iOS研发助手DoraemonKit技术实现（二）

景铭巴巴 (/u/c3c893a27097) 十关注
3.5 2018.11.22 12:57* 字数 2548 阅读 2676 评论 2 喜欢 23
(/u/c3c893a27097)

一、前言

iOS研发助手DoraemonKit技术实现（一）(https://www.jianshu.com/p/00763123dbc4)中介绍了几个常用工具集的技术实现，大家如果有疑问的话，可以在文章中进行留言，也希望大家接入试用，或者加入到DoraemonKit交流群一起交流。

性能问题极大程度的会影响到用户的体验，对于我们开发者和测试同学要随时随地保证我们app的质量，避免不好的体验带来用户的流失。本篇文章我们来讲一下，性能监控的几款工具的技术实现。主要包括，帧率监控、CPU监控、内存监控、流量监控、卡顿监控和自定义监控这几个功能。

有人说帧率、CPU和内存这些信息我们都可以在Xcode中的Instruments工具进行联调的时候可以查看，为什么还要在客户端中打印出来呢?

1. 第一、很多测试同学比较关注App质量，但是他们却没有Xcode运行环境，他们对于质量数据无法很有效的查看。
2. 第二、App端实时的查看App的质量数据，不依赖IDE，方便快捷直观。
3. 第三、实时采集性能数据，为后期结合测试平台产生性能数据报表提供数据来源。

二、技术实现

3.1：帧率展示

帧率展示

app的流畅度是最直接影响用户体验的，如果我们app持续卡顿，会严重影响我们app的用户留存度。所以对于用户App是否流畅进行监控，能够让我们今早的发现我们app的性能问题。对于App流畅度最直观最简单的监控手段就是对我们App的帧率进行监控。

帧率（FPS）是指画面每秒传输帧数，通俗来讲就是指动画或视频的画面数。FPS是测量用于保存、显示动态视频的信息数量。每秒钟帧数愈多，所显示的动作就会越流畅。对于我们App开发来说，我们要保持FPS高于50以上，用户体验才会流畅。

在YYKit Demo工程中有一个工具类叫YYFPSLabel，它是基于CADisplayLink这个类做FPS计算的，CADisplayLink是CoreAnimation提供的另一个类似于NSTimer的类，它会在屏幕每次刷新回调一次。既然CADisplayLink可以以屏幕刷新的频率调用指定selector，而且iOS系统中正常的屏幕刷新率为60Hz（60次每秒），那只要在这个方法里面统计每秒这个方法执行的次数，通过次数/时间就可以得出当前屏幕的刷新率了。

大致实现思路如下：

```
- (void)startRecord{
    if (_link) {
        _link.paused = NO;
    }else{
        _link = [CADisplayLink displayLinkWithTarget:self selector:@selecto
r(trigger:)];
        [_link addToRunLoop:[NSRunLoop mainRunLoop] forMode:NSRunLoopCommon
Modes];
        _record = [DoraemonRecordModel instanceWithType:DoraemonRecordTypeF
PS];
        _record.startTime = [[NSDate date] timeIntervalSince1970];
    }
}

- (void)trigger:(CADisplayLink *)link{
    if (_lastTime == 0) {
        _lastTime = link.timestamp;
        return;
    }

    _count++;
    NSTimeInterval delta = link.timestamp - _lastTime;
    if (delta < 1) return;
    _lastTime = link.timestamp;
    CGFloat fps = _count / delta;
    _count = 0;

    NSInteger intFps = (NSInteger)(fps+0.5);
    // 0~60   对应 高度0~200
    [self.record addRecordValue:fps time:[[NSDate date] timeIntervalSince19
70]];
    [_oscillogramView addHeightValue:fps*200./60. andTipValue:[NSString str
ingWithFormat:@"%zi",intFps]];
}
```

值得注意的是基于CADisplayLink实现的 FPS 在生产场景中只有指导意义，不能代表真实的 FPS，因为基于CADisplayLink实现的 FPS 无法完全检测出当前 Core Animation 的性能情况，它只能检测出当前 RunLoop 的帧率。但要真正定位到准确的性能问题所在，最好还是通过Instrument来确认。具体原因可以参考iOS中基于CADisplayLink的FPS指示器详解 (https://www.jianshu.com/p/86705c95c224)。

所有代码请参考：DorameonKit/Core/Plugin/FPS

3.2：CPU展示

CPU展示

CPU是移动设备的运算核心和控制核心，如果我们的App的使用率长时间处于高消耗的话，我们的手机会发热，电量使用加剧，导致App产生卡顿，严重影响用户体验。所以对于CPU使用率进行实时的监控，也有利于及时的把控我们App的整体质量，阻止不合格的功能上线。

对于app使用率的获取，网上的方案还是比较统一的。

1. 使用task_threads函数，获取当前App行程中所有的线程列表。
2. 对于第一步中获取的线程列表进行遍历，通过thread_info函数获取每一个非闲置线程的cpu使用率，进行相加。
3. 使用vm_deallocate函数释放资源。

代码实现如下：

```
+ (CGFloat)cpuUsageForApp {
    kern_return_t kr;
    thread_array_t         thread_list;
    mach_msg_type_number_t thread_count;
    thread_info_data_t     thinfo;
    mach_msg_type_number_t thread_info_count;
    thread_basic_info_t basic_info_th;

    // get threads in the task
    //  获取当前进程中 线程列表
    kr = task_threads(mach_task_self(), &thread_list, &thread_count);
    if (kr != KERN_SUCCESS)
        return -1;

    float tot_cpu = 0;

    for (int j = 0; j < thread_count; j++) {
        thread_info_count = THREAD_INFO_MAX;
        //获取每一个线程信息
        kr = thread_info(thread_list[j], THREAD_BASIC_INFO,
                         (thread_info_t)thinfo, &thread_info_count);
        if (kr != KERN_SUCCESS)
            return -1;

        basic_info_th = (thread_basic_info_t)thinfo;
        if (!(basic_info_th->flags & TH_FLAGS_IDLE)) {
            // cpu_usage : Scaled cpu usage percentage. The scale factor is
 TH_USAGE_SCALE.
            //宏定义TH_USAGE_SCALE返回CPU处理总频率：
            tot_cpu += basic_info_th->cpu_usage / (float)TH_USAGE_SCALE;
        }

    } // for each thread

    // 注意方法最后要调用 vm_deallocate，防止出现内存泄漏
    kr = vm_deallocate(mach_task_self(), (vm_offset_t)thread_list, thread_c
ount * sizeof(thread_t));
    assert(kr == KERN_SUCCESS);

    return tot_cpu;
}
```

测试结果基本和Xcode测量出来的cpu使用率是一样的，还是比较准确的。

所有代码请参考：DorameonKit/Core/Plugin/CPU

3.3：内存展示

内存展示

设备内存和CPU一样都是系统中最稀少的资源，也是最有可能产生竞争的资源，应用内存跟app的性能直接相关。如果一个app在前台消耗内存过多，会引起系统强杀，这种现象叫做OOM。表现跟crash一样，而且这种crash事件无法被捕获到的。

获取app消耗的内存，刚开始使用的是获取使用的物理内存大小resident_size,网上大部分也是这种方案。

```
- (NSUInteger)getResidentMemory{
    struct mach_task_basic_info info;
    mach_msg_type_number_t count = MACH_TASK_BASIC_INFO_COUNT;

    int r = task_info(mach_task_self(), MACH_TASK_BASIC_INFO, (task_info_t)
& info, & count);
    if (r == KERN_SUCCESS)
    {
        return info.resident_size;
    }
    else
    {
        return -1;
    }
}
```

使用这种方式之后方向，会与Xcode自带统计内存消耗的工具有一些偏差。这个时候，多谢yxjxx同学提交的PRuse phys_footprint to get instruments memory usage (https://github.com/didi/DoraemonKit/pull/2)。使用phys_footprint代替resident_size获取的内存消耗基本与Xcode自带的统计工具相同。具体原因可以参考正确地获取 iOS 应用占用的内存 (http://www.samirchen.com/ios-app-memory-usage/)。

修改之后，具体实现的主要代码如下：

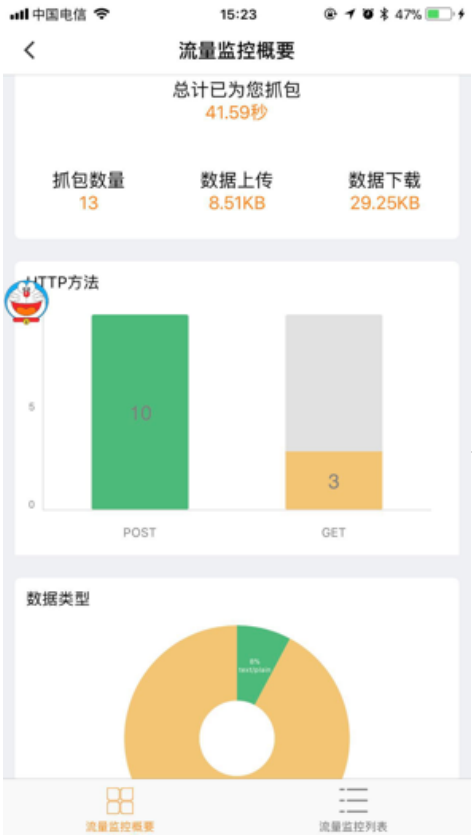
```
//当前app消耗的内存
+ (NSUInteger)useMemoryForApp{
    task_vm_info_data_t vmInfo;
    mach_msg_type_number_t count = TASK_VM_INFO_COUNT;
    kern_return_t kernelReturn = task_info(mach_task_self(), TASK_VM_INFO,
(task_info_t) &vmInfo, &count);
    if(kernelReturn == KERN_SUCCESS)
    {
        int64_t memoryUsageInByte = (int64_t) vmInfo.phys_footprint;
        return memoryUsageInByte/1024/1024;
    }
    else
    {
        return -1;
    }
}

//设备总的内存
+ (NSUInteger)totalMemoryForDevice{
    return [NSProcessInfo processInfo].physicalMemory/1024/1024;
}
```

所有代码请参考：DorameonKit/Core/Plugin/Memory

3.4：流量监控

流量监控1

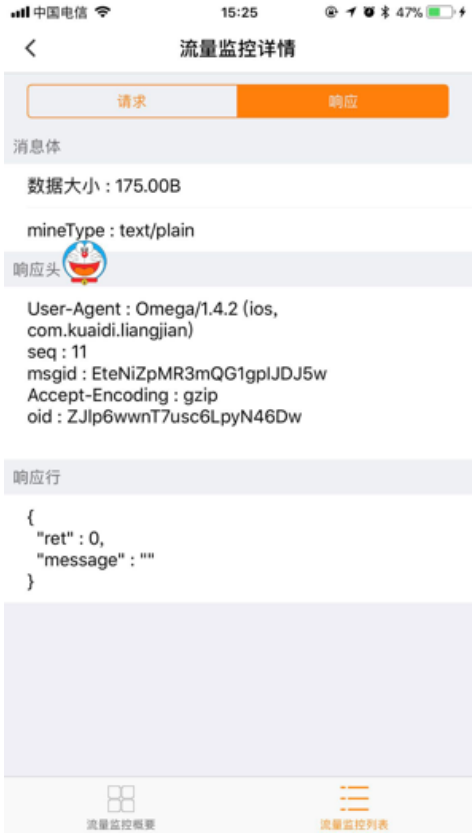

中国电信
15:23
47%
流量监控概要
总计已为您抓包
41.59秒
抓包数量
13
数据上传
8.51KB
数据下载
29.25KB
HTTP方法
10
3
POST
GET
数据类型
流量监控概要
流量监控列表

流量监控2


中国电信
15:23
47%
流量监控列表
支持筛选
https://omgup.xiaojukeji.com/api/stat/ios
POST > text/plain [200]
2018-05-10 15:23:05 耗时:0.064425s
↑ 2.08KB↓ 175.00B
https://test.kuaidadi.com:9006/data/djmapposition.node?cityId=330100
GET > application/json [200]
2018-05-10 15:23:05 耗时:0.024902s
↑ 190.00B↓ 1.17KB
POST > application/json [200]
2018-05-10 15:23:03 耗时:0.083534s
↑ 651.00B↓ 421.00B
https://test.kuaidadi.com:9006/data/djmapposition.node?cityId=330100
GET > application/json [200]
2018-05-10 15:23:02 耗时:0.019119s
↑ 190.00B↓ 1.17KB
流量监控概要
流量监控列表

流量监控3

流量监控4



流量监控5

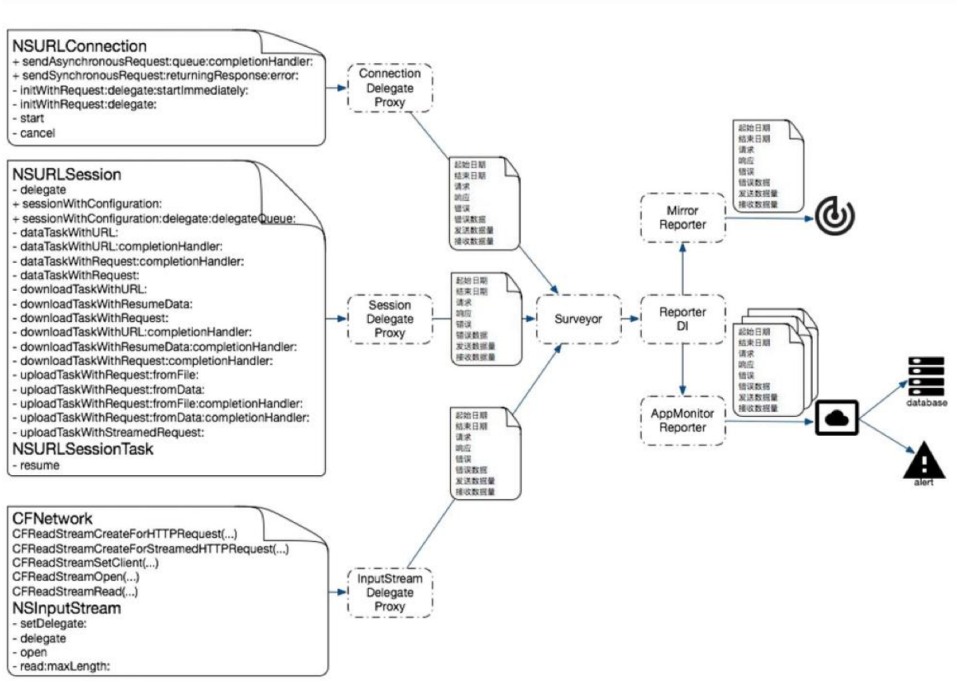
在线下开发阶段，我们开发要和服务端联调结果，我们需要Xcode断点调试服务器返回的结果是否正确。测试阶段，测试同学会通过Charles设置代理查看结果，这些操作都需要依赖第三方工具才能实现流量监控。能不能有一个工具，能够随身携带，对流量进行监控拦截，能够方便我们很多。我们DoraemonKit就做了这件事。

对于流量监控，业界基本有以上几个方案:

- 方案1 : 腾讯GT的方案，监控系统的上行流量和下行流量。这样监控的话，力度太粗了，不能得到每一个app的流量统计，更不能的得到每一个接口的流量和统计，不符合我们的需求。

- 方案2 : 浸入业务方自己的网路库，做流量统计，这种方案可以做的非常细节，但是不是特别通用。我们公司内部omega监控平台就是这么做的，omega的流量监控代码是写在OneNetworking中的。不是特别通用。比如我们杭州团队的网路库是自研的，如果要接入omega的网络监控功能，就需要在自己的网络库中，写流量统计代码。

- 方案3 : hook系统底层网络库，这种方式比较通用，但是非常繁琐，需要hook很多个类和方法。阿里有篇文档化介绍了他们流量监控的方案，就是采用这种，下面这张图我截取过来的，看一下，还是比较复杂的。



network_monitor.jpeg

- 方案4 : 也是DoraemonKit采用的方案，使用iOS中一个非常强大的类，叫NSURLProtocol，这个类可以拦截NSURLConnection、NSUrlSession、UIWebView中所有的网络请求，获取每一个网络请求的request和response对象。但是这个类无法拦截tcp的请求，这个是他的缺点。美团的内部监控工具赫兹就是基于该类进行处理的。之余这个类具体怎么使用，由于时间原因，我在这里就不说，我想大家推荐一下我的博客，我有篇文章 (https://www.jianshu.com/p/03ddcfe5ebd7)专门写了这个类的使用。

下面就是DoraemonKit中NSURLProtocol的具体实现：

```
@interface DoraemonNSURLProtocol()<NSURLConnectionDelegate,NSURLConnectionD
ataDelegate>

@property (nonatomic, strong) NSURLConnection *connection;
@property (nonatomic, assign) NSTimeInterval startTime;
@property (nonatomic, strong) NSURLResponse *response;
@property (nonatomic, strong) NSMutableData *data;
@property (nonatomic, strong) NSError *error;

@end

@implementation DoraemonNSURLProtocol

+ (BOOL)canInitWithRequest:(NSURLRequest *)request{
    if ([NSURLProtocol propertyForKey:kDoraemonProtocolKey inRequest:reques
t]) {
        return NO;
    }
    if (![DoraemonNetFlowManager shareInstance].canIntercept) {
        return NO;
    }
    if (![request.URL.scheme isEqualToString:@"http"] &&
        ![request.URL.scheme isEqualToString:@"https"]) {
        return NO;
    }
    //NSLog(@"DoraemonNSURLProtocol == %@",request.URL.absoluteString);
    return YES;
}

+ (NSURLRequest *)canonicalRequestForRequest:(NSURLRequest *)request{
    //NSLog(@"canonicalRequestForRequest");
    NSMutableURLRequest *mutableReqeust = [request mutableCopy];
    [NSURLProtocol setProperty:@YES forKey:kDoraemonProtocolKey inRequest:m
utableReqeust];
    return [mutableReqeust copy];
}

- (void)startLoading{
    //NSLog(@"startLoading");
    self.connection = [[NSURLConnection alloc] initWithRequest:[[self class
] canonicalRequestForRequest:self.request] delegate:self];
    [self.connection start];
    self.data = [NSMutableData data];
    self.startTime = [[NSDate date] timeIntervalSince1970];
}

- (void)stopLoading{
    //NSLog(@"stopLoading");
    [self.connection cancel];
    DoraemonNetFlowHttpModel *httpModel = [DoraemonNetFlowHttpModel dealWit
hResponseData:self.data response:self.response request:self.request];
    if (!self.response) {
        httpModel.statusCode = self.error.localizedDescription;
    }
    httpModel.startTime = self.startTime;
    httpModel.endTime = [[NSDate date] timeIntervalSince1970];

    httpModel.totalDuration = [NSString stringWithFormat:@"%f",[[NSDate dat
e] timeIntervalSince1970] - self.startTime];
    [[DoraemonNetFlowDataSource shareInstance] addHttpModel:httpModel];
}
```

```
#pragma mark - NSURLConnectionDelegate
- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError
*)error{
    [[self client] URLProtocol:self didFailWithError:error];
    self.error = error;
}

- (BOOL)connectionShouldUseCredentialStorage:(NSURLConnection *)connection
{
    return YES;
}

- (void)connection:(NSURLConnection *)connection didReceiveAuthenticationCh
allenge:(NSURLAuthenticationChallenge *)challenge {
    [[self client] URLProtocol:self didReceiveAuthenticationChallenge:chall
enge];
}

- (void)connection:(NSURLConnection *)connection didCancelAuthenticationCha
llenge:(NSURLAuthenticationChallenge *)challenge {
    [[self client] URLProtocol:self didCancelAuthenticationChallenge:challe
nge];
}

#pragma mark - NSURLConnectionDataDelegate
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLR
esponse *)response{
    [[self client] URLProtocol:self didReceiveResponse:response cacheStorag
ePolicy:NSURLCacheStorageAllowed];
    self.response = response;
}

- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)d
ata{
    [[self client] URLProtocol:self didLoadData:data];
    [self.data appendData:data];
}

- (NSCachedURLResponse *)connection:(NSURLConnection *)connection willCache
Response:(NSCachedURLResponse *)cachedResponse{
    return cachedResponse;
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection {
    [[self client] URLProtocolDidFinishLoading:self];
}
```

所有代码请参考：DorameonKit/Core/Plugin/NetFlow

3.5：自定义监控

自定义监控

以上所有的操作都是针对于单个指标，无法提供一套全面的监控数据，自定义监控可以选择你需要监控的数据，目前包括帧率、CPU使用率、内存使用量和流量监控，这些监控没有波形图进行显示，均在后台进行监控，测试完毕，会把这些数据上传到我们后台进行分析。

因为目前后台是基于我们内部平台上开发的，暂时不提供开源。不过后续的话，我们也会考虑将后台的功能的功能对外提供，请大家拭目以待。对于开源版本的话，目前性能测试的结果保存在沙盒Library/Caches/DoraemonPerformance中，使用者可以使用沙盒浏览器功能导出来之后自己进行分析。

所有代码请参考：DorameonKit/Core/Plugin/AllTest

三、总结

写这篇文章主要是为了能够让大家对于DorameonKit进行快速的了解，大家如果有什么好的想法，或者发现我们的这个项目有bug，欢迎大家去github上提Issues或者直接Pull requests，我们会第一时间处理，也可以加入我们的qq交流群进行交流，也希望我们这个工具集合能在大家的一起努力下，继续做大做好。

如果大家觉得我们这个项目还可以的话，点上一颗star吧。

DoraemonKit项目地址：https://github.com/didi/DoraemonKit (https://github.com/didi/DoraemonKit)

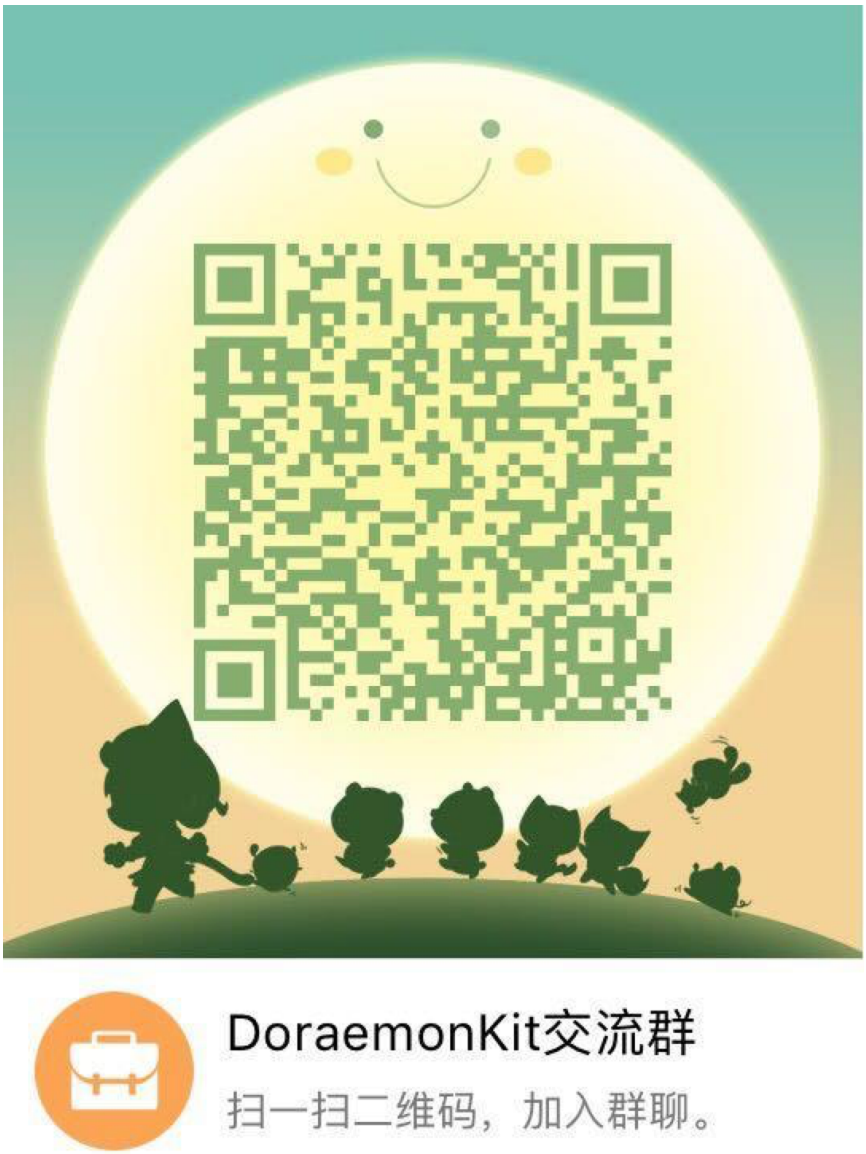
四、参考文章

iOS中基于CADisplayLink的FPS指示器详解 (https://www.jianshu.com/p/86705c95c224)

iOS-Monitor-Platform (https://github.com/aozhimin/iOS-Monitor-Platform)

正确地获取 iOS 应用占用的内存 (http://www.samirchen.com/ios-app-memory-usage/)

五、交流群



QQ交流群

小礼物走一走，来简书关注我

赞赏支持

日记本 (/nb/3262665)

景铭巴巴 (/u/c3c893a27097)
写了 26441 字，被 1822 人关注，获得了 1251 个喜欢
(/u/c3c893a27097)

+关注

iOS开发者

喜欢 | 23

更多分享

登录 (/sign后发表评论source=desktop&utm_medium=not-signed-in-com

2条评论 只看作者 按时间倒序 按时间正序

helloDolin (/u/6c0a7aeb2e84)
3楼 · 2019.04.29 18:23
(/u/6c0a7aeb2e84)
mark

赞 回复

开发者头条_程序员必装的App (/u/6a1613a5b777)
2楼 · 2018.11.23 13:49
(/u/6a1613a5b777)
感谢分享！已推荐到《开发者头条》： https://toutiao.io/posts/qwkjyp
(https://toutiao.io/posts/qwkjyp) 欢迎点赞支持！
使用开发者头条 App 搜索 376974 即可订阅《易翔的独家号》

赞 回复

被以下专题收入，发现更多相似内容

移动前沿 (/c/5aac963ca52d?utm_source=desktop&utm_medium=notes-included-collection)

iOS精英班 (/c/e5d51bd792c6?utm_source=desktop&utm_medium=notes-included-collection)

iOS精选 (/c/a2ebf109812e?utm_source=desktop&utm_medium=notes-included-collection)

Vender (/c/a332fe5ffd48?utm_source=desktop&utm_medium=notes-included-collection)

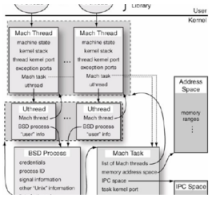
iOS开发攻城... (/c/de77ae960f88?utm_source=desktop&utm_medium=notes-included-collection)

iOS性能优化 (/c/4fbf5bbb5606?utm_source=desktop&utm_medium=notes-included-collection)

iOS 性能 (/c/1e457faaee2c?utm_source=desktop&utm_medium=notes-included-collection)

展开更多

(/p/73c55ed83e46?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendatio

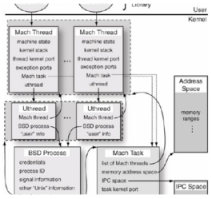
iOS内存优化【转载】(/p/73c55ed83e46?utm_campaign=maleskine&...

APP的性能监控包括： CPU 占用率、 内存使用情况、网络状况监控、启动时闪退、卡顿、FPS、使用时崩溃、耗电量监控、流量监控等等。 文中所有代码都已同步到github中，有兴趣的可以clone下来一起探讨

it彭于晏 (/u/4162bc5e399d?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendatio

---

(/p/95df83780c8f?



utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendatio

iOS开发--APP性能检测方案汇总(一) (/p/95df83780c8f?utm_campaign...

APP的性能监控包括： CPU 占用率、 内存使用情况、网络状况监控、启动时闪退、卡顿、FPS、使用时崩溃、耗电量监控、流量监控等等。 文中所有代码都已同步到github中，有兴趣的可以clone下来一起探讨

青苹果园 (/u/b534ce5f8fae?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendatio

---

负离子 (/p/24bf037ecc51?utm_campaign=maleskine&utm_content=n...

放肆呼吸 无畏想法 不是所有的旅行都能说走就走 不是所有的口罩都能爱不释手。 污染的时代，废气尾气蔓延，你敢放肆呼吸吗？ 随我的时代，个性率性张扬，你还畏惧空气吗？ 负离子生态口罩（带阀） 放肆

 睿世堡 (/u/ff3b1f0fb7f8?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendatio

---

《风和人》 (/p/014bfd766869?utm_campaign=maleskine&utm_conte...

《风和人》 -- 杏子 孤寂的风 孤独的人 风吹过人的耳旁在呢喃 人享受着风吹过的感觉 风在人旁咆哮 人在风中思考 咆哮明天该要往哪吹 思考明天路该怎么走 它彷徨 他迷茫 直到太阳落下 风停止了 人睡着了 直到

 一种缺陷ing (/u/c393c34664ec?utm_campaign=maleskine&utm_content=user&utm_medium=seo_notes&utm_source=recommendatio