18-怎么减少App电量消耗？

你好，我是戴铭。

手机设备电量有限，App 开发时如不注意电量的的消耗，当用户发现你的 App 是耗电大户时，就会毫不犹豫地将其抛弃。所以，每次开发完，我们都需要去检查自己的App有没有耗电的问题。

耗电的原因有千万种，如果每次遇到耗电过多的问题，我们都从头查找一番的话，那必然会效率低下。

就比如说，测试同学过来跟你说“某个页面的前一个版本还好好的，这个版本的耗电怎么多了那么多”，那么你首先想到可能就是这个页面有没有开启定位，网络请求是不是频繁，亦或是定时任务时间是不是间隔过小。接下来，你会去查找耗电问题到底是怎么引起的。你去翻代码的时候却发现，这个页面的相关功能在好几个版本中都没改过了。

那么，到底是什么原因使得这一个版本的耗电量突然增加呢？不如就使用排除法吧，你把功能一个个都注释掉，却发现耗电量还是没有减少。这时，你应该怎么办呢？接下来，我就在今天的文章里面和你详细分享一下这个问题的解法吧。

我们首先需要明确的是，只有获取到电量，才能够发现电量问题。所以，我就先从如何获取电量和你讲起。

如何获取电量？

在iOS中，IOKit framework 是专门用于跟硬件或内核服务通信的。所以，我们可以通过IOKit framework 来获取硬件信息，进而获取到电量消耗信息。在使用IOKit framework时，你需要：

- 首先，把IOPowerSources.h、IOPSKeys.h和IOKit 这三个文件导入到工程中；
- 然后，把batteryMonitoringEnabled置为true；
- 最后，通过如下代码获取1%精确度的电量信息。

```
#import "IOPSKeys.h"
#import "IOPowerSources.h"

-(double) getBatteryLevel{
    // 返回电量信息
    CFTypeRef blob = IOPSCopyPowerSourcesInfo();
    // 返回电量句柄列表数据
    CFArrayRef sources = IOPSCopyPowerSourcesList(blob);
    CFDictionaryRef pSource = NULL;
    const void *psValue;
    // 返回数组大小
    int numOfSources = CFArrayGetCount(sources);
    // 计算大小出错处理
    if (numOfSources == 0) {
        NSLog(@"Error in CFArrayGetCount");
        return -1.0f;
    }

    // 计算所剩电量
    for (int i=0; i<numOfSources; i++) {
        // 返回电源可读信息的字典
        pSource = IOPSGetPowerSourceDescription(blob, CFArrayGetValueAtIndex(sources, i));
        if (!pSource) {
```

```
            NSLog(@"Error in IOPSGetPowerSourceDescription");
            return -1.0f;
        }
        psValue = (CFStringRef) CFDictionaryGetValue(pSource, CFSTR(kIOPSNameKey));

        int curCapacity = 0;
        int maxCapacity = 0;
        double percentage;

        psValue = CFDictionaryGetValue(pSource, CFSTR(kIOPSCurrentCapacityKey));
        CFNumberGetValue((CFNumberRef)psValue, kCFNumberSInt32Type, &curCapacity);

        psValue = CFDictionaryGetValue(pSource, CFSTR(kIOPSMaxCapacityKey));
        CFNumberGetValue((CFNumberRef)psValue, kCFNumberSInt32Type, &maxCapacity);

        percentage = ((double) curCapacity / (double) maxCapacity * 100.0f);
        NSLog(@"curCapacity : %d / maxCapacity: %d , percentage: %.1f ", curCapacity, maxCapacity, percenta
        return percentage;
    }
    return -1.
```

说完耗电量的获取方法，我们再继续看如何解决电量问题。

如何诊断电量问题？

回到最开始的问题，当你用排除法将所有功能注释掉后，如果还有问题，那么这个耗电一定是由其他线程引起的。创建这个耗电线程的地方可能是在其他地方，比如是由第三方库引起，或者是公司其他团队开发的二方库。

所以，你需要逆向地去思考这个问题。这里，你不妨回顾一下，我们在第12篇文章“iOS崩溃千奇百怪，如何全面监控”中是怎么定位问题的。

也就是说，我们还是先反过来看看出现电量问题的期间，哪个线程是有问题的。通过下面的这段代码，你就可以获取到所有线程的信息：

```
thread_act_array_t threads;
mach_msg_type_number_t threadCount = 0;
const task_t thisTask = mach_task_self();
kern_return_t kr = task_threads(thisTask, &threads, &threadCount);
```

从上面代码可以看出，通过 task_threads 函数，我们就能够得到所有的线程信息数组 threads，以及线程总数 threadCount。threads 数组里的线程信息结构体 thread_basic_info 里有一个记录 CPU 使用百分比的字段 cpu_usage。thread_basic_info结构体的代码如下：

```
struct thread_basic_info {
        time_value_t    user_time;      /* user 运行的时间 */
        time_value_t    system_time;    /* system 运行的时间 */
        integer_t       cpu_usage;      /* CPU 使用百分比 */
        policy_t        policy;         /* 有效的计划策略 */
        integer_t       run_state;      /* run state (see below) */
```

```
        integer_t       flags;          /* various flags (see below) */
        integer_t       suspend_count;  /* suspend count for thread */
        integer_t       sleep_time;     /* 休眠时间 */
};
```

有了这个 cpu_usage 字段，你就可以通过遍历所有线程，去查看是哪个线程的 CPU 使用百分比过高了。如果某个线程的CPU使用率长时间都比较高的话，比如超过了90%，就能够推断出它是有问题的。这时，将其方法堆栈记录下来，你就可以知道到底是哪段代码让你 App 的电量消耗多了。

通过这种方法，你就可以快速定位到问题，有针对性地进行代码优化。多线程 CPU 使用率检查的完整代码如下：

```
// 轮询检查多个线程 CPU 情况
+ (void)updateCPU {
    thread_act_array_t threads;
    mach_msg_type_number_t threadCount = 0;
    const task_t thisTask = mach_task_self();
    kern_return_t kr = task_threads(thisTask, &threads, &threadCount);
    if (kr != KERN_SUCCESS) {
        return;
    }
    for (int i = 0; i < threadCount; i++) {
        thread_info_data_t threadInfo;
        thread_basic_info_t threadBaseInfo;
        mach_msg_type_number_t threadInfoCount = THREAD_INFO_MAX;
        if (thread_info((thread_act_t)threads[i], THREAD_BASIC_INFO, (thread_info_t)threadInfo, &threadInfo
            threadBaseInfo = (thread_basic_info_t)threadInfo;
            if (!(threadBaseInfo->flags & TH_FLAGS_IDLE)) {
                integer_t cpuUsage = threadBaseInfo->cpu_usage / 10;
                if (cpuUsage > 90) {
                    //cup 消耗大于 90 时打印和记录堆栈
                    NSString *reStr = smStackOfThread(threads[i]);
                    //记录数据库中
                    [[[SMLagDB shareInstance] increaseWithStackString:reStr] subscribeNext:^(id x) {}];
                    NSLog(@"CPU useage overload thread stack: \n%@",reStr);
                }
            }
        }
    }
}
```

优化电量

现在我们已经知道了在线上碰到电量问题时，应该如何解决，但是电量的不合理消耗也可能来自其他方面。CPU 是耗电的大头，引起 CPU 耗电的单点问题可以通过监控来解决，但点滴汇聚终成大海，每一个不合理的小的电量消耗，最终都可能会造成大的电量浪费。所以，我们在平时的开发工作中，时刻关注对耗电量的优化也非常重要。

对 CPU 的使用要精打细算，要避免让 CPU 做多余的事情。对于大量数据的复杂计算，应该把数据传到服务器去处理，如果必须要在 App 内处理复杂数据计算，可以通过 GCD 的 dispatch_block_create_with_qos_class 方法指定队列的 Qos 为 QOS_CLASS_UTILITY，将计算工作放到这个队列的 block 里。在 QOS_CLASS_UTILITY 这种 Qos 模式下，系统针对大量数据的计算，以及复杂数据处理

专门做了电量优化。

接下来，我们再看看**除了 CPU 会影响耗电，对电量影响较大的因素还有哪些呢？**

除了 CPU，I/O操作也是耗电大户。任何的 I/O 操作，都会破坏掉低功耗状态。那么，针对 I/O 操作要怎么优化呢？

业内的普遍做法是，将碎片化的数据磁盘存储操作延后，先在内存中聚合，然后再进行磁盘存储。碎片化的数据进行聚合，在内存中进行存储的机制，可以使用系统自带的 NSCache 来完成。

NSCache 是线程安全的，NSCache 会在到达预设缓存空间值时清理缓存，这时会触发 cache:willEvictObject: 方法的回调，在这个回调里就可以对数据进行 I/O 操作，达到将聚合的数据 I/O 延后的目的。I/O 操作的次数减少了，对电量的消耗也就减少了。

SDWebImage 图片加载框架，在图片的读取缓存处理时没有直接使用 I/O，而是使用了NSCache。使用 NSCache 的相关代码如下：

```
- (UIImage *)imageFromMemoryCacheForKey:(NSString *)key {
    return [self.memCache objectForKey:key];
}

- (UIImage *)imageFromDiskCacheForKey:(NSString *)key {
    // 检查 NSCache 里是否有
    UIImage *image = [self imageFromMemoryCacheForKey:key];
    if (image) {
        return image;
    }
    // 从磁盘里读
    UIImage *diskImage = [self diskImageForKey:key];
    if (diskImage && self.shouldCacheImagesInMemory) {
        NSUInteger cost = SDCacheCostForImage(diskImage);
        [self.memCache setObject:diskImage forKey:key cost:cost];
    }
    return diskImage;
}
```

可以看出，SDWebImage 将获取的图片数据都放到了 NSCache 里，利用 NSCache 缓存策略进行图片缓存内存的管理。每次读取图片时，会检查 NSCache 是否已经存在图片数据：如果有，就直接从 NSCache 里读取；如果没有，才会通过 I/O 读取磁盘缓存图片。

使用了 NSCache 内存缓存能够有效减少 I/O 操作，你在写类似功能时也可以采用这样的思路，让你的 App 更省电。

CPU 和 I/O 这两大耗电问题都解决后，还有什么要注意的呢？这里还有两份关于App电量消耗的资料，你可以对照你的App来查看。

苹果公司专门维护了一个电量优化指南“Energy Efficiency Guide for iOS Apps”，分别从 CPU、设备唤醒、网络、图形、动画、视频、定位、加速度计、陀螺仪、磁力计、蓝牙等多方面因素提出了电量优化方面的建议。所以，当使用了苹果公司的电量优化指南里提到的功能时，严格按照指南里的最佳实践去做就能够

保证这些功能不会引起不合理的电量消耗。

同时，苹果公司在2017年 WWDC 的 Session 238 也分享了一个关于如何编写节能 App 的主题“Writing Energy Efficient Apps”。

小结

今天我跟你分享了如何通过获取线程信息里的cpu_usage 字段来判断耗电线程，进而得到当前线程执行方法堆栈，从而精准、快速地定位到引起耗电的具体方法。我曾经用这个方法解决了几起难以定位的耗电问题，这些问题都出在二方库上。通过获取到的方法堆栈，我就有了充足的证据去推动其他团队进行电量优化。

除此之外，我还跟你介绍了如何在平时开发中关注电量的问题。在我看来，减少 App 耗电也是开发者的天职，不然如何向我们可爱的用户交代呢。

课后小作业

请你使用我今天分享的耗电检查方法，检查一下你的 App，看看哪个方法最耗电。

感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎把它分享给更多的朋友一起阅读。

精选留言：

- Ant 2019-04-21 12:40:09
  受教了

- 泽七 2019-04-20 16:03:52
  获取电量为什么不用 [[UIDevice currentDevice] batteryLevel] 呢？

- 黄昏 2019-04-20 10:04:12
  cpu使用量确实是耗电的重大原因。不合理的动画可能导致cpu暴涨，电量损耗巨大。