iOS稳定性治理及性能优化实践

司机端-马克

前言

1.由于在过去几年，货拉拉业务高速发展的同时，作为核心业务入口的司机端，同样在以「快」为第一目标实现业务需求的迭代，积累了较多的技术债（各项技术指标与业界优秀的app相比都差强人意），并且线上经常会收到司机反馈手机发烫，耗电，闪退等等问题；

2.司机使用的手机相比用户来说性能普遍较差，同时司机的在线时长较高，由于以上客观原因的存在，给司机端稳定性带来了巨大的挑战；

综上，稳定性建设迫在眉睫，改善用户体验，降低Crash率成了我们的重要目标。

拉货就找货拉拉

01 司机端现状分析

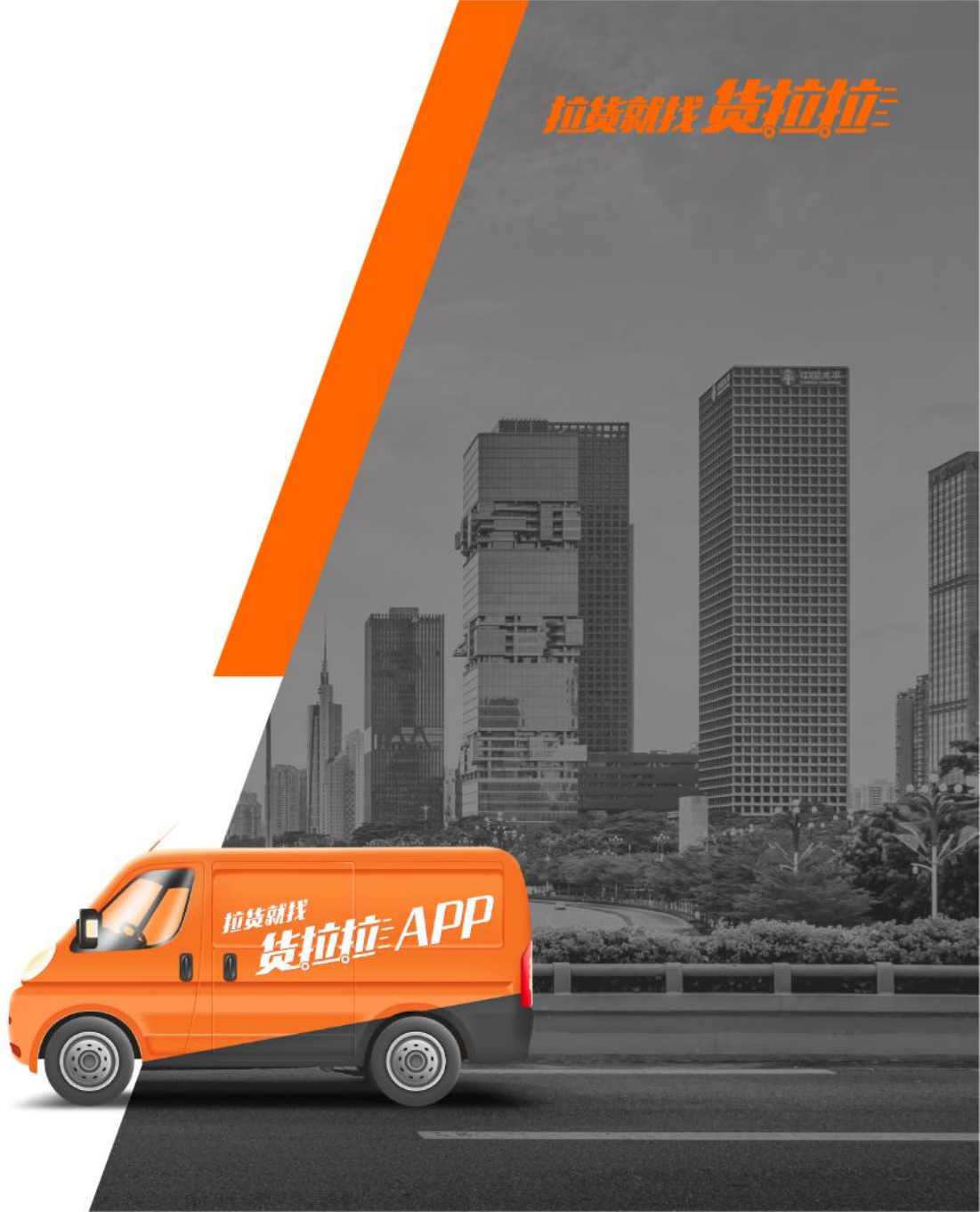
02 线程优化

03 内存优化

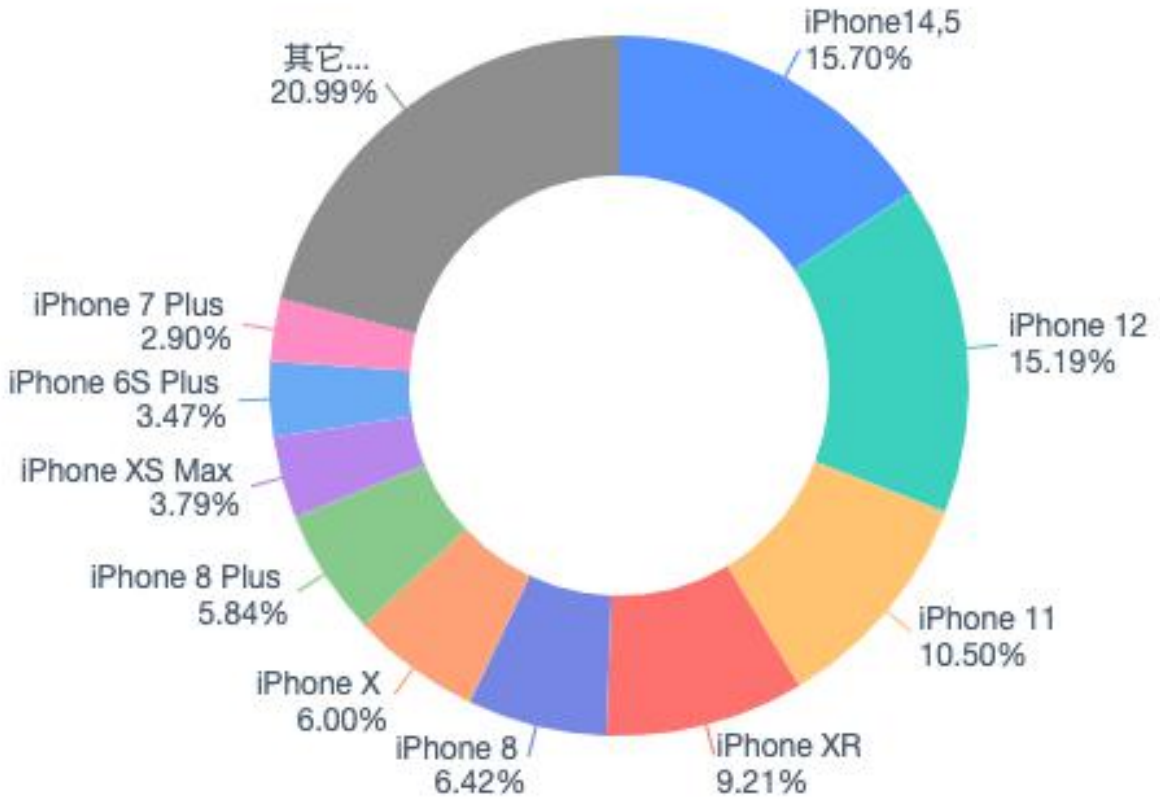
04 疑难问题归因

05 总结回顾

司机端现状分析



现状分析-机型分布



分布特点:
1.iPhone6和iPhone7占比超10%
2.iPhoneX及以下机型占比超30%
3.前三大主力机型分别是iPhone13，iPhone12和iPhone8及Plus

总结:
1.低端机占比高达30%
2.司机钟爱iPhone8及Plus机型

现状分析-Crash时的机型分布

设备分布 TOP5 | 明细

iPhone10,2(iPhone 8 Plus)	42%
iPhone9,2(iPhone 7 Plus)	25%
iPhone10,3(iPhone X Global)	8%
iPhone11,8(iPhone XR)	4%
iPhone12,8(iPhone SE 2nd Gen)	4%

设备分布 TOP5 | 明细

iPhone10,2(iPhone 8 Plus)	72%
iPhone10,5(iPhone 8 Plus)	4%
iPhone12,8(iPhone SE 2nd Gen)	4%
iPhone9,2(iPhone 7 Plus)	2%
iPhone10,3(iPhone X Global)	2%

设备分布 TOP5 | 明细

iPhone10,2(iPhone 8 Plus)	60%
iPhone9,2(iPhone 7 Plus)	9%
iPhone12,8(iPhone SE 2nd Gen)	5%
iPhone10,3(iPhone X Global)	4%
iPhone10,5(iPhone 8 Plus)	4%

分布特点:
1.iPhone8的Crash占比最高
2.iPhone13和iPhone12榜上无名
3.iPhone7iPhone8为代表的低端机，以30%左右的份额贡献了大约80%的Crash

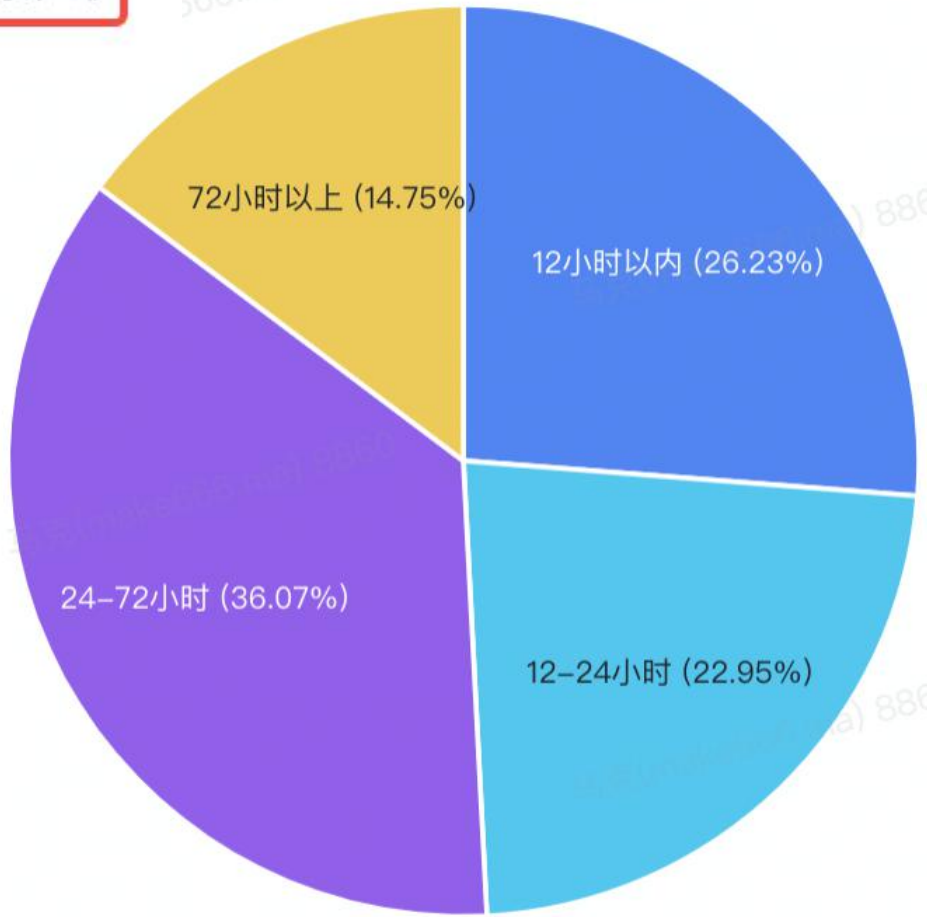
总结：低端机是Crash泛滥的重灾区

现状分析-Crash时使用时长分布

运行状态: 未知

使用时长: 1天18小时50分钟29秒

崩溃线程ID: -



分布特点:
1.使用时长超过10小时的情况高达74%
2.司机app连续在线时间非常长且在前台不息屏
3.工作机，可能边充电边运行

关键词：在线时间长

现状分析-怎么做

先治本

- 线程优化
- 内存优化

后治标

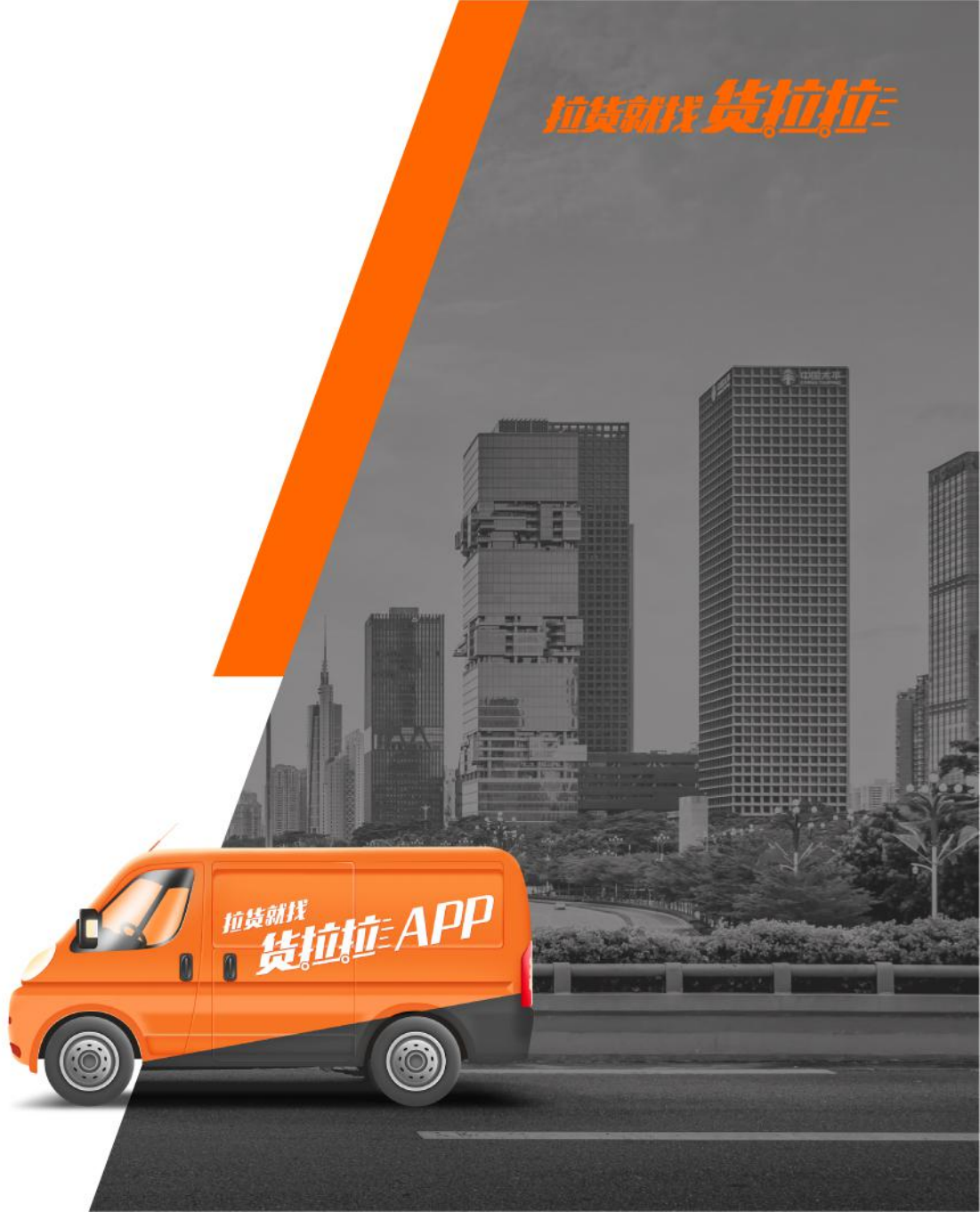
- 野指针
- Watchdog
- OOM
- 疑难Crash

逐步形成体系

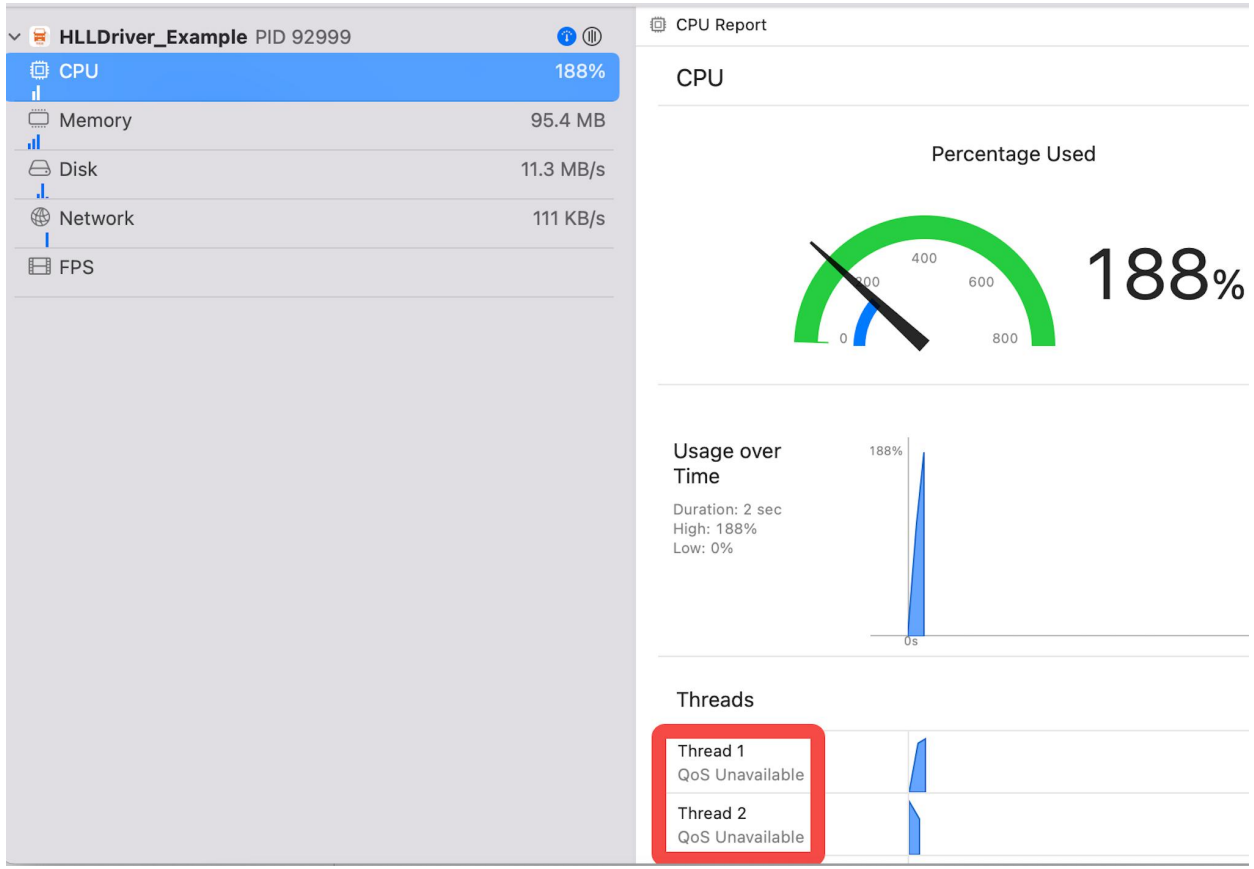
- 监控预警
- 问题归因
- 治理
- 测试验证
- 灰度上线

线程优化

1. CPU调度
2. 任务复杂度
3. 任务优先级



线程优化-CPU调度



发现问题

启动和特定场景并发较高，线程竞争激烈，且没有根据任务区分优先级

影响范围

1.因为CPU超负荷运转，直观的感受就是手机发烫，耗电，卡顿
2.线程竞争激烈，很容易造成线程资源耗尽，触发watchdog
3.如果在主线程初始化了一个高耗时的任务，可能会造成阻塞而触发watchdog
4.如果优先级设置不合理也会造成卡死，触发watchdog

线程优化-CPU调度

解决方案

1.统一线程池，统一调度，兼容低端机
2.创建CPU核数相同的串行队列，控制线程数量
3.可复用的队列，避免产生额外的CPU和内存消耗
4.放弃使用全局队列，并修改使用HLLQueue

```
//与用户交互的任务，这些任务通常跟UI级别的刷新相关，比如动画，cell高度，frame等UI的计算
extern dispatch_queue_t HLLQueueForQoSUserInteractive(void);
//由用户发起的并且需要立即得到结果的任务，比如读取数据（配置，用户信息等）来加载UI，会在几秒或者更短的时间内完成
extern dispatch_queue_t HLLQueueForQoSUserInitiated(void);
//一些耗时的任务，比如复杂的组合的网络请求，图片下载，上传
extern dispatch_queue_t HLLQueueForQoSUtility(void);
//对用户不可见，可以长时间在后台运行，比如，拉取配置，地理位置上报，日志上报等
extern dispatch_queue_t HLLQueueForQoSBackground(void);
//默认，不推荐作为首选使用
extern dispatch_queue_t HLLQueueForQoSDefault(void);
```


线程优化-任务复杂度

0	libsystem_kernel.dylib	_semaphore_wait_trap + 8
2	libdispatch.dylib	__dispatch_semaphore_wait_slow + 136
3	HLLDriver_Example	__42-[AccountManager getDriverCurrentAddress:]_block_invoke (AccountManager.m:208)
4	libdispatch.dylib	__dispatch_call_block_and_release + 24
9	libsystem_pthread.dylib	__pthread_wqthread + 1176

发现问题

当时在排查其它Crash的时候，查看其它线程信息，无意间发现[AccountManager getDriverCurrentAddress:]这个方法，然后随机看了几个crash，基本上都有出现[AccountManager getDriverCurrentAddress:]的身影，而且有并发调用现象，较为频繁

• 线程优化-任务复杂度

```
//实时获取司机位置信息 异步
- (void)getDriverCurrentAddress:(void(^)(HLLAddressComponent *component))complete {
    dispatch_async(dispatch_get_global_queue(0, 0), ^{
        // step1 异步获取当前最新坐标
        // step2 异步坐标转换
        // step3 异步逆地理编码
        // step4 异步获取cityId
        //信号量同步，等全部完成后回调
        dispatch_async(dispatch_get_main_queue(), ^{
            complete(xxx);
        });
    });
}
```

问题原因

1. 在抢单大厅多个接口，依赖此方法异步返回的结果
2. 内部又会通过全局队列生成很多线程处理任务
3. 任务较为耗时，而且是同步完成后进行回调

线程优化-任务复杂度

解决方案

1. 抢单大厅相关的业务方面进行重构，降低了调用的并发
2. 内部去掉了全局队列的使用
3. 使用缓存，外部访问时直接调用属性即可，同时同步最新数据，任务执行保证唯一性

```
- (void)setLatestComponent:(HLLLocationComponent *)latestComponent
{
    //...
    _latestComponent = latestComponent;
}

- (HLLLocationComponent *)latestComponent
{
    //...
    return _latestComponent;
}
```


线程优化-任务优先级

问题原因

当任务设置的优先级不合理，就会出现主线程等待子线程的情况。极易引发watchdog
1.优先级反转: 对高优先级的滥用，导致低优先级的任务迟迟无法完成，造成卡死
2.主线程阻塞: 由于任务复杂度的评估不足，给了过高的优先级，而造成主线程阻塞

解决方案

1.启动初始化时，对所有二三方库全部检查整改，强制异步调用且使用HLLQueue，对部分不紧急的初始化进行延迟加载
2.根据场景和任务复杂度，让业务重新评估优先级使用的合理性
3.拉取配置，日志上报，埋点等数据，使用较低的优先级。

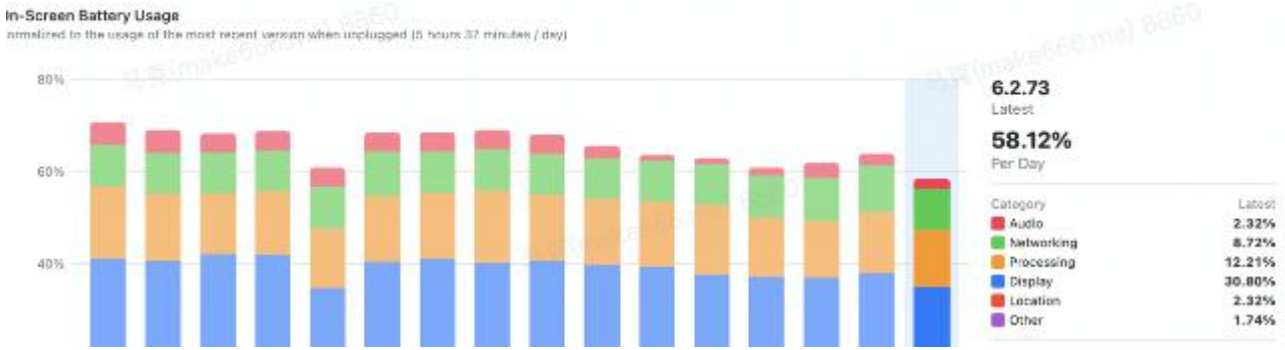
线程优化-小结

- 统一线程池的使用
- 对业务复杂度高的任务，进行了梳理和重构
- 对重点业务场景的优先级使用进行重估
- 放弃对全局队列的使用

收益：
1.CPU/GPU能耗下降2.38个百分点，同比下降了约16.5%
2.网络能耗，下降了1.89个百分点，同比下降了约17.8%
3.线程数量降低了大约18.7%

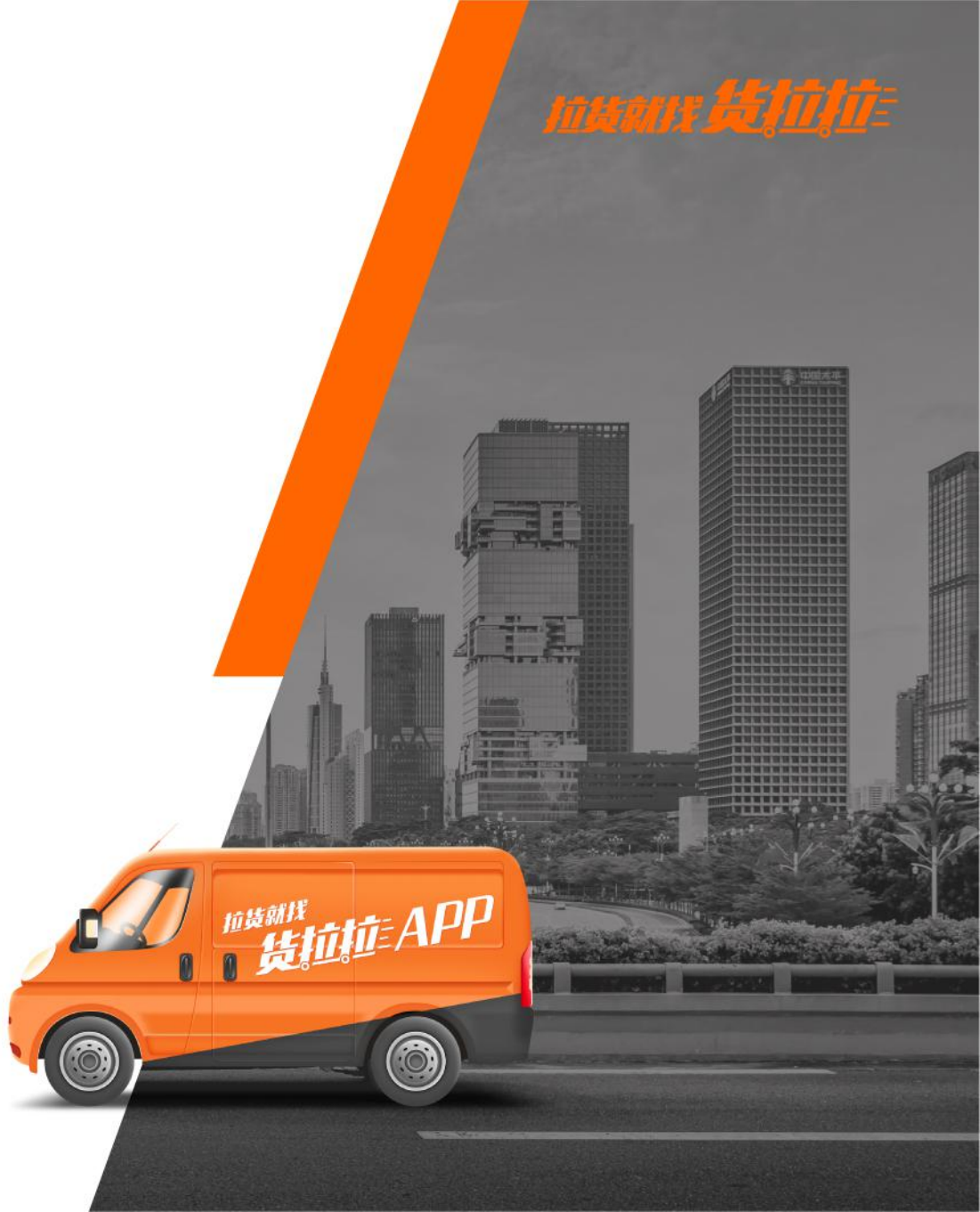
Processing (14.59% -> 12.21%)：CPU/GPU 计算能耗，下降2.38%

Networking (10.61% -> 8.72%)：蜂窝网络、WiFi网络耗电，下降1.89%



内存优化

1. 内存溢出
2. 内存泄漏
3. 内存警告



• 内存优化-内存溢出

设备机型 iPhone10,2(iPhone 8 Plus)

运行状态: 前台

崩溃线程名: -

使用时长: 1天7小时20分钟24秒

错误堆栈 页面跟踪 跟踪日志 符号表 扩展信息

解析 原始

```
1  // remark
2
3  Exception Type:  EXC_CRASH (SIGABRT)
4  Exception Codes: 0x00000000 at 0x0000000000000000
5  Crashed Thread:  34
6
7  Application Specific Information:
8  *** Terminating app due to uncaught exception 'std::bad_alloc', reason: 'std::bad_alloc'
```

错误堆栈 页面跟踪 跟踪日志 符号表 扩展信息

```
1  2022-12-21 10:33:38 [ViewDidAppear]HLLCameraStickerVC
2  2022-12-21 10:33:37 [ViewWillDisappear]HLLDriverWebVC
3  2022-12-21 10:33:37 [ViewWillDisappear]HLLTabBarController
4  2022-12-21 10:33:37 [ViewDidLoad]HLLCameraStickerVC
5  2022-12-21 10:33:31 [ViewDidAppear]HLLDriverWebVC
```


内存优化-内存溢出

问题原因

可用内存不足或者大块连续内存的申请失败，抛出异常

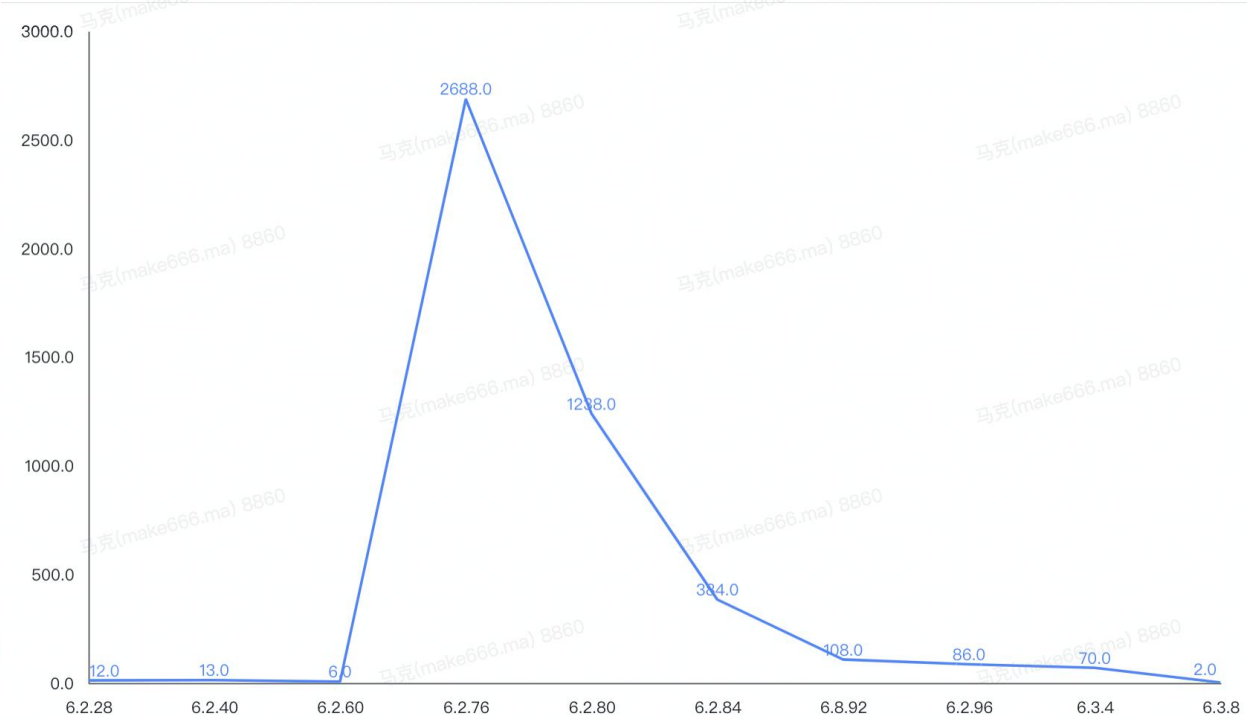
问题分析

1. 因为可用内存空间不足，申请了较大的内存，而失败，由c++抛出的异常。
2. 申请大块且连续内存的内存时，被系统认定为异常，抛出的错误。
3. 长时间的运行，内存泄漏也会导致内存触顶

内存优化-内存溢出

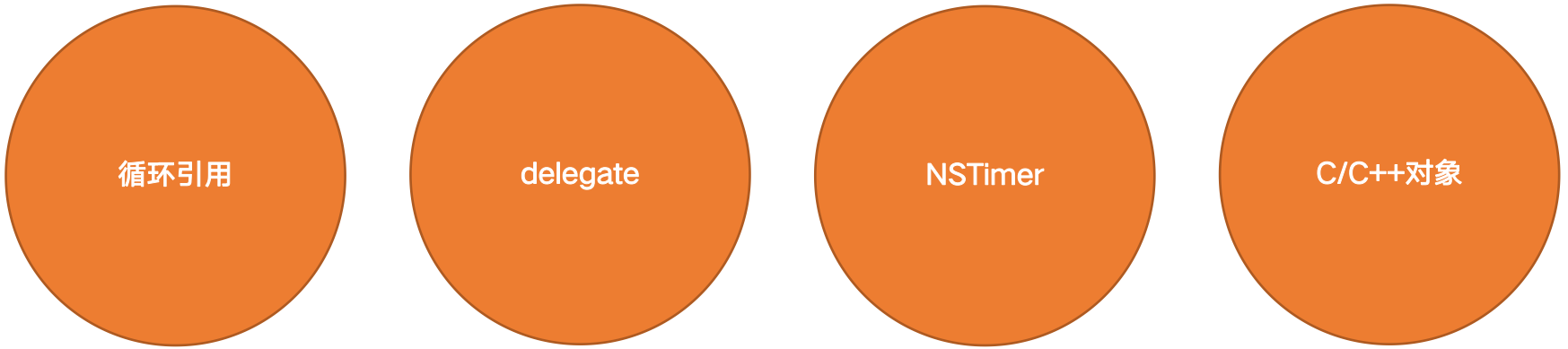
解决方案

1.对大块内存的分配，通过try-catch捕获异常进行上报
2.在局部申请连续内存时，添加autoreleasePool，防止局部申请内存过大
3.针对地图加载，做单页面等大内存页面进行了重构和拆分
4.检查从c/c++相关的对象调用，内存释放是否正常



内存优化-内存泄漏

最常见引起内存泄漏的原因



内存优化-内存警告

01 清理缓存

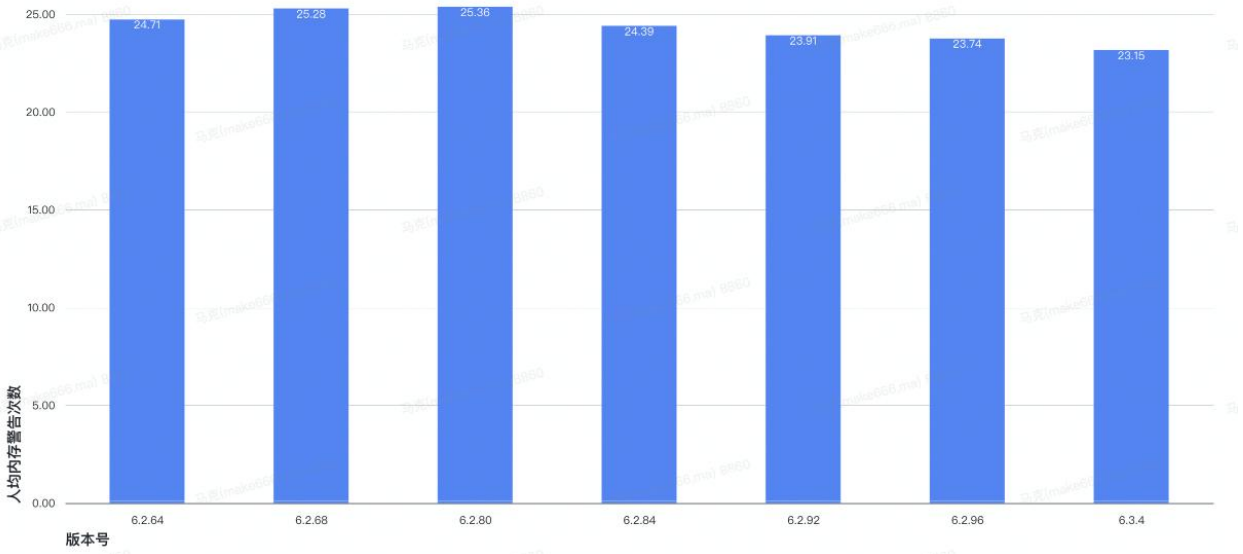
02 释放单例

03 上报

04 线上内存泄漏监控

内存优化-小结

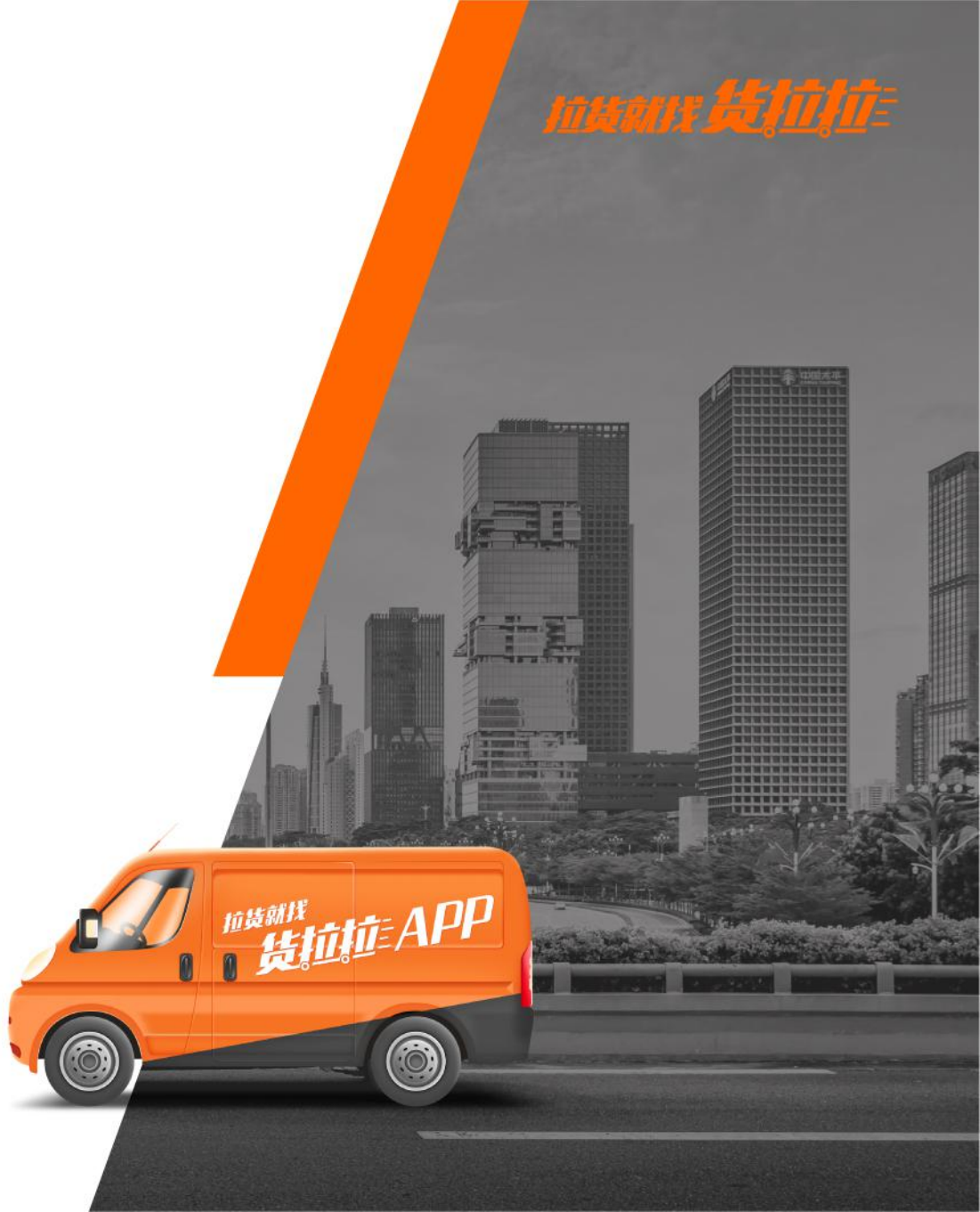
- 对大块内存方面的业务进行了重构，拆分，对于无法拆分的系统功能，添加了try-catch来捕捉异常
- 对于局部连续申请大块内存的地方，添加了autoreleasePool
- 检查内存泄漏，业务统一管理定时器，重点排查了c/c++对象的使用情况
- 线上增加了内存预警小工具，可帮助锁定内存增加的页面
- 收到内存警告的处理



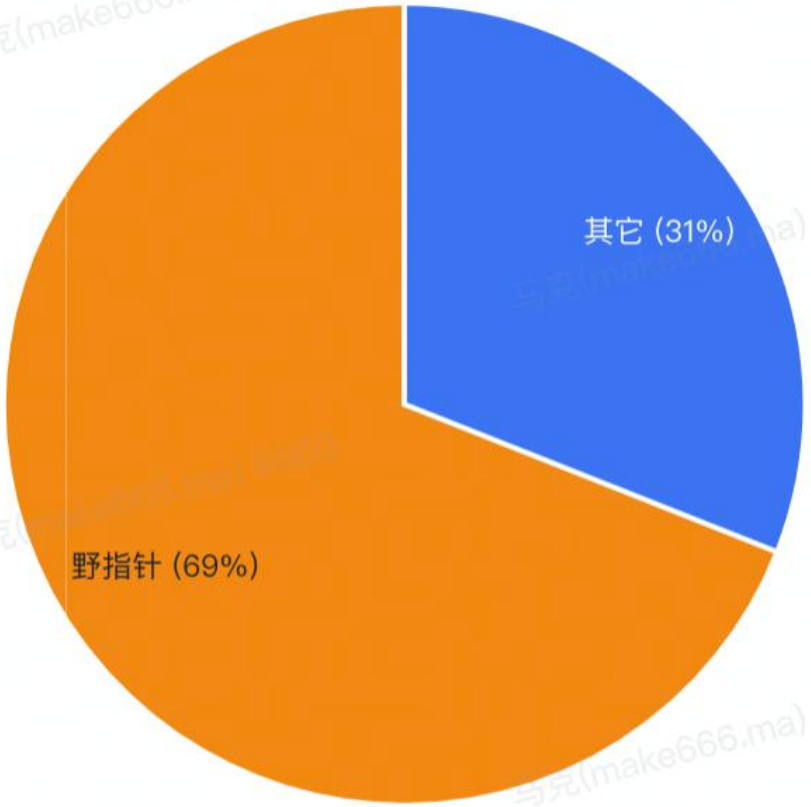
收益：
1.std::bad_alloc的crash从日均200+降低到了个位数
2.抢单大厅的运行内存从140M降低到90M
3.人均内存警告次数从25.36降低到了23.15，大约降低了10%

疑难问题归因

1. 野指针
2. Watchdog
3. OOM
4. 疑难Crash



疑难问题归因-野指针

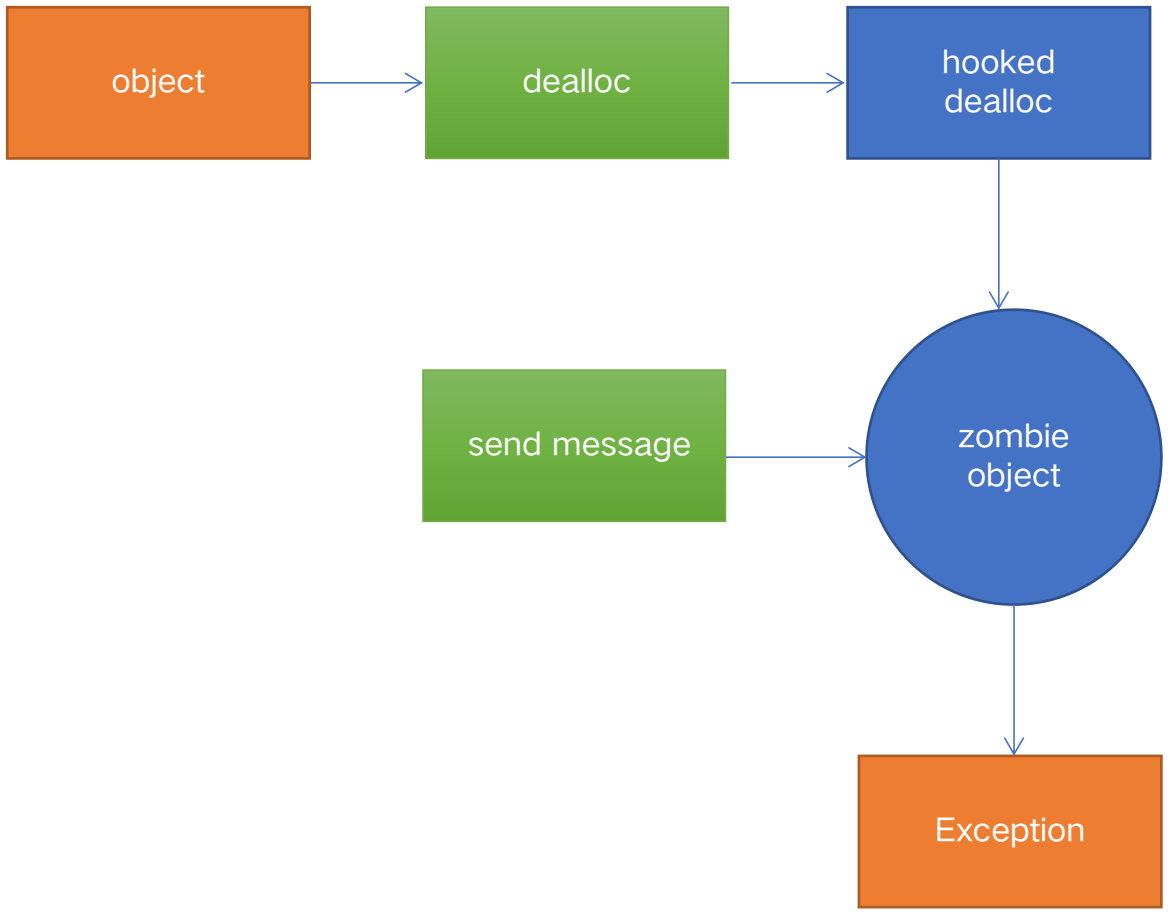


- 统计了现有的crash数据，发现内存错误占比超过2/3
- 表现形式多为Mach或者signal异常

归因难点：

- 崩溃栈并非第一现场，纯系统调用栈
- 分布广，难聚合
- 随机性强难定位

疑难问题归因-野指针

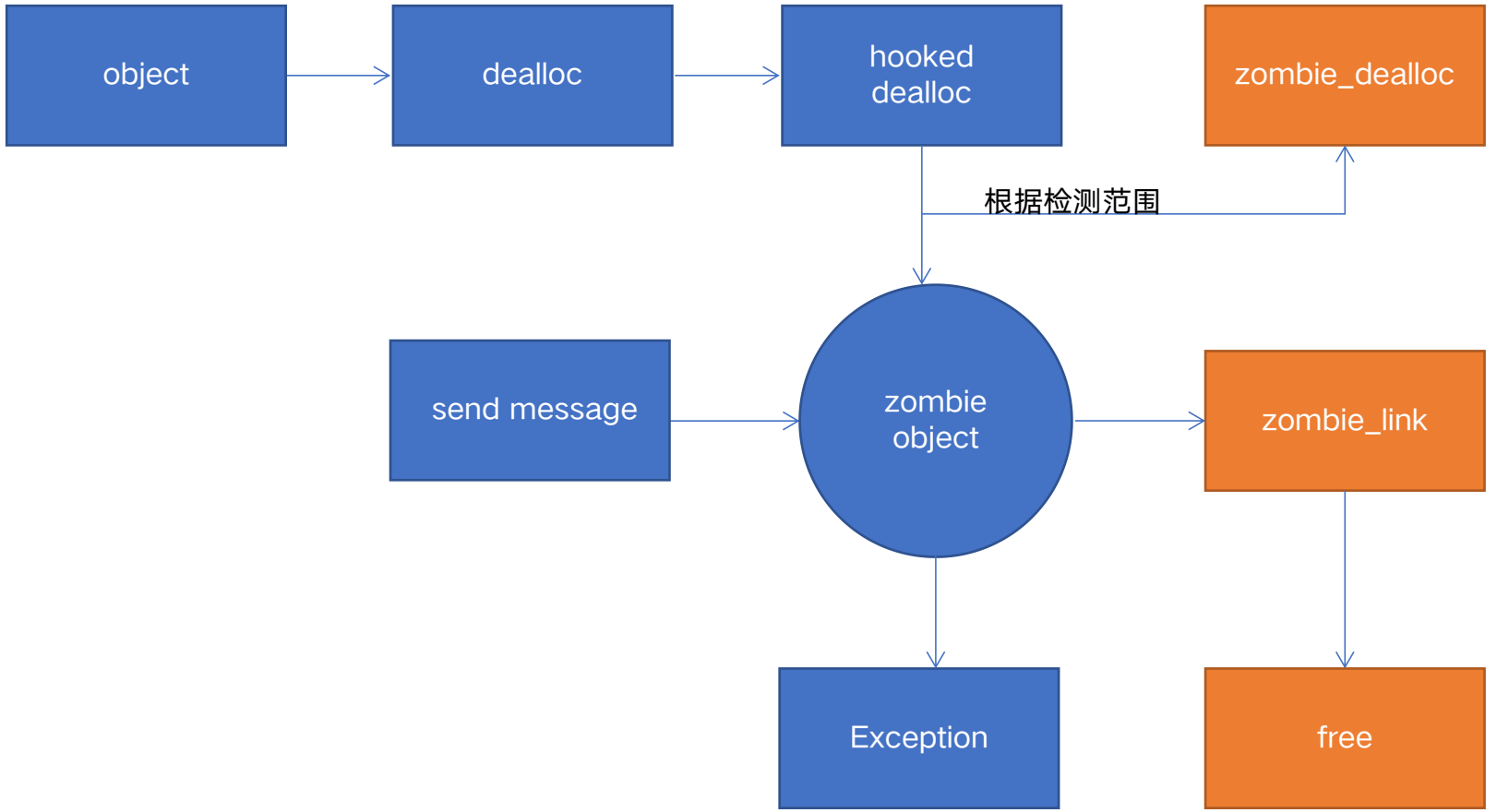


归因优势
1.可以直接定位到发生问题的类
2.可以提高偶现问题的复现概率

适用范围
OC 对象野指针导致的内存问题

缺陷
只能连接xcode

疑难问题归因-野指针



优势
1.完全继承xcode-zombie的特性
2.经过线上验证
3.可定义检测范围和僵尸存活时间
4.捕获第一案发现场后自定义上报

缺点
会增加一点运行内存

疑难问题归因-野指针

```
Application Specific Information:
*** Terminating app due to uncaught exception 'NSInternalInconsistencyException', reason: 'zombieCheck:(-[__NSArrayM retain]) was sent to a zombie object at address: 0x6000021eea90

zombieStack:[
    0   HLLDriver_Example                 0x00000001086028e6 -[HDZombieProxy sentExceptionWithSelector:] + 60,
    1   HLLDriver_Example                 0x0000000108602820 -[HDZombieProxy retain] + 20,
    2   HLLDriver_Example                 0x0000000107365596 __72-[HLLLoginManager handleLoginCompleteWithPhone:data:autoLogin:complete:]_block_invoke.909 + 150,
    3   HLLDriver_Example                 0x00000001073660ad __38-[HLLLoginManager requestLoginConfig:]_block_invoke + 77,
    4   libdispatch.dylib                 0x00007ff80013b7fb _dispatch_call_block_and_release + 12,
    5   libdispatch.dylib                 0x00007ff80013ca3a _dispatch_client_callout + 8,
    6   libdispatch.dylib                 0x00007ff80014c32c _dispatch_main_queue_drain + 1338,
    7   libdispatch.dylib                 0x00007ff80014bde4 _dispatch_main_queue_callback_4CF + 31,
    8   CoreFoundation                    0x00007ff800387b1f __CFRUNLOOP_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__ + 9,
    9   CoreFoundation                    0x00007ff800382436 __CFRunLoopRun + 2482,
    10  CoreFoundation                    0x00007ff8003816a7 CFRunLoopRunSpecific + 560,
    11  GraphicsServices                  0x00007ff809cb128a GSEventRunModal + 139,
    12  UIKitCore                         0x000000011ef46ad3 -[UIApplication _run] + 994,
    13  UIKitCore                         0x000000011ef4b9ef UIApplicationMain + 123,
    14  HLLDriver_Example                 0x0000000104c585e3 main + 131,
    15  dyld                              0x000000011014e2bf start_sim + 10,
    16  ???                               0x0000000111c5c52e 0x0 + 4593141038
]
```

问题分析

1.频繁登录，退出登录的场景，可能会出现
2.场景比较单一，算是多线程安全问题导致

疑难问题归因-Watchdog

触发Watchdog的条件

生命周期	超时时间
启动 Launch	20 s
恢复 Resume	10 s
悬挂 Suspend	10 s
退出 Quit	6 s
后台 Background	10 min

归因难点
1.常见于低端机，线下复线难
2.卡死非单一问题
3.疑似死锁问题的分析门槛高

疑难问题归因-Watchdog

这个是咱们卡顿线上检测工具，捕获到的堆栈。

```
Thread 0:
0       libsystem_kernel.dylib          _mach_msg_trap
1       libsystem_kernel.dylib          _mach_msg
2       libdispatch.dylib               __dispatch_mach_send_and_wait_for_reply
3       libdispatch.dylib               _dispatch_mach_send_with_result_and_wait_for_reply$VARIANT$armv81
4       libxpc.dylib                    _xpc_connection_send_message_with_reply_sync
5       Security                        _securityd_message_with_reply_sync
6       Security                        _securityd_send_sync_and_do
7       Security                        ___SecItemDelete_block_invoke_2
8       Security                        ___SecItemAuthDoQuery_block_invoke
9       Security                        _SecItemAuthDo
10      Security                        _SecItemAuthDoQuery
11      Security                        _SecOSStatusWith
12      Security                        _SecItemDelete
13      HLLDriver_Example               +[HLLRSAUtil addPublicKey:] HLLRSAUtil.m:261
14      HLLDriver_Example               +[HLLRSAUtil encryptData:publicKey:]    HLLRSAUtil.m:221
15      HLLDriver_Example               +[HLLRSAUtil encrypt:PublicKey:]    HLLRSAUtil.m:34
16      HLLDriver_Example               -[HLLIMNetworkManager encrypt:] HLLIMNetworkManager.m:1086
17      HLLDriver_Example               -[HLLIMNetworkManager getBatchQueryAccountInfoImIdList:completionBlock:]    HLLIMNetworkManager.m:318
18      HLLDriver_Example               __67-[HLLIMService loadConversationAccountInfoNextSeq:count:succ:fail:]_block_invoke    HLLIMService.m:1008
19      libdispatch.dylib               __dispatch_call_block_and_release
20      libdispatch.dylib               __dispatch_client_callout
21      libdispatch.dylib               __dispatch_main_queue_callback_4CF$VARIANT$armv81
22      CoreFoundation                  ___CFRUNLOOP_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__
23      CoreFoundation                  ___CFRunLoopRun
24      CoreFoundation                  _CFRunLoopRunSpecific
25      GraphicsServices                _GSEventRunModal
26      UIKitCore                       -[UIApplication _run]
27      UIKitCore                       _UIApplicationMain
28      HLLDriver_Example               main    main.m:39
29      libdyld.dylib                   _start
```


• 疑难问题归因-Watchdog

```
#pragma mark - 根据imId批量查询账号信息
- (void)getBatchQueryAccountInfoImIdList:(NSArray *)imIdList
    completionBlock:(CompletionBlock)completionBlock {
    NSMutableArray *rasIMIDArray = @[].mutableCopy; //使用RSA加密
    if ([[HLLIMService sharedInstance] imVersion] == HLLIMVersionTpyeAccount) {

        [rasIMIDArray addObjectsFromArray:imIdList];
    }else {
        for (NSString *imid in imIdList) {
            [rasIMIDArray addObject:[self encrypt:imid]?:@""];
        }
    }
```

问题分析

1.经过调试后，发现接口请求是在主线程
2.ras对称加密非常耗时

解决办法

由于业务特征决定，没有办法优化任务复杂度。所以将耗时操作放在子线程执行。

疑难问题归因-Watchdog

死锁案例

- _psynch_mutexwait
- _ulock_wait
- _psynch_cvwait
- _semwait_signal

#35 Thread

0	libsystem_kernel.dylib	__psynch_cvwait + 8
1	libsystem_pthread.dylib	0x0000000194d96000 + 15608
2	Glog	glog::ConditionVariable::await(glog::ThreadLock&) + 40
3	Glog	glog::Glog::getArchiveSnapshot(std::__1::vector<std::__1::basic_string<ch char> >, std::__1::allocator<std::__1::basic_string<char, std::__1::char_ g::ArchiveCondition const&, glog::FileOrder) + 540
4	Glog	+[GlogWrapper getArchiveSnapshot:] + 184
5	Glog	+[GlogWrapper glogUpload:andRemoveHandel:] + 416
6	HLLDriver_Example	-[ArgusReadManager readUplad] (ArgusReadManager.m:0)
7	HLLDriver_Example	+[ArgusRequest uploadLogs] (ArgusRequest.m:0)
8	HLLPollManager	__23-[HLLPollManager fire:]_block_invoke_2 + 456
9	libdispatch.dylib	__dispatch_call_block_and_release + 24
15	libsystem_pthread.dylib	_pthread_wqthread + 464

特征：
1. 崩溃调用栈随机性很高，参考意义不大
2. 查看其它线程，多数线程处于标签所示状态
3. 有部分任务，在不同线程多次执行且重复出现

则可以初步判定为疑似死锁

疑难问题归因-Watchdog

导出本页Crash列表 | 打包本页extraMessage附件

上报 ID	版本	异常上报时间	设备	系统版本
# 9001	6.1.56.3	2022-04-27 11:18:20 937	iPhone 6 Plus	12.5.2 (16H30)
# 8999	6.1.52.5	2022-04-18 18:45:26 145	iPhone 6	12.5.5 (16H62)
# 9069	6.1.52.5	2022-04-16 16:33:41 238	iPhone 6 Plus	12.4.8 (16G201)
# 9002	6.1.52.5	2022-04-16 15:14:30 144	iPhone 6	12.5.5 (16H62)
# 9060	v6.1.97_2	2022-04-14 11:59:54 890	iPhone 6s Plus	13.6 (17G68)
# 9003	6.1.52.5	2022-04-06 10:01:20 487	iPhone 6	12.4.1 (16G102)
# 8999	6.1.52.5	2022-04-04 13:37:09 474	iPh iPhone 6	12.5.4 (16H50)
# 9003	v6.2.12_3	2022-04-03 16:49:31 274	iPhone 7 Plus	15.3.1 (19D52)
# 8999	v6.1.97_2	2022-04-02 22:02:58 782	iPhone 6	12.5.4 (16H50)
# 9003	6.1.52.5	2022-04-02 10:29:42 364	iPhone 6	12.4.9 (16H5)

问题原因

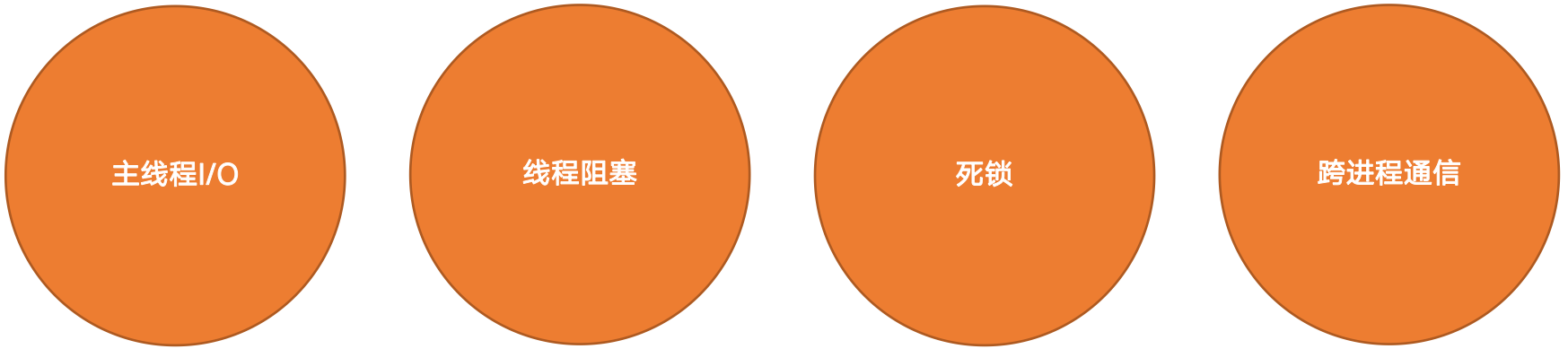
1.定时器每隔1分钟开辟一条线程，上报堆积的数据
2.由于glog上报任务的底层积压问题，导致子线程数据迟迟无法完成上报，线程也就无法释放，从而导致线程堆积，大量的线程等待或者阻塞

解决方案

1.开辟任务专用线程，防止创建大量线程
2.由于网络等原因无法完成上报时，底层进行数据离线缓存，待网络恢复后，一并上报

疑难问题归因-Watchdog

触发Watchdog的场景



疑难问题归因-OOM

触发OOM的条件

App使用内存，超过系统可提供的内存阀值

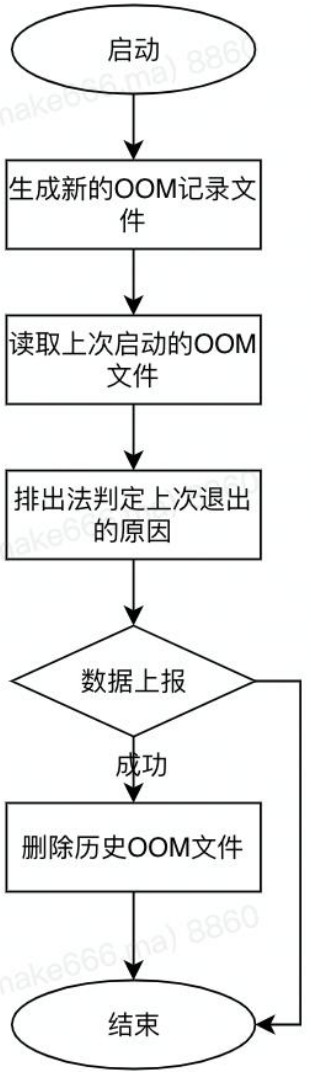
归因难点

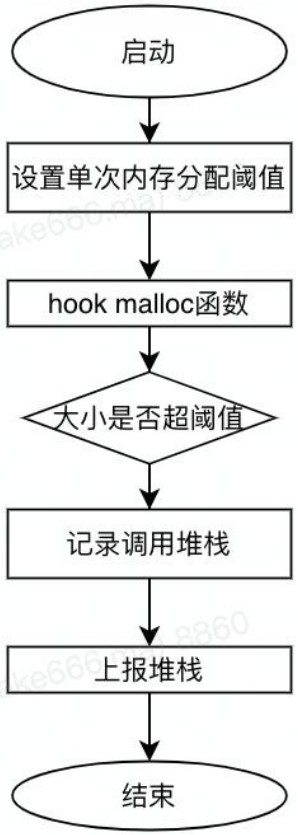
1.没有明确的调用栈信息
2.线下工具不适用于线上场景
3.模拟和复现问题非常困难

排除法收集OOM

大内存分配监控

线上内存泄漏检测





疑难问题归因-OOM

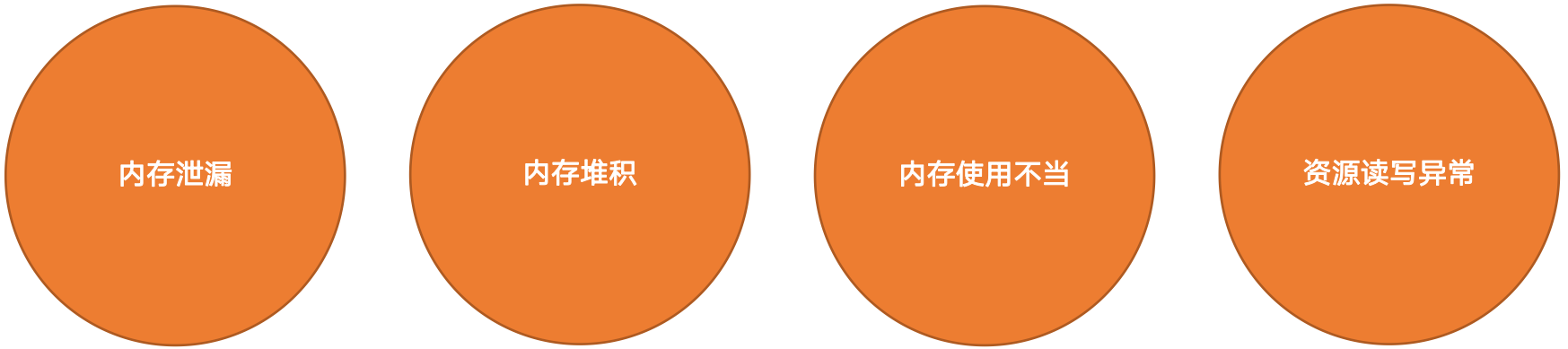
发生OOM最多的视图容器：

	A	B
1	视图容器	次数
2	HLLOrderDetailMainVC	33
3	HLLOrderHallMainVC	28
4	CAMImagePickerCameraViewController	25
5	HLLOrderDetailContainerVC	17
6	HLLPickupMainVC	10
7	HLLCameraStickerVC	9
8	HLLHeatMapVC	5
9	HLLHallOrderMapVC	4

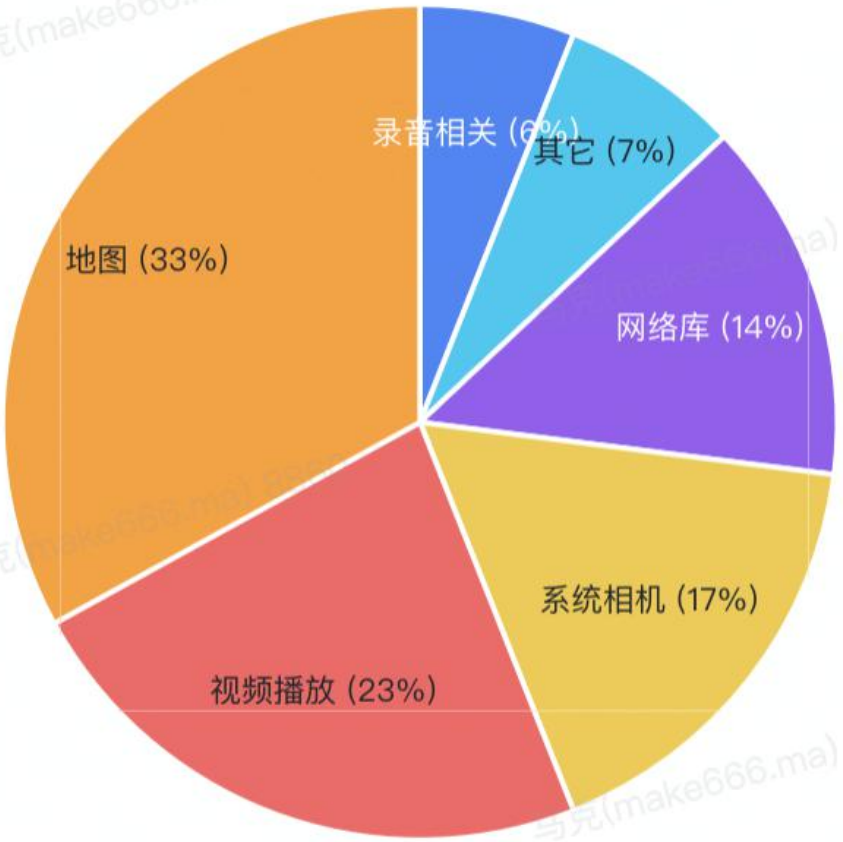
	A	B
1	mallocSize(MB)	[A] mallocReport的总次数(次)
2	10~20	518013
3	20~30	121530
4	30～40	27765
5	40~50	162016
6	50~60	11
7	60~70	15
8	70~80	224
9	80～90	7
10	90~100	8626
11	100~110	1
12	120~130	52
13	130~140	1
14	180~190	3

疑难问题归因-OOM

触发OOM的原因



• 疑难问题归因-疑难Crash



归因难点
1.纯系统调用栈
2.模拟和复现难度高
3.系统API或者版本导致

疑难问题归因-疑难Crash

```
// 崩溃线程
Thread 37 Crashed:
0       libsystem_platform.dylib        __os_unfair_lock_recursive_abort
1       libsystem_platform.dylib        __os_unfair_lock_lock_slow
2       CoreFoundation                  _CFSocketInvalidate
3       CFNetwork                       __CFNetworkErrorGetLocalizedDescription
4       CoreFoundation                  _CFArrayApplyFunction
5       CFNetwork                       __CFNetworkErrorGetLocalizedDescription
6       CoreFoundation                  __CFRelease
7       CoreFoundation                  _CFSocketInvalidate
8       CFNetwork                       __CFNetworkErrorGetLocalizedDescription
9       CoreFoundation                  __CFStreamClose
10      HLLPushService                  -[GtAsyncSocket closeWithError:]
11      HLLPushService                  ___27-[GtAsyncSocket disconnect]_block_invoke
12      libdispatch.dylib               __dispatch_client_callout
13      libdispatch.dylib               __dispatch_lane_barrier_sync_invoke_and_complete
14      HLLPushService                  -[GtAsyncSocket disconnect]
15      HLLPushService                  ___31-[GXDatagramChannel disconnect]_block_invoke
16      libdispatch.dylib               __dispatch_call_block_and_release
17      libdispatch.dylib               __dispatch_client_callout
18      libdispatch.dylib               __dispatch_lane_serial_drain$VARIANT$armv81
19      libdispatch.dylib               __dispatch_lane_invoke$VARIANT$armv81
20      libdispatch.dylib               __dispatch_workloop_worker_thread
21      libsystem_pthread.dylib         __pthread_wqthread
```

• 疑难问题归因-疑难Crash

```
// 崩溃线程
Thread 0 Crashed:
0       libobjc.A.dylib                 AutoreleasePoolPage::AutoreleasePoolPage(AutoreleasePoolPage*)
1       libobjc.A.dylib                 AutoreleasePoolPage::autoreleaseFullPage(objc_object*, AutoreleasePoolPage*)
2       libobjc.A.dylib                 AutoreleasePoolPage::autoreleaseFullPage(objc_object*, AutoreleasePoolPage*)
3       libobjc.A.dylib                 objc_object::rootAutorelease2()
4       CoreFoundation                  _CFArrayGetFirstIndexOfValue
5       CoreFoundation                  __CFStreamClose
6       HLLMQTTClient                   -[MQTTCFSocketDecoder close]    MQTTCFSocketDecoder.m:38
7       HLLMQTTClient                   -[MQTTCFSocketTransport internalClose]  MQTTCFSocketTransport.m:148
8       HLLMQTTClient                   -[MQTTSession closeInternal]    MQTTSession.m:543
9       HLLMQTTClient                   -[MQTTSession error:error:] MQTTSession.m:1274
10      HLLMQTTClient                   -[MQTTSession decoder:didReceiveMessage:]   MQTTSession.m:881
11      HLLMQTTClient                   -[MQTTDecoder stream:handleEvent:]  MQTTDecoder.m:171
12      CoreFoundation                  __signalEventSync
13      CoreFoundation                  ____signalEventQueue_block_invoke
14      libdispatch.dylib               __dispatch_call_block_and_release
15      libdispatch.dylib               __dispatch_client_callout
16      libdispatch.dylib               __dispatch_main_queue_callback_4CF$VARIANT$armv81
17      CoreFoundation                  ___CFRUNLOOP_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__
18      CoreFoundation                  ___CFRunLoopRun
19      CoreFoundation                  _CFRunLoopRunSpecific
20      GraphicsServices                _GSEventRunModal
21      UIKitCore                       -[UIApplication _run]
22      UIKitCore                       _UIApplicationMain
23      HLLDriver_Example               main    main.m:39
24      (null)                          0x0 + 4435640912
```


疑难问题归因-疑难Crash

问题分析

1.查看了用户操作日志，发现问题集中在前后台切换的过程中；
2.因为项目有MQTT和个推，都使用了CocoaAsyncSocket，就定位到了CocoaAsyncSocket这个库
3.然后在Github上查看了对应的issues, 看到之前大佬的留言，发现kCFStreamNetworkServiceTypeVoIP已经过期了，调用CocoaAsyncSocket中的对外enableBackgroundingOnSocket就会崩溃

```
* - (void)socket:(GCDAsyncSocket *)sock didConnectToHost:(NSString *)host port:(uint16_t)port
* {
*     [asyncSocket performBlock:^{
*         [asyncSocket enableBackgroundingOnSocket];
*     }];
* }
*/
- (BOOL)enableBackgroundingOnSocket;
```

问题修复

1.MQTT当时正在灰度，打算在它上边尝试，删掉了enableBackgroundingOnSocket方法的调用
2.上线后，发现相关类型的Crash确实在减少
3.方案同步给个推开发人员，个推更新版本。

疑难问题归因-疑难Crash

***** reason: EXC_BREAKPOINT (SIGTRAP)**

```
Thread 37 Crashed:
0       libsystem_platform.dylib        __os_unfair_lock_recursive_abort
1       libsystem_platform.dylib        __os_unfair_lock_lock_slow
2       CoreFoundation                  _CFSocketInvalidate
3       CFNetwork                       __CFNetworkErrorGetLocalizedDescription
4       CoreFoundation                  _CFArrayApplyFunction…
```

HLLPushService ＋添加标签

累计影响设备数	累计发生次数
361台	447次

解决方案: 开发中

上报趋势 | 版本信息 | 设备信息 | 地域分布 | 前后台统计 | 崩溃进程统计 | 崩溃页面统

应用版本	首次上报时间
6.2.84(4)	2022-10-28 18:03:16
6.2.76(6)	2022-09-30 08:57:39
6.2.80(2)	2022-10-12 17:21:17
6.2.82(2)	2022-10-21 23:27:14
6.2.74(1)	2022-09-30 12:11:23
6.2.73(1)	2022-10-11 15:42:40
6.2.64(4)	2022-10-24 17:44:15

止步于84

***** reason: EXC_BAD_ACCESS (SIGSEGV)**

```
Thread 0 Crashed:
0       libobjc.A.dylib                 AutoreleasePoolPage::AutoreleasePoolPage(AutoreleaseF
1       libobjc.A.dylib                 AutoreleasePoolPage::autoreleaseFullPage(objc_object*
2       libobjc.A.dylib                 AutoreleasePoolPage::autoreleaseFullPage(objc_object*
3       libobjc.A.dylib                 objc_object::rootAutorelease2()
4       CoreFoundation                  _CFArrayGetFirstIndexOfValue…
```

MQTT ＋添加标签

累计影响设备数	累计发生次数
456台	503次

解决方案: 开发中

上报趋势 | 版本信息 | 设备信息 | 地域分布 | 前后台统计 | 崩溃进程统计 | 崩溃页面统计

应用版本	首次上报时间
6.2.76(6)	2022-09-30 03:07:08
6.2.80(2)	2022-10-17 01:09:20
6.2.74(1)	2022-09-30 00:24:35
6.2.64(4)	2022-10-09 04:40:20
6.2.73(1)	2022-09-30 17:41:53
6.2.82(2)	2022-10-29 01:19:33
6.2.61(3)	2022-10-27 02:01:53

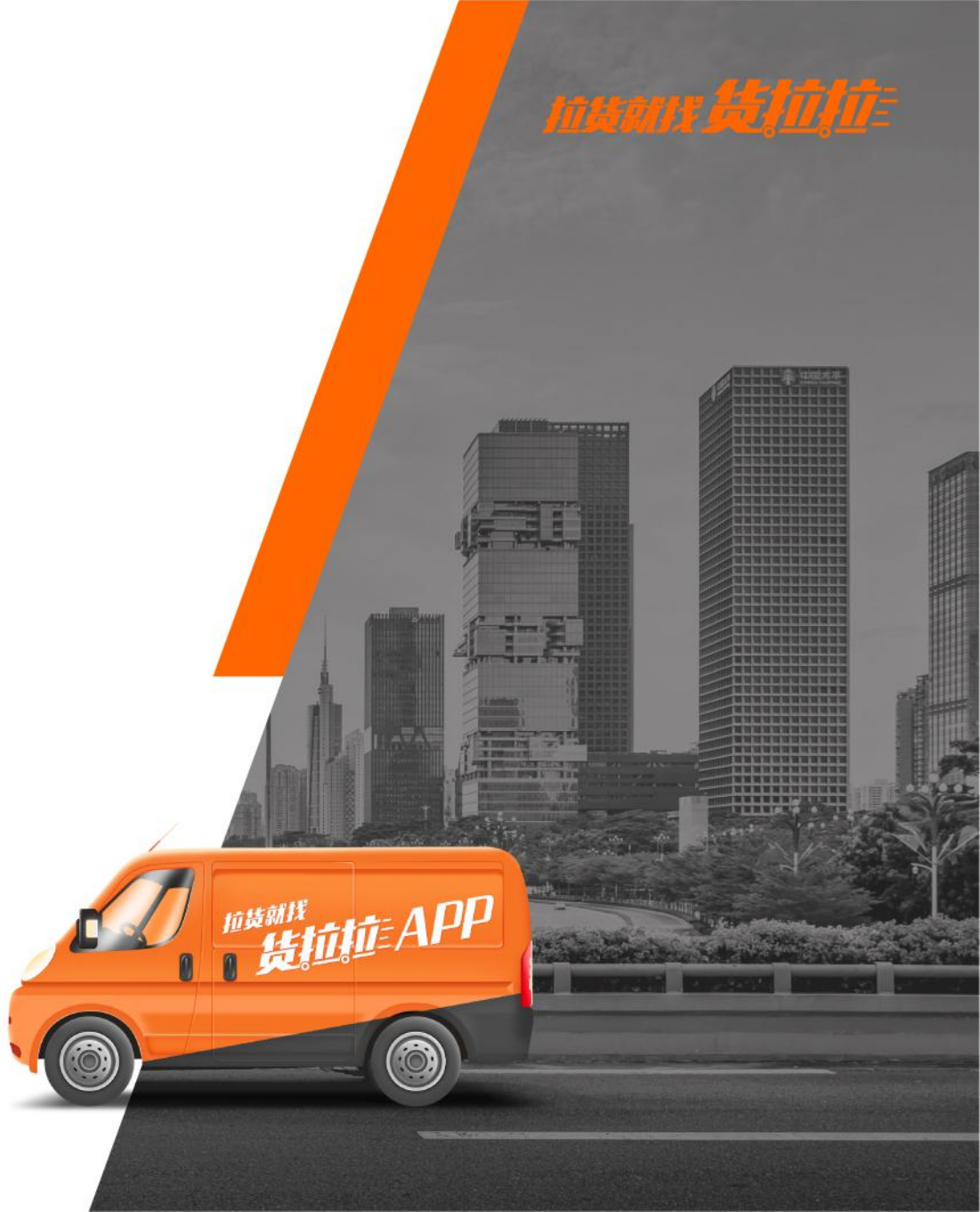
止步于80版本。
82特殊，不算它

疑难问题归因-小结

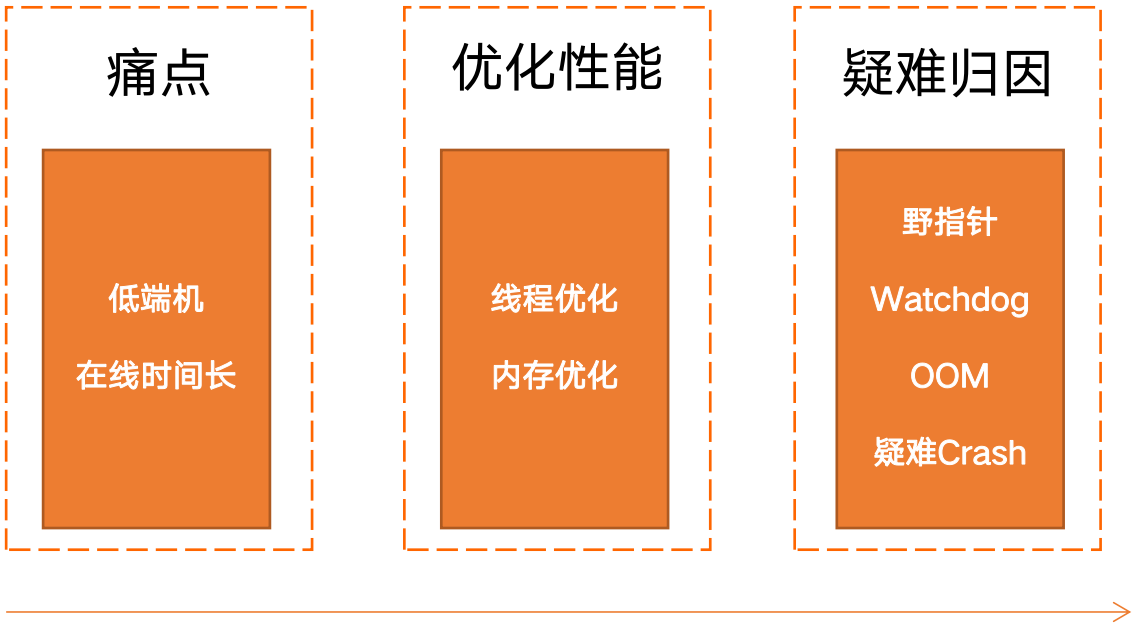
- zombie的原理和case分析
- 卡顿的case分析
- 死锁的排查思路
- Watchdog的工具的设计原理
- 疑难Crash的跟踪方法

收益：
1.目前司机端业务造成的设备崩溃率大概降到了万6-8
2.野指针检测工具，OOM检测工具，卡顿检测工具等其它性能工具都已经上线了

总结回顾



总结回顾


痛点
低端机
在线时间长
优化性能
线程优化
内存优化
疑难归因
野指针
Watchdog
OOM
疑难Crash


异常分析
异常概览
异常列表
卡顿监控
FPS监控
内存监控
启动监控
CPU监控
线程监控