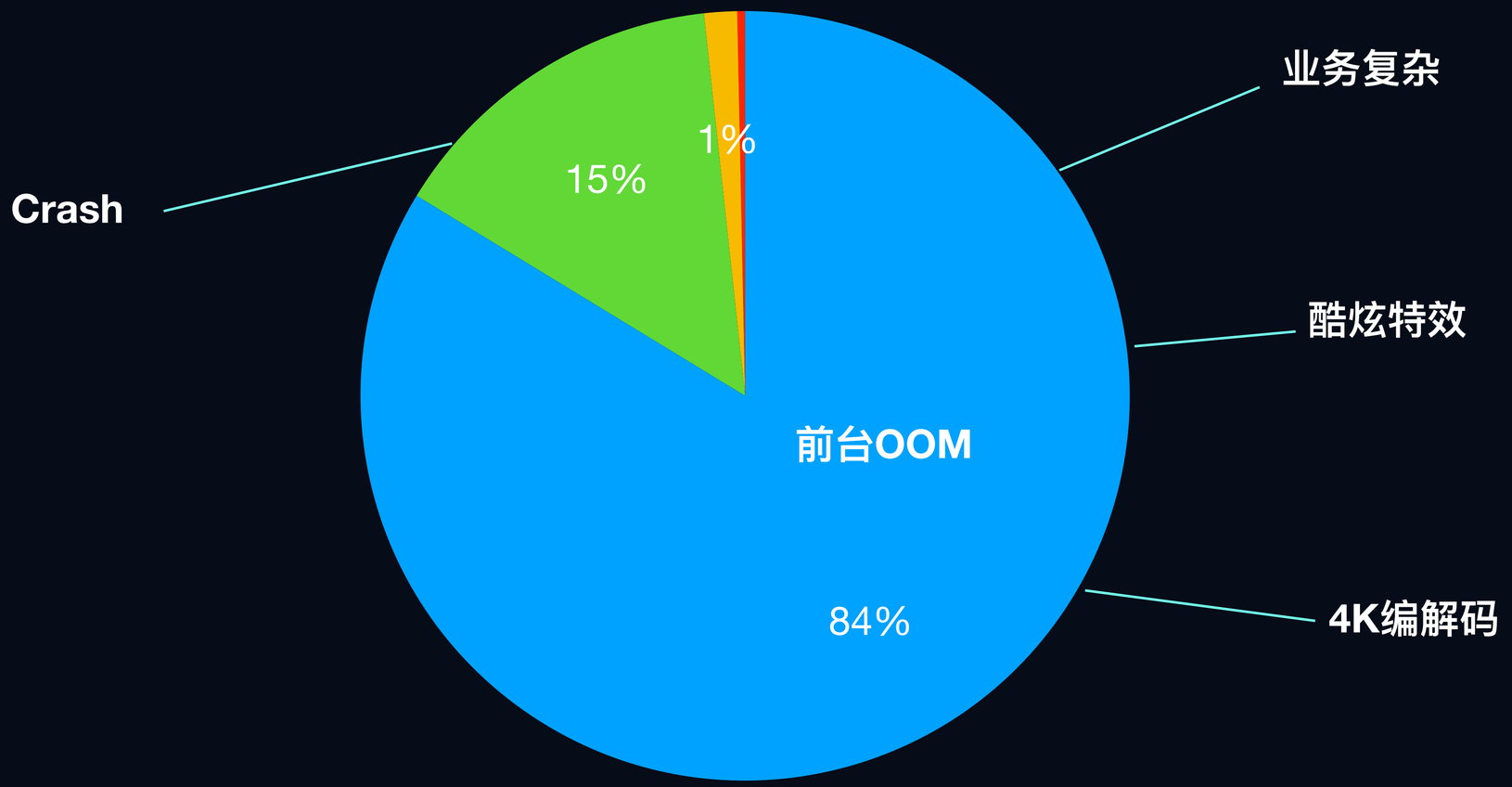
```
android.database.CursorWindowAllocationException: Could not allocate
CursorWindow ‘/data/user/0/com.smile.gifmaker/databases/
real_time_reporting_v2.db' of size 2097152 due to error -12.
    android.database.CursorWindow.nativeCreate(Native Method:0)
    android.database.CursorWindow.<init>(CursorWindow.java:139)
    android.database.CursorWindow.<init>(CursorWindow.java:120)
```

为什么要做OOM治理?

用户体验

- Crash
- 黑屏、白屏
- 卡顿

为什么要做OOM治理?



为什么要做OOM治理?

研发成本

- 人力成本
    1. 线下复现问题
    2. 提交记录二分查找
- 时间成本
    1. 灰度收集数据
    2. 发版delay

为什么要做OOM治理?

LeakCanary能满足我们的需求么?

LeakCanary有哪些不足?

无法线上部署

- 主动触发GC，造成卡顿
- Dump内存镜像造成app冻结
- 解析镜像成功率低
- 没有上报能力

LeakCanary有哪些不足?

适用范围有限

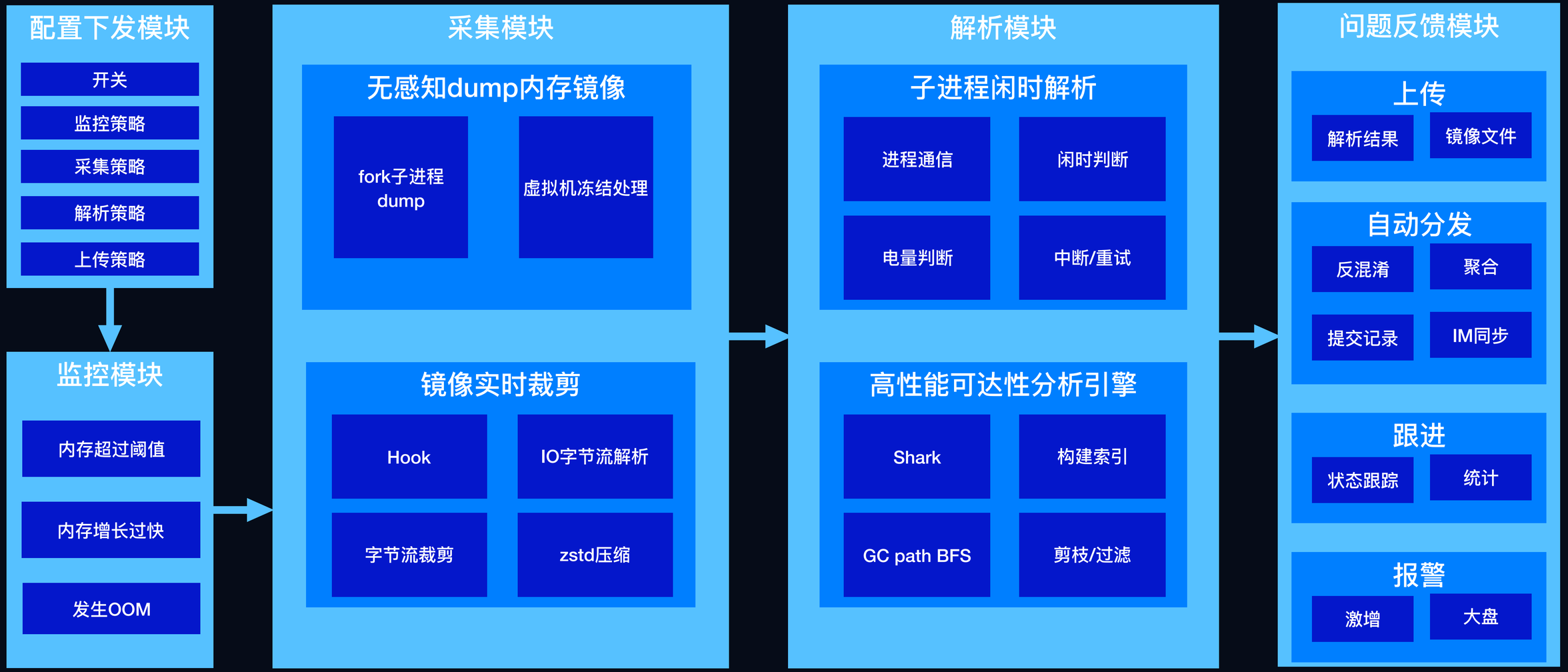
- 只能定位 Activity & Fragment 泄漏
- 无法定位 大对象、频繁分配等问题

LeakCanary有哪些不足?

自动化程度低

- 人工埋点
- 无法对问题聚类

解决方案


配置下发模块
开关
监控策略
采集策略
解析策略
上传策略
监控模块
内存超过阈值
内存增长过快
发生OOM
采集模块
无感知dump内存镜像
fork子进程dump
虚拟机冻结处理
镜像实时裁剪
Hook
IO字节流解析
字节流裁剪
zstd压缩
解析模块
子进程闲时解析
进程通信
闲时判断
电量判断
中断/重试
高性能可达性分析引擎
Shark
构建索引
GC path BFS
剪枝/过滤
问题反馈模块
上传
解析结果
镜像文件
自动分发
反混淆
聚合
提交记录
IM同步
跟进
状态跟踪
统计
报警
激增
大盘

技术难题

- 监控
    1. 主动触发GC造成卡顿
- 采集
    1. Dump内存镜像造成app冻结
    2. 镜像文件过大
- 解析
    1. 耗时
    2. OOM

解决GC卡顿

频繁主动触发GC，用户明显感知到卡顿

解决GC卡顿

- 内存阈值监控

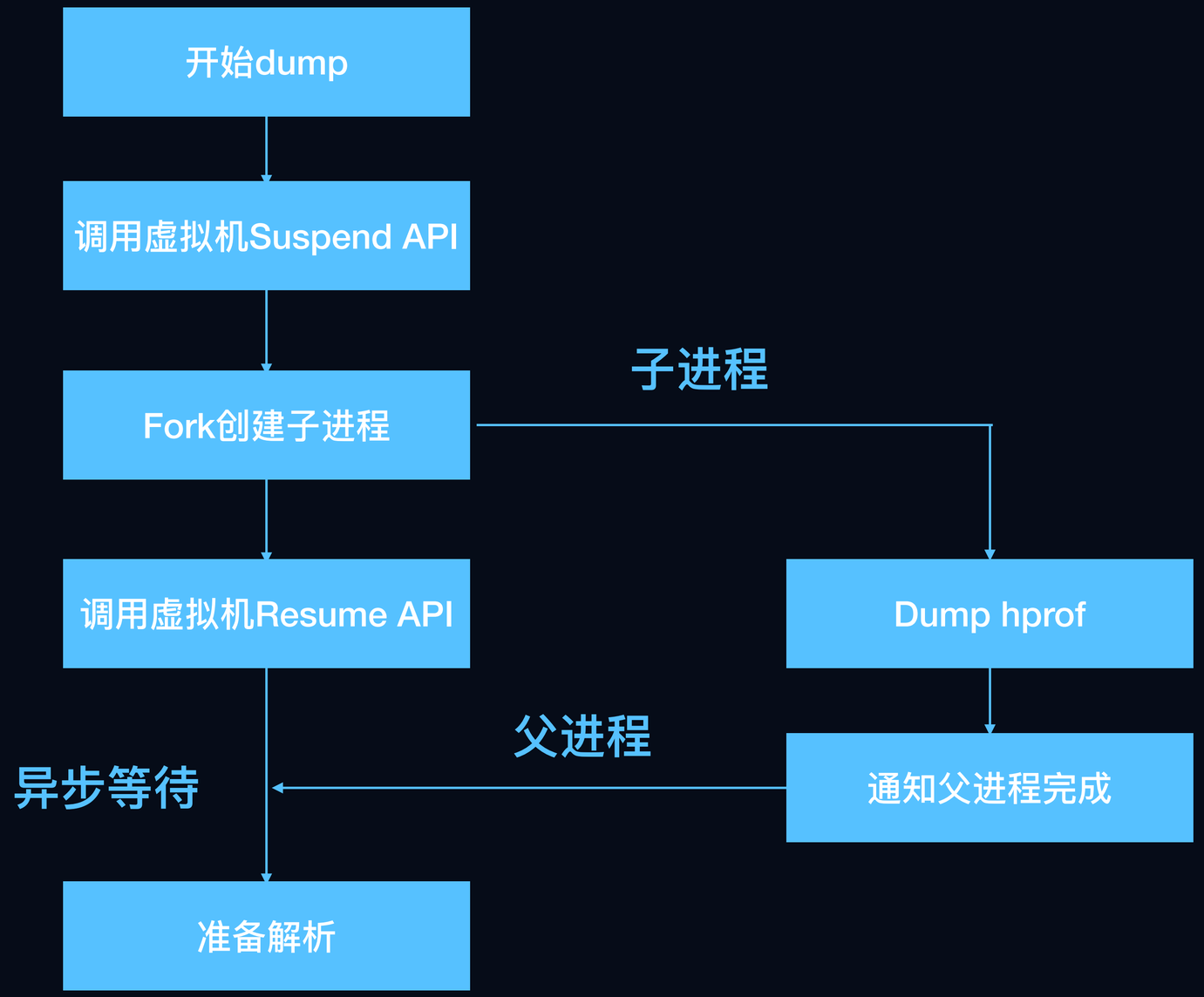
1. Java堆/线程/文件描述符 峰值突破阈值
2. Java堆快速上涨
3. 发生OOM
4. 泄漏判定延迟至解析时

解决app冻结

Dump镜像文件时app冻结长达几秒钟以上，极易触发ANR或用户强杀

为什么会冻结，能否避免呢?

解决app冻结


开始dump
调用虚拟机Suspend API
Fork创建子进程
子进程
调用虚拟机Resume API
Dump hprof
父进程
异步等待
通知父进程完成
准备解析

解决文件过大

为什么能做裁剪?

解决文件过大

- 镜像文件(Hprof)中的冗余数据

1. Zygote Space
2. Image Space
3. 基本类型数组

解决文件过大

怎么裁?

解决文件过大

- 实时裁剪

1. PLT/GOT hook
2. Hook public API open/write
3. 兼容Android 5 ~ 10

解决文件过大


Hook open
打开hprof文件
记录fd
Hook write
根据hprof协议
解析字节流，过滤裁剪数据
裁剪后写入

解决文件过大

- 裁剪成果

1. 裁剪后镜像文件减小10倍

2. 再经压缩后比原始镜像文件减小50倍

解决解析耗时与OOM

镜像解析要做什么？

解决解析耗时与OOM

- 镜像解析

1. 扫描内存中的关键对象

2. 可达性分析生成关键对象到GC ROOT的引用链

解决解析耗时与OOM

端上解析 vs Server解析?

解决解析耗时与OOM

- 端上解析的优点

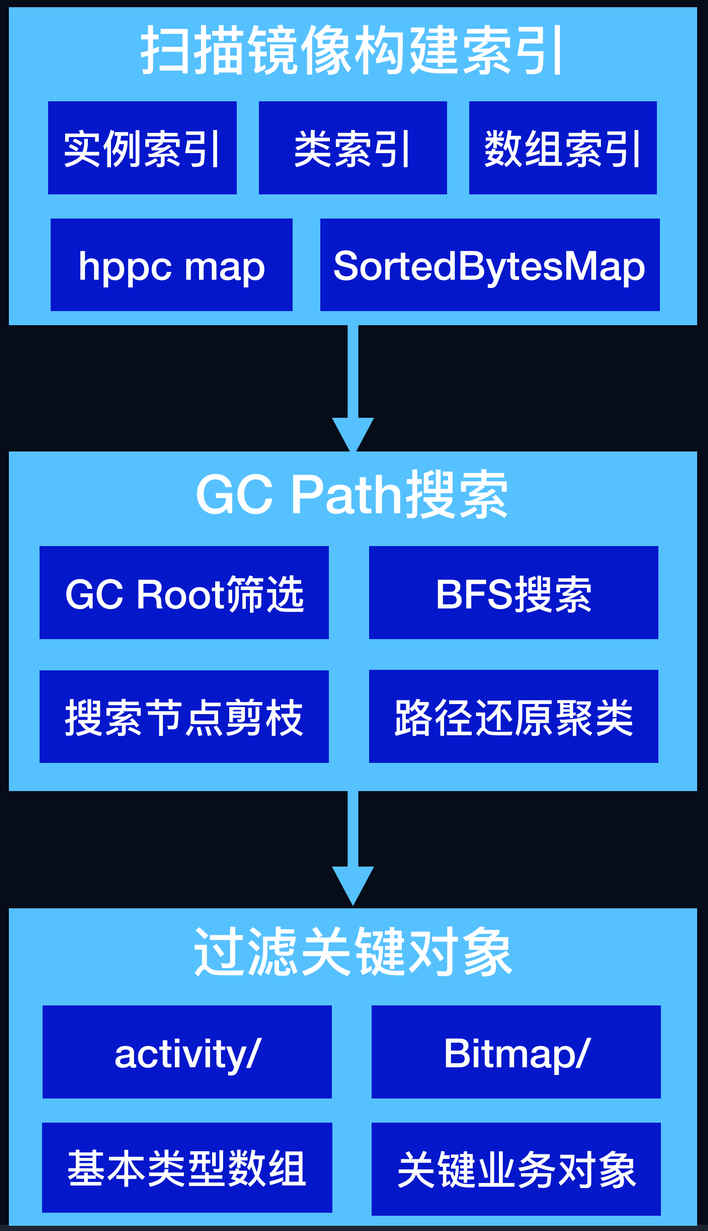
  1. 端上解析不需要上传hprof文件，只上传包含解析结果的json文件，节省用户流量

  2. 利用用户闲时算力分布式计算，降低server压力

解决解析耗时与OOM

如何解决耗时和OOM问题?

解决解析耗时与OOM


扫描镜像构建索引
实例索引
类索引
数组索引
hppc map
SortedBytesMap
GC Path搜索
GC Root筛选
BFS搜索
搜索节点剪枝
路径还原聚类
过滤关键对象
activity/
Bitmap/
基本类型数组
关键业务对象

分发 & 跟进

TITLE	起止版本	最近上报	跟进信息
com.yxcorp.gifshow.detail.PhotoDetailActivity 关键对象	版本追溯		进度 & owner
com.gifshow.kuaishou.preloader.feed.AwesomeCacheCallbackWrapper.callback	7.6.10.14879	2020-07-17	完成
	7.6.10.14941	15:57:36	
[redacted] 7.6.10 必解 头部页面泄漏block发版			

```
com.gifshow.kuaishou.preloader.feed.AwesomeCacheCallbackWrapper.callback   GC ROOT
com.kuaishou.preloader.PageableDataPreloaderKt$config$1.$callback
com.kuaishou.preloader.PageableDataPreloader$callback$1.this$0
com.kuaishou.preloader.LifecycleDataPreloader.lifecycle
androidx.lifecycle.LifecycleRegistry
androidx.arch.core.internal.SafeIterableMap.mStart
androidx.arch.core.internal.SafeIterableMap$Entry.mKey
androidx.activity.ComponentActivity$2.this$0
com.yxcorp.gifshow.detail.PhotoDetailActivity
```

反混淆后的引用链
聚合 & 分发的依据

分发 & 跟进

CLASSNAME	OBJECTCOUNT
com.gifshow.kuaishou.thanos.detail.fragment.ThanosVerticalPhotosFragment	9
android.graphics.Bitmap	12
com.yxcorp.gifshow.detail.PhotoDetailActivity	8

CLASSNAME	SIZE
android.graphics.Bitmap	2073600
android.graphics.Bitmap	2073600
byte[]	4665600
byte[]	2088960

报告中包含关键对象的数量、大小、引用链等信息

Stack 3

com.kuaishou.live.core.show.gift.GiftStore.sGiftImages
android.util.SparseArray.mValues
java.lang.Object[]
android.graphics.Bitmap.mBuffer
byte[]

收获 & 展望

成果

- 通过退出率完善了稳定性体系
- 大幅降低OOM率
- 减少人力投入

收获 & 展望

经验总结

- 遇到未知问题刨根问底
- 抓头部问题(入口页面泄漏)
- 大需求合入前需做对比灰度
- 从根源上解决问题(64位)
- 方案选型做好可用性和性能的平衡

收获 & 展望

Roadmap

- 退出率进一步完善
- 解析性能优化
- 完善问题判定规则
- Native OOM监控

Q & A

快手大前端
技术交流会 2020

THANKS