HXPerformanceCenter

性能平台的搭建

性能平台

- iOS 客户端

- 展示平台

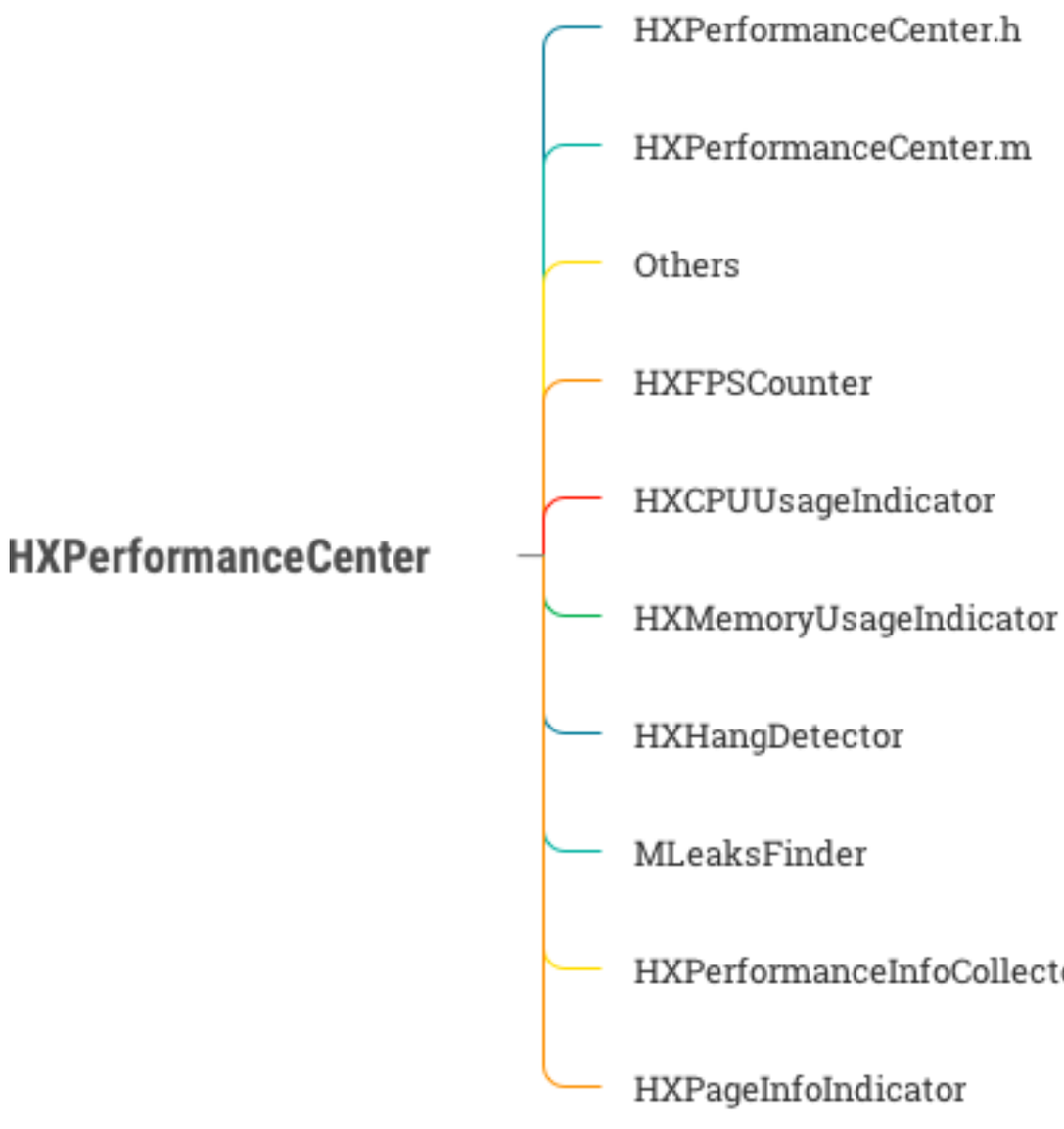
iOS 客户端


FPS 54 优
CPU 35.3%
Memory 49.3 MB
VIP
IMP
自选股
大盘指数
涨跌排名
热门板块
委托交易
我的资产
决策
4.6%
基金理财
猜涨跌，赢超值好礼
投资参考
更多
严查次新股暴炒 六大监管动作影响下周A股
【除了“闪崩股” 这五大因素将搅动下周市场】
【焦点】全世界屏住呼吸!朝鲜:今天极度危险
【花妹】让郭嘉队先走！小散又当接盘侠啦?
【特色】草根股民致刘士余：监管让制度说话
神奇电波的秘密？ 60秒开户私享
首页
行情
自选
交易
资讯

iOS 客户端

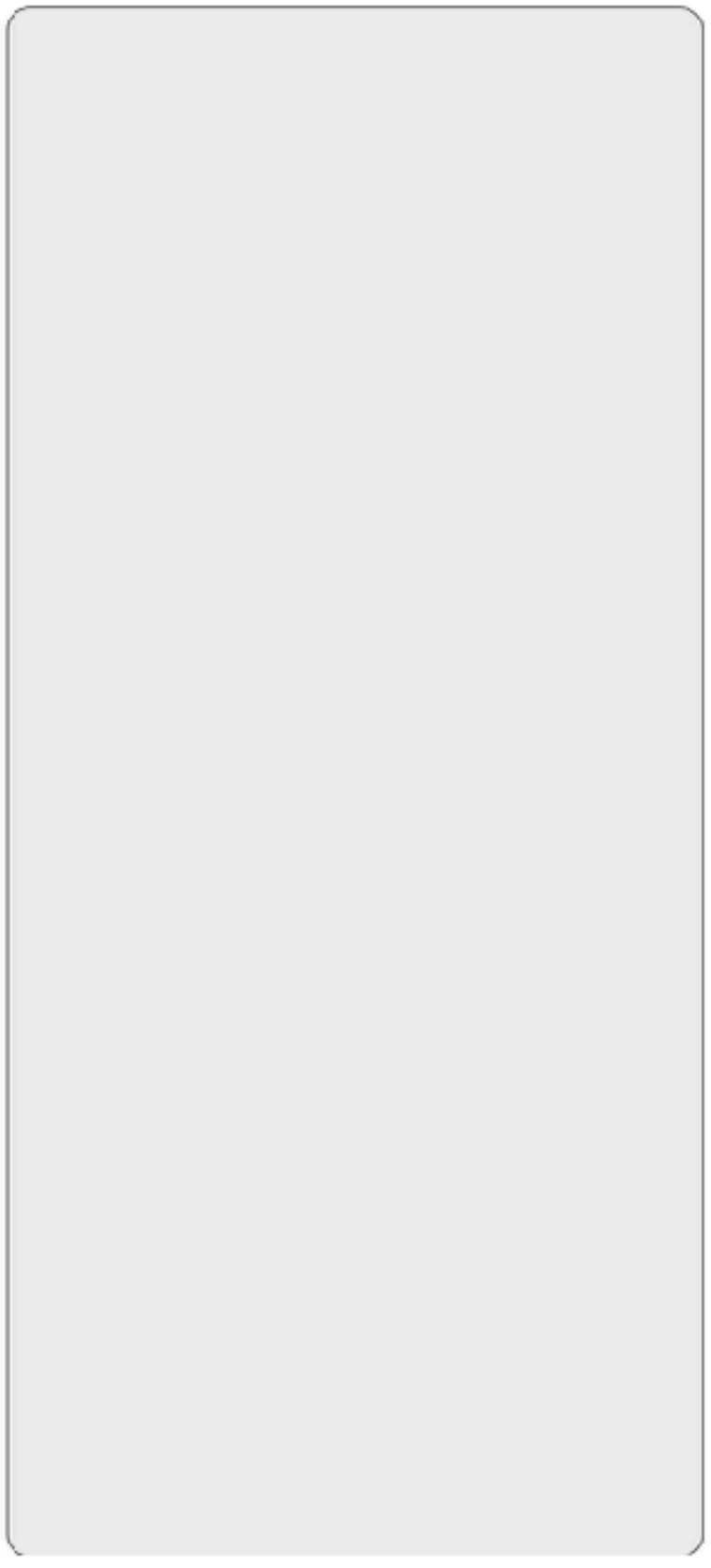
- 性能平台的框架
- 性能参数介绍
- 一些问题与优化

框架



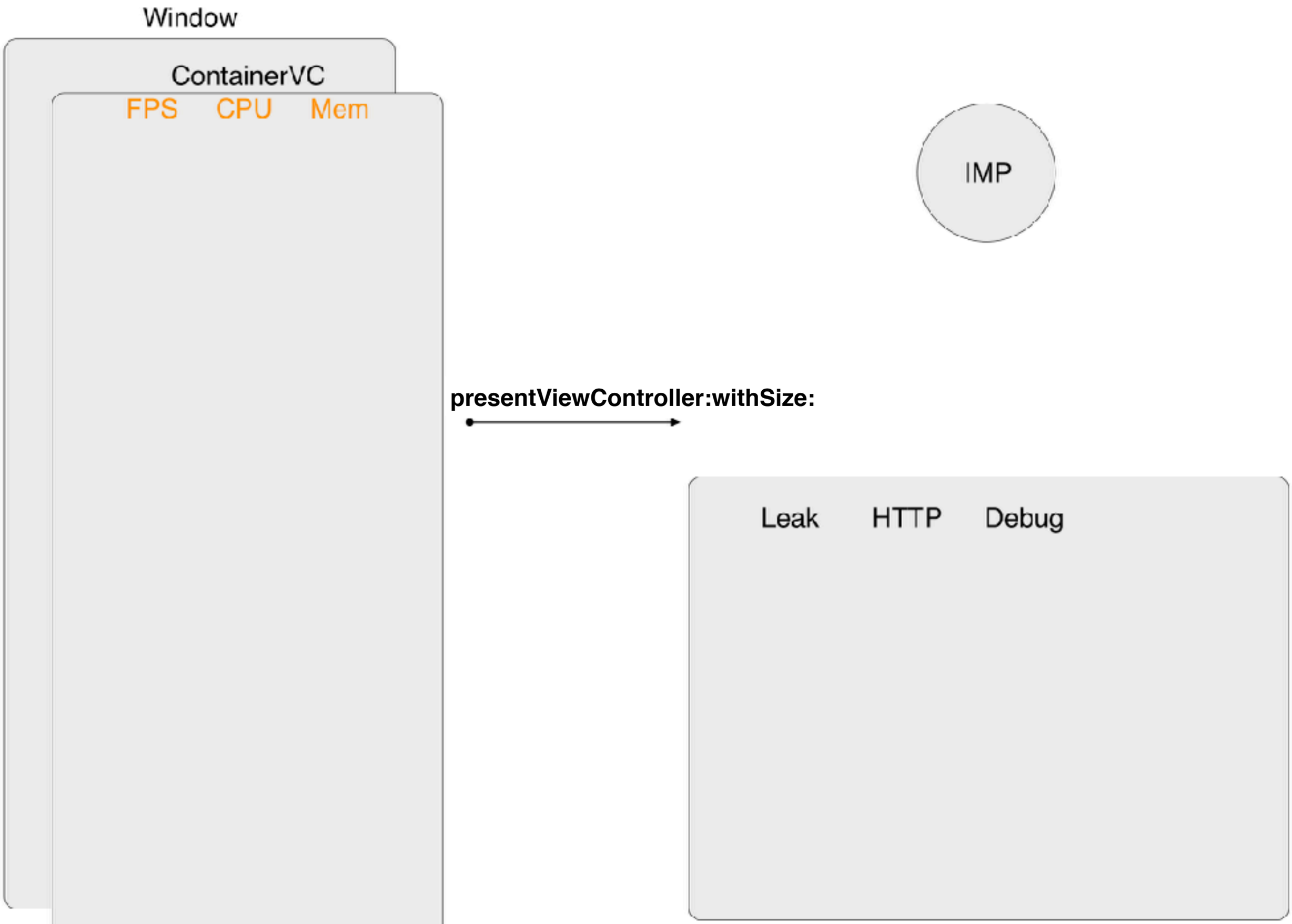
- 入口
- HXPerformanceCenter 相关
- FPS
- CPU
- Memory
- 卡顿检测
- 内存泄漏检测
- 性能数据收集
- 页面数据上报

Window



windowLevel =
UIWindowLevelStatusBar + 100

- pointInside:withEvent:


Window
ContainerVC
FPS CPU Mem
IMP
presentViewController:withSize:
Leak HTTP Debug

Window

ContainerVC

FPS CPU Mem

Condensed

IMP

FloatButtonVC

presentViewController:withSize:

FullWindow

Leak HTTP Debug

VC

FPS

- Frames Per Second

FPS

- Frames Per Second

~~First-person shooting game~~

FPS

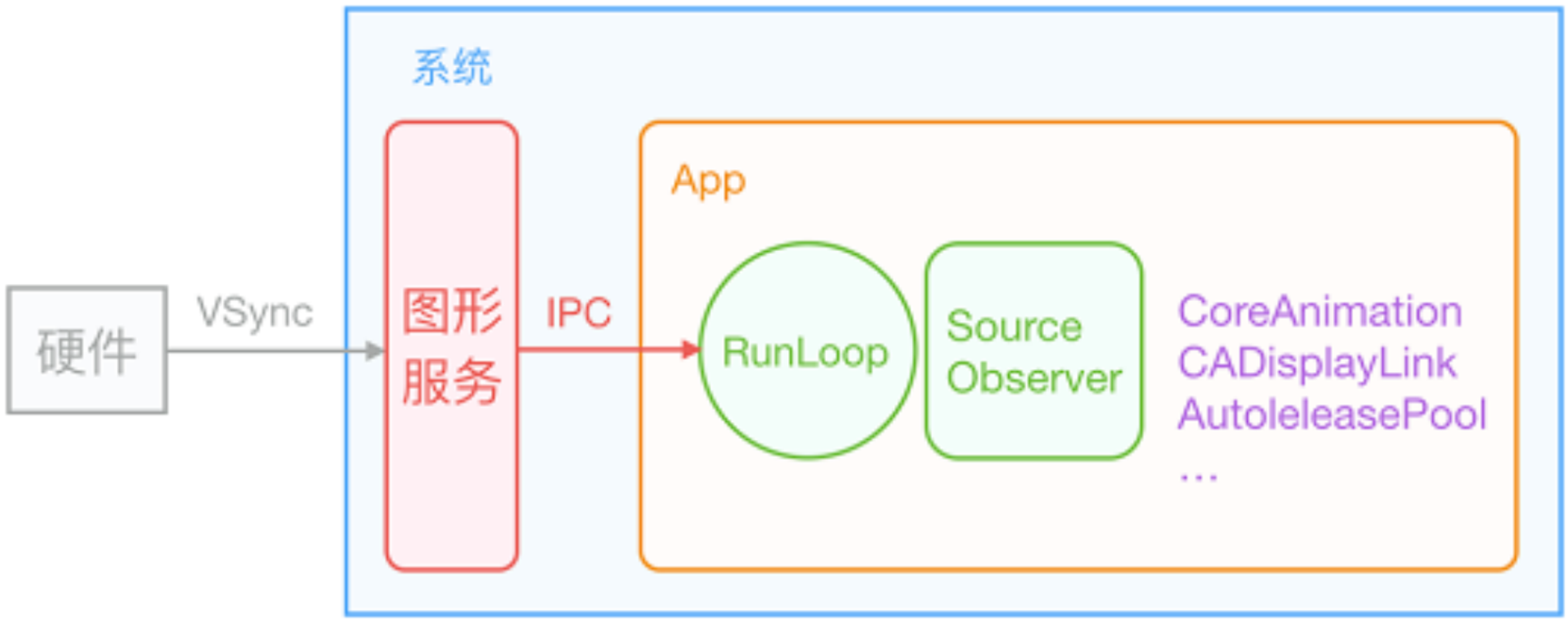
- Frames Per Second
- <30 差
- 60

FPS

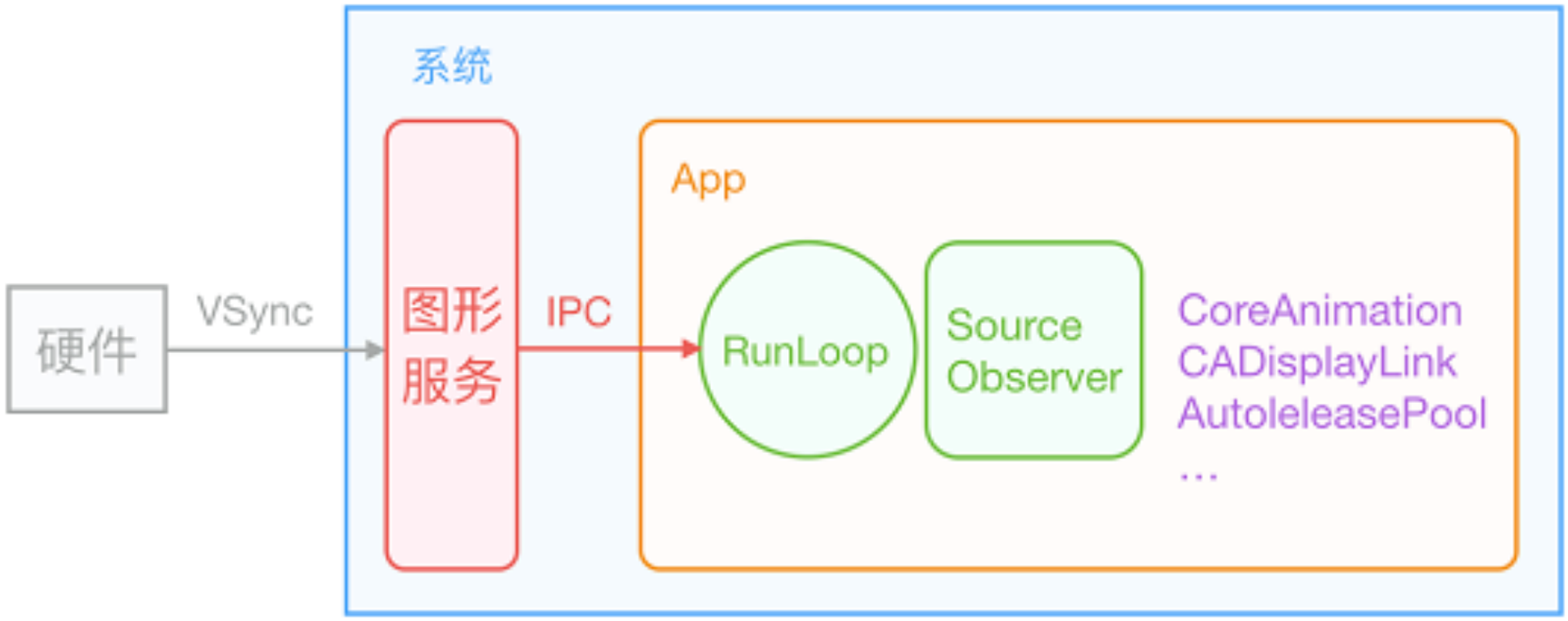
- Frames Per Second
- <30 差
- 60
- CADisplayLink

FPS

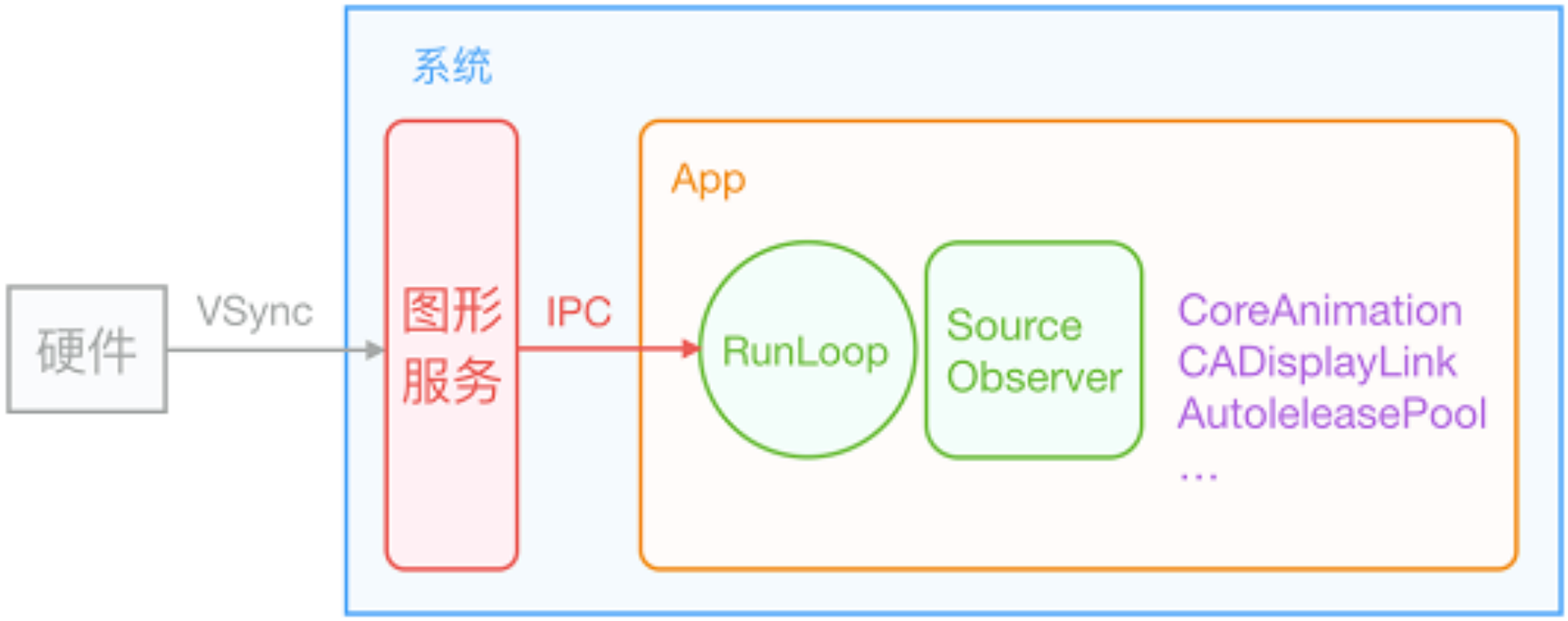
- Frames Per Second
- <30 差
- 60
- CADisplayLink



FPS


系统
硬件
VSync
图形
服务
IPC
App
RunLoop
Source
Observer
CoreAnimation
CADisplayLink
AutoleleasePool
...

FPS



- HSync vs VSync
- IPC (Inter-Process Communication)

CPU

CPU

```
#import <mach/mach.h>

// 获取线程列表与数量
task_threads(&threadList, &threadCount);

float totalUsage = 0.0;

// 遍历所有线程
for (int i = 0; i < threadCount; i++) {
   // 获取线程的基本信息
   thread_info(threadList[i], threadInfo, &threadInfoCount);

   // 线程非闲置状态时，进行累加
   if (threadInfo -> flag) {
        totalUsage += threadInfo -> cpu_usage
   }

}
```

Memory

Memory

```
#import <mach/mach.h>

struct task_basic_info info;
mach_msg_type_number_t size = sizeof(info);

kern_return_t kerr = task_info(mach_task_self(),
TASK_BASIC_INFO, (task_info_t)&info, &size);

if( kerr == KERN_SUCCESS ) {
   return info.resident_size;
}
```

clean memory

clean memory are memories that can be recreated, on iOS it is memory of:

- system framework / binary executable of your app / memory mapped files

Also notice this situation: when your app link to a framework, the clean memory will increase by the size of the framework binary. But most of time, only part of binary is really loaded in physical memory.

dirty memory

All memory that is not clean memory is dirty memory, dirty memory can't be recreated by system.

When there is a memory pressure, system will unload some clean memory, when the memory is needed again, system will recreate them.

But for dirty memory, system can't unload them, and iOS has no swap mechanism, so dirty memory will always be kept in physical memory, till it reach a certain limit, then your App will be terminated and all memory for it is recycled by system.

virtual memory

```
virtual memory = clean memory + dirty memory.
```

That means virtual memory is all the memory your App want.

resident memory

```
resident memory = dirty memory + clean memory that loaded in physical memory
```

resident memory is the memory really loaded in your physical memory, it mean all the dirty memory and parts of your clean memory.

Memory

Conclusion

At any time, this is always true:

```
virtual memory == (clean memory + dirty memory) > resident memory
> dirty memory
```

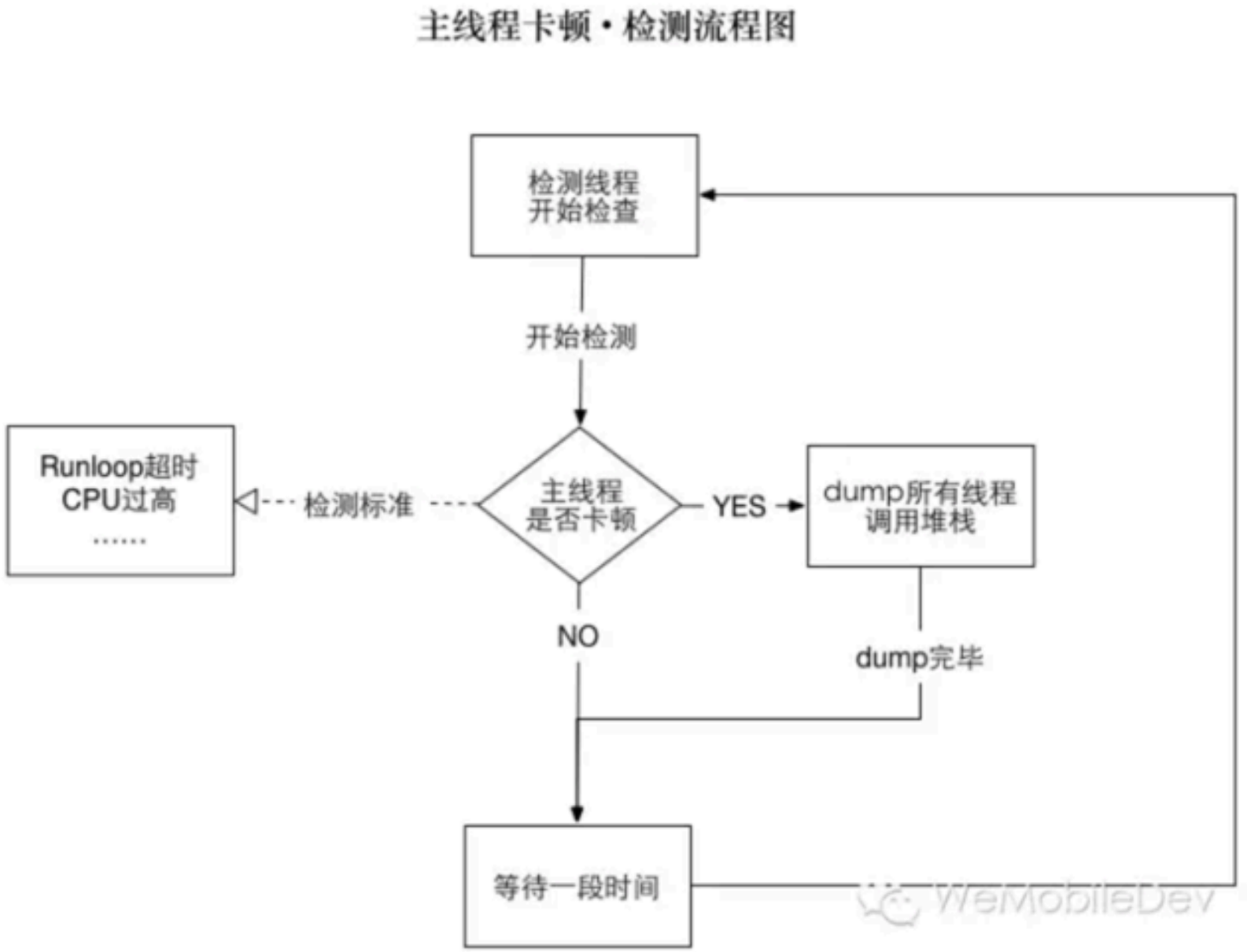
If you are worrying about the physical memory your App is taking(which is the key reason your App is terminated due to low memory), you should mainly focus on resident memory.

卡顿检测

卡顿原因

- 死锁：主线程拿到锁 A，需要获得锁 B，而同时某个子线程拿了锁 B，需要锁 A，这样相互等待就死锁了。
- 抢锁：主线程需要访问 DB，而此时某个子线程往 DB 插入大量数据。通常抢锁的体验是偶尔卡一阵子，过会就恢复了。
- 主线程大量 IO：主线程为了方便直接写入大量数据，会导致界面卡顿。
- 主线程大量计算：算法不合理，导致主线程某个函数占用大量 CPU。
- 大量的 UI 绘制：复杂的 UI、图文混排等，带来大量的 UI 绘制。

卡顿检测


主线程卡顿·检测流程图
检测线程
开始检查
开始检测
主线程
是否卡顿
检测标准
Runloop超时
CPU过高
.....
YES
dump所有线程
调用堆栈
dump完毕
NO
等待一段时间
WeMobileDev

卡顿检测

```
// 回调
callback(observer, activity, info) {
    // 信号量 +1
    dispatch_semaphore_signal()
}

// 创建信号量，以 0 为起始值
dispatch_semaphore_create(0)
// 上下文
CFRunloopObserverContext context = {0, self, NULL, NULL};
// 创建 runloop 的观察者
CFRunloopObserverCreate(&callback,&context);
// 添加对 runloop 的观察
CFRunloopAddObserver(CFRunloopGetMain(), observer, kCFRunloopCommonModes);
// 子线程监听
dispatch_async(dispatch_get_global_queue(0,0), ^{
    while(YES) {
        // 捕获信号量的值
        long value = dispatch_semaphore_wait(semaphore, interval);
        if (value != 0) {
            // PLCrashReport 堆栈
        }
    }
})
```

遇到的一些问题

- 解决与性能工具冲突的功能问题?
- 对于性能工具增加一个快捷键，能在界面上直接关闭性能工具，无需进入myhexin后台进行关闭操作
- 将测试过程中的性能检测数据传到服务器上，最好能形成图表，表格形式，方便排查问题
- 上传数据的机制太暴力，需要优化

问题与优化

- keyWindow

```
[UIApplication aspect_hookSelector:@selector(keyWindow)
withOptions:AspectPositionAfter usingBlock:^(id<AspectInfo>
aspectInfo, BOOL animated) {
    NSInvocation *invo = aspectInfo.originalInvocation;
    // 性能工具开启时
    if (_enable) {
       // 遍历 windows 数组找到 EQUIWindow
       UIWindow *kw = EQUIWindow;
       // 设置返回值
       [invo setReturnValue:&kw];
    }

} error:NULL];
```

问题与优化

- 横竖屏

性能工具开启时，关闭接收设备的转向通知

```
while ([UIDevice currentDevice].isGeneratingDeviceOrientationNotification) {
    [[UIDevice currentDevice] endGeneratingDeviceOrientationNotification];
}
```

性能工具关闭时，标记一下 shouldDealloc = YES

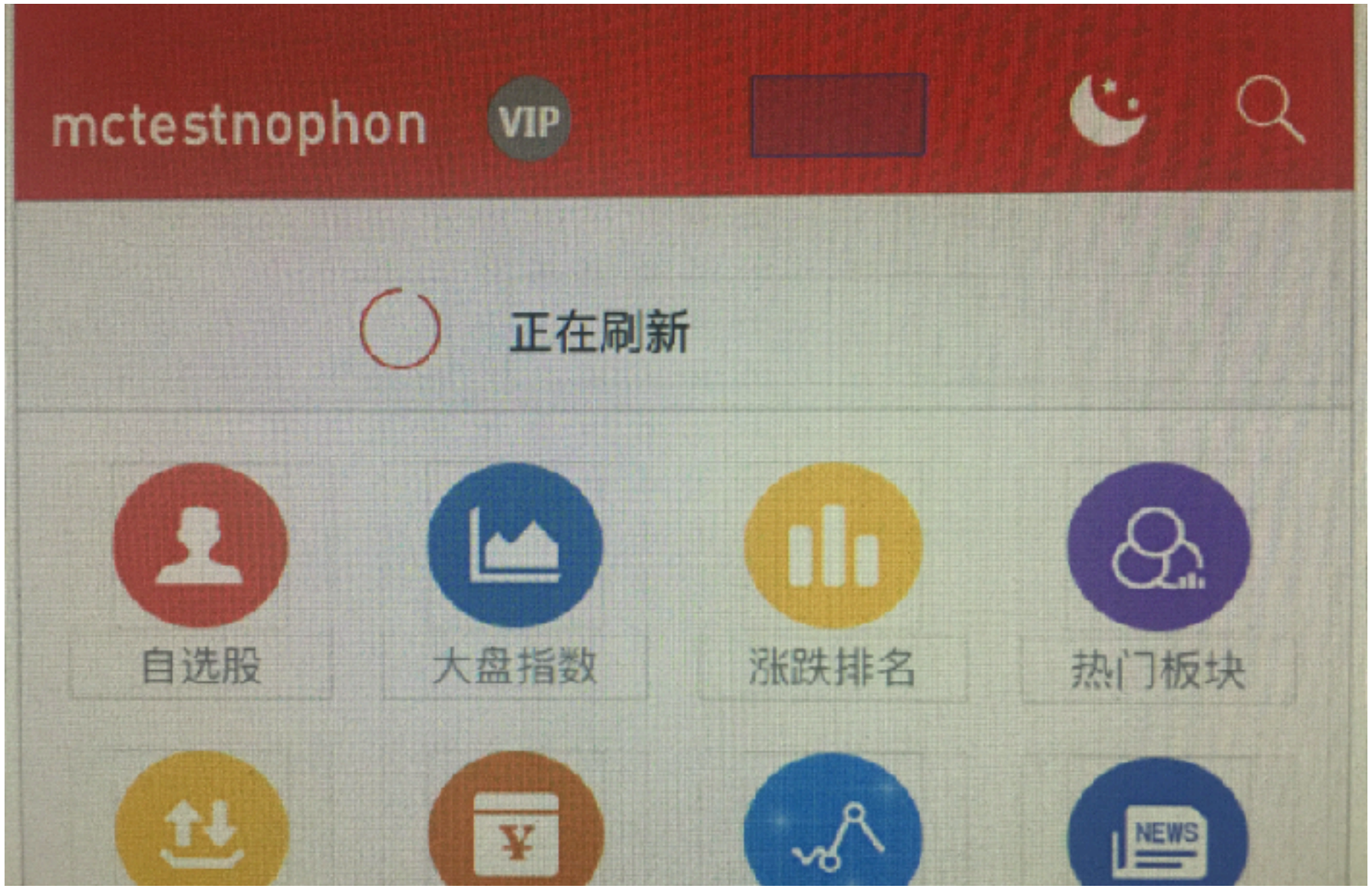
window:

监听 UIDeviceOrientationDidChangeNotification

```
// 判断是否需要接收设备的转向通知
if (shouldDealloc) {
    // 开始接收设备的转向通知
    while (![UIDevice currentDevice].isGeneratingDeviceOrientationNotification) {
        [[UIDevice currentDevice] beginGeneratingDeviceOrientationNotification];
    }
} else {
    // 不接收设备的转向通知
    while ([UIDevice currentDevice].isGeneratingDeviceOrientationNotification) {
        [[UIDevice currentDevice] endGeneratingDeviceOrientationNotification];
    }
}
```

问题与优化

- 快捷开关



问题与优化

- 性能数据上传 HXPerformanceInfoCollector
- pageID
- [AMUIPublicProxy currentPageControl].pageID;

问题与优化

- 上传机制优化

问题与优化

- 上传机制优化
- 原：应用退后台，写数据到文件，再进前台，三秒后会进行上传

问题与优化

- 上传机制优化
- 原：应用退后台，写数据到文件，再进前台，三秒后会进行上传
- 优化：
- 1、数据少于 15 条，则不上传，直接抛弃
- 2、数据达到一定长度（如，60 条）时直接上传，但仍保留原机制，服务器端以 sessionID 区分是否为同一次前台运行

性能平台

- 展示平台


iOS Pi 平台
iOS 安装包下载平台
admin
MAIN
Dashboard
TestAutomation

Overview
Linus iPhone
2017/4/14 上午11:31:20
Basic Info
Chart with FPS
FPS
60
-1
55.66
pageId: 2602
FPS : 36
60
50
40
30
20
10
0
-10
11:28:03
11:28:07
11:28:11
11:28:15
11:28:19
11:28:23
11:28:27
11:28:31
11:28:35
11:28:39
11:28:43
11:28:47
11:28:51
© 10JQKA.com.cn
Powered by: Linus

Q&A

参考文献

- http://blog.ibireme.com/2015/11/12/smooth_user_interfaces_for_ios/

- http://stackoverflow.com/questions/13437365/what-is-resident-and-dirty-memory-of-ios

- http://wereadteam.github.io/2016/05/03/WeRead-Performance/

- http://mp.weixin.qq.com/s?__biz=MzAwNDY1ODY2OQ==&mid=207890859&idx=1&sn=e98dd604cdb854e7a5808d2072c29162&scene=4#wechat_redirect