《高性能iOS 应用开发》之降低你 APP 的电量消耗

在编写高性能 代码时, 电量消耗是一个需要重点处理的重要因素, 就执行时间和 CPU 资源的利用而言, 我们不仅要实现高效的数据结构和算法, 还需要考虑其他的因素,如果某个应用是个电池黑洞,那么一定不会有人喜欢他 电量消耗除了 CPU 外,还有一些硬件模块:网络硬件, 蓝牙,GPS, 麦克风,加速计,摄像头,扬声器,和屏幕. 我们可以带着以下问题来看这篇文章:

- 消耗电量的关键领域有哪些
- 如何降低电量的消耗
- 如何在 IOS 应用中分析电源, CPU 和资源的使用

— CPU

不论用户是否正在直接使用, CPU 都是应用所使用的主要硬件, 在后台操作和处理推送通知时, 应用仍然会消耗 CPU 资源

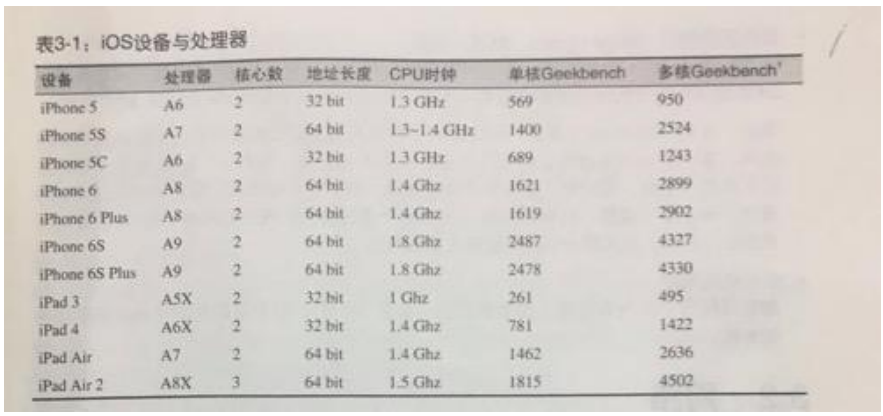


表3-1：iOS设备与处理器

设备	处理器	核心数	地址长度	CPU时钟	单核Geekbench	多核Geekbench'
iPhone 5	A6	2	32 bit	1.3 GHz	569	950
iPhone 5S	A7	2	64 bit	1.3~1.4 GHz	1400	2524
iPhone 5C	A6	2	32 bit	1.3 GHz	689	1243
iPhone 6	A8	2	64 bit	1.4 Ghz	1621	2899
iPhone 6 Plus	A8	2	64 bit	1.4 Ghz	1619	2902
iPhone 6S	A9	2	64 bit	1.8 Ghz	2487	4327
iPhone 6S Plus	A9	2	64 bit	1.8 Ghz	2478	4330
iPad 3	A5X	2	32 bit	1 Ghz	261	495
iPad 4	A6X	2	32 bit	1.4 Ghz	781	1422
iPad Air	A7	2	64 bit	1.4 Ghz	1462	2636
iPad Air 2	A8X	3	64 bit	1.5 Ghz	1815	4502

应用计算的越多,消耗的电量越多.在完成相同的基本操作时, 老一代的设备会消耗更多的电量(换电池呀 哈哈哈 开个玩笑),计算量的消耗取决于不同的因素

- 对数据的处理
- 待处理的数据大小---更大的显示屏允许软件在单个视图中展示更多的信息,但这也意味着要处理更多的数据
- 处理数据的算法和数据结构
- 执行更新的次数,尤其是在数据更新后,触发应用的状态或 UI 进行更新(应用收到的推送通知也会导致数据更新,如果此用户正在使用应用,你还需要更新 UI)

没有单一原则可以减少设备中的执行次数,很多规则都取决于操作的本质, 以下是一些可以在应用中投入使用的最佳实践

- 针对不同的情况选择优化的算法 例如,当你在排序时,如果列表少于43个实例, 则插入排序优于归并排序, 但实例对于286时, 应当使用快速排序,要优先使用双枢轴快速排序而不是传统的单枢轴快速排序
- 如果应用从服务器接受数据,尽量减少需要在客户端进行的处理 例如如果一段文字需要在客户端进行渲染,尽可能在服务器将数据清理干净 我曾经做个一个项目, 因为服务器的实现主要用于服务桌面用户,所以返回的文本中包含 HTML 标签, 清理 HTML 标签的工作并没有放在客户端进行, 而是放在了服务端实现,从而减少了设备上的计算过程, 降低了处理时间
- 优化静态编译(ahead-of-time,AOT)处理 动态编译处理的缺点在于他会强制用户等待操作完成, 但是激进的 AOT 处理则会导致计算资源的浪费, 需要根据应用和设备选择精确定量的 AOT 处理. 例如,在 UITableView 中渲染一组记录时,在载入列表是处理全部的记录并不是明智之举,基于单元格的高度,如果设备可以渲染 N 条记录, 那么3N 或4N 则是一个理想的数据载入规模, 类似的,用户快速滑动,则不应立即载入记录,而应推迟带滚动速度下降到某一阈值.精确的阈值应该由每个单元格的处理时间和单元格的 UI 的复杂性来决定

二 网络

智能的网络访问管理可以让应用响应的更快,并有助于延长电池寿命.在无法访问网络时,应该推迟后续的网络请求, 直到网络连接恢复为止. 此外,应避免在没有连接 WiFi 的情况下进行高宽带消耗的操作.比如视频流, 众所周知, 蜂窝无线系统(LTE,4G,3G等)对电量的消耗远远大于 WiFi信号, 根源在于 LTE 设备基于多输入,多输出技术,使用多个并发信号以维护两端的 LTE 链接,类似的,所有的蜂窝数据链接都会定期扫描以寻找更强的信号. 因此:我们需要

- 在进行任何网络操作之前,先检查合适的网络连接是否可用
- 持续监视网络的可用性,并在链接状态发生变化时给与适当的反馈

三 定位管理器和 GPS

> 这个知识点我项目中并没有用到**定位相关**的功能 ,不过也总结一下书中所讲的知识点 有用的定位功能的朋友可以参考此知识点来优化自己的 app

我们都知道定位服务是很耗电的,使用 GPS 计算坐标需要确定两点信息:

- 时间锁 每个 GPS 卫星每毫秒广播唯一一个1023位随机数, 因而数据传播速率是1.024Mbit/s GPS 的接收芯片必须正确的与卫星的时间锁槽对齐
- 频率锁 GPS 接收器必须计算由接收器与卫星的相对运动导致的多普勒偏移带来的信号误差

计算坐标会不断的使用 CPU 和 GPS 的硬件资源,因此他们会迅速的消耗电池电量 先来看一下初始化 `CLLocationManager` 并高效接受地理位置更新的典型代码

```
#import "LLLocationViewController.h"
#import <CoreLocation/CoreLocation.h>

@interface LLLocationViewController ()<CLLocationManagerDelegate>
@property (nonatomic, strong)CLLocationManager *manager;
```

```
@end

@implementation LLLocationViewController

- (void)viewDidLoad {
    [super viewDidLoad];
    self.manager = [[CLLocationManager alloc]init];
    self.manager.delegate = self;
}

- (void)enableLocationButtonClick:(UIButton *)sender{

    self.manager.distanceFilter = kCLDistanceFilterNone;
    // 按照最大精度初始化管理器
    self.manager.desiredAccuracy = kCLLocationAccuracyBest;

    if (IS_IOS8) {
        [self.manager requestWhenInUseAuthorization];
    }
    [self.manager startUpdatingLocation];
}

- (void)locationManager:(CLLocationManager *)manager
     didUpdateLocations:(NSArray<CLLocation *> *)locations{

    CLLocation *loc = [locations lastObject];
    // 使用位置信息
}
```

3.1 最佳的初始化

- **distanceFilter** 只要设备的移动超过了最小的距离, 距离过滤器就会导致管理器对委托对象的 `LocationManager:didUpdateLocations:` 事件通知发生变化,该距离单位是 M
- **desiredAccuracy** 精度参数的使用直接影响了使用天线的个数, 进而影响了对电池的消耗.精度级别的选取取决于应用的具体用途,精度是一个枚举 我们应该依照不同的需求去恰当的选取精度级别

距离过滤器只是软件层面的过滤器,而精度级别会影响物理天线的使用.当委托方法

`LocationManager:didUpdateLocations:` 被调用时,使用距离范围更广泛的过渡器只会影响间隔.另一方面,更高的精度级别意味着更多的活动天线,这会消耗更多的能量

3.2 关闭无关紧要的特性

判断何时需要跟踪位置的变化, 在需要跟踪的时候调用 `startUpdatingLocation` 方法, 无须跟踪时调用 `stopUpdatingLocation` 方法.

当应用在后台运行或用户没有与别人聊天时,也应该关闭位置跟踪,也就说说,浏览媒体库,查看朋友列表或调整应用设置时, 都应该关闭位置跟踪

3.3 只在必要时使用网络

为了提高电量的使用效率, IOS 总是尽可能地保持无线网络关闭.当应用需要建立网络连接时, IOS 会利用这个机会向后台应用分享网络会话, 以便一些低优先级能够被处理, 如推送通知, 收取电子邮件等 关键在于每当用户建立网络连接时,网络硬件都会在连接完成后多维持几秒的活动时间.每次集中的网络通信都会消耗大量的电量 要想减轻这个问题带来的危害,你的软件需要有所保留的的使用网络.应该定期集中短暂的使用网络,而不是持续的保持着活动的数据流.只有这样,网络硬件才有机会关闭

3.4 后台定位服务

`CLLocationManager`提供了一个替代的方法来监听位置的更新. `[self.manager startMonitoringSignificantLocationChanges]`可以帮助你在更远的距离跟踪运动.精确的值由内部决定,且与`distanceFilter`无关 使用这一模式可以在应用进入后台后继续跟踪运动,典型的做法是在应用进入后台时执行`startMonitoringSignificantLocationChanges`方法,而当应用回到前台时执行`startUpdatingLocation`

如下代码

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    [self.manager stopUpdatingLocation];
    [self.manager startMonitoringSignificantLocationChanges];
}
- (void)applicationWillEnterForeground:(UIApplication *)application {
    [self.manager stopMonitoringSignificantLocationChanges];
    [self.manager startUpdatingLocation];
}
```

3.5 在应用关闭后重启

在其他应用需要更多资源时, 后台的应用可能会被关闭.在这种情况下, 一旦发生位置变化,应用会被重启,因而需要重新初始化监听过程,若出现这种情况,`application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions`方法会受到键值为`UIApplicationLaunchOptionsLocationKey`的条目 如下代码: 在应用关闭后重新初始化监听

```
- (void)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)la
    // 因缺乏资源而关闭应用后, 监测应用是否因为位置变化而被重启
    if (launchOptions[UIApplicationLaunchOptionsLocationKey]) {
// 开启监测位置的变化
        [self.manager startMonitoringSignificantLocationChanges];
    }
}
```

###四 屏幕 屏幕非常耗电, 屏幕越大就越耗电.当然,如果你的应用在前台运行且与用户进行交互,则势必会使用屏幕并消耗电量 这里仍然有一些方案可以优化屏幕的使用

4.1 动画

当应用在前台时, 使用动画, 一旦应用进入了后台,则立即暂停动画.通常来说,你可以通过监听`UIApplicationWillResignActiveNotification`或`UIApplicationDIdEnterBackgroundNotification`的通知事件来暂停或停止动画,也可以通过监听`UIApplicationDidBecomeActiveNotification`的通知事件来恢复动画

4.2 视频播放

> 我在上家公司就是做视频类App的,当时就采用了这个技术 保持屏幕常亮

在视频播放期间,最好保持屏幕常量.可以使用`UIApplication`对象的 `idleTimerDisabled`属性来实现这个目的.一旦设置了 YES, 他会阻止屏幕休眠,从而实现常亮. 与动画类似,你可以通过相应应用的通知来释放和获取锁

4.3 多屏幕

使用屏幕比休眠锁或暂停/恢复动画要复杂得多

如果正在播放电影或运行动画, 你可以将它们从设备的屏幕挪到外部屏幕,而只在设备的屏幕上保留最基本的设置,这样可以减少设备上的屏幕更新,进而延长电池寿命

处理这一场景的典型代码会涉及一下步骤

- 1 在启动期间监测屏幕的数量 如果屏幕数量大于1,则进行切换
- 2 监听屏幕在链接和断开时的通知. 如果有新的屏幕加入, 则进行切换. 如果所有的外部屏幕都被移除,则恢复到默认显示

```
@interface LLMultiScreenViewController ()
@property (nonatomic, strong)UIWindow  *secondWindow;
@end

@implementation LLMultiScreenViewController

- (void)viewDidAppear:(BOOL)animated{
    [super viewDidAppear:animated];
    [self updateScreens];
}

- (void)viewDidDisappear:(BOOL)animated{
    [super viewDidDisappear:animated];
    [self disconnectFromScreen];

}

- (void)viewDidLoad {
    [super viewDidLoad];
    [self registerNotifications];
}

- (void)registerNotifications{

    NSNotificationCenter *nc = [NSNotificationCenter defaultCenter];
    [nc addObserver:self selector:@selector(scrensChanged:) name:UIScreenDidConnectNotification c
}

- (void)scrensChanged:(NSNotification *)nofi{
    [self updateScreens];
}

- (void)updateScreens{

    NSArray *screens = [UIScreen screens];
    if (screens.count > 1) {
        UIScreen *secondScreen = [screens objectAtIndex:1];
        CGRect rect =secondScreen.bounds;
        if (self.secondWindow == nil) {
            self.secondWindow = [[UIWindow alloc]initWithFrame:rect];
            self.secondWindow.screen = secondScreen;

            LLScreen2ViewController *svc = [[LLScreen2ViewController alloc]init];
            svc.parent = self;
            self.secondWindow.rootViewController = svc;
        }
        self.secondWindow.hidden = NO;
    }else{
        [self disconnectFromScreen];
    }
}
```

```
- (void)disconnectFromScreen{

    if (self.secondWindow != nil) {
        // 断开连接并释放内存
        self.secondWindow.rootViewController = nil;
        self.secondWindow.hidden = YES;
        self.secondWindow = nil;
    }
}

- (void)dealloc{

    [[NSNotificationCenter defaultCenter] removeObserver:self];

}
```

五 其他硬件

当你的应用进入后台是, 应该释放对这些硬件的锁定:

- 蓝牙
- 相机
- 扬声器,除非应用是音乐类的
- 麦克风

基本规则: 只有当应用处于前台时才与这些硬件进行交互, 应用处于后台时应停止交互

不过扬声器和无线蓝牙可能例外, 如果你正在开发音乐,收音机或其他的音频类应用,则需要在应用进入后台后继续使用扬声器.不要让屏幕仅仅为音频播放的目的而保持常量.类似的, 若应用还有未完成的数据传输, 则需要在应用进入后台后持续使用无线蓝牙,例如,与其他设备传输文件

六 电池电量与代码感知

> 这一条我发现 摩拜单车小程序 做的挺好的,如果晚上骑车扫描二维码的话是需要开闪光灯达到照亮二维码的效果, 但是如果你的手机处于低电量的话 ,你的闪光灯是打不开的, 这一个细节就说明了用户体验很重要,他首先会保证不让你的手机因为闪光灯而直接关机

一个智能的应用会考虑到电池的电量和自身的状态, 从而决定是否执行资源密集消耗性的操作.另外一个有价值的点是对充电的判断,确定设备是否处于充电状态

来看一下此处的代码实施

```
- (BOOL)shouldProceedWithMinLevel:(NSUInteger)minLevel{

    UIDevice *device = [UIDevice currentDevice];
    // 打开电池监控
    device.batteryMonitoringEnabled = YES;

    UIDeviceBatteryState state = device.batteryState;
    // 在充电或电池已经充满的情况下,任何操作都可以执行
    if (state == UIDeviceBatteryStateCharging ||
        state == UIDeviceBatteryStateFull) {
        return YES;
    }
    // UIdevice 返回的 batteryLevel 的范围在0.00 ~ 1.00
    NSUInteger batteryLevel = (NSUInteger)(device.batteryLevel * 100);
    if (batteryLevel >= minLevel) {
```

```
        return YES;
    }
    return NO;
}
```

我们也可以得到应用对 CPU 的利用率

```
// 需要导入这两个头文件
#import <mach/mach.h>
#import <assert.h>

- (float)appCPUUsage{
    kern_return_t kr;
    task_info_data_t info;
    mach_msg_type_number_t infoCount = TASK_INFO_MAX;
    kr = task_info(mach_task_self(), TASK_BASIC_INFO, info, &infoCount);
    if (kr != KERN_SUCCESS) {
        return -1;
    }
    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;

    kr = task_threads(mach_task_self(), &thread_list, &thread_count);
    if (kr != KERN_SUCCESS) {
        return -1;
    }
    float tot_cpu = 0;
    int j;
    for (j = 0; j < thread_count; j++) {
        thread_info_count = THREAD_INFO_MAX;
        kr = thread_info(thread_list[j], THREAD_BASIC_INFO, 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)) {
            tot_cpu += basic_info_th -> cpu_usage / TH_USAGE_SCALE * 100.0;
        }
    }
    vm_deallocate(mach_task_self(), (vm_offset_t)thread_list, thread_count * sizeof(thread_t));
    return tot_cpu;

}
```

当剩余电量较低时,提醒用户,并请求用户授权执行电源密集型的操作,---当然,只在 用户同意的前提下执行 总是用一个指示符(也就是进度条百分比)显示长时间任务的进度, 包括设备上即将完成的计算或者只是下载一些内容.向用户提供完成进度的估算, 以帮助他们决定是否需要为设备充电

七 最佳实践

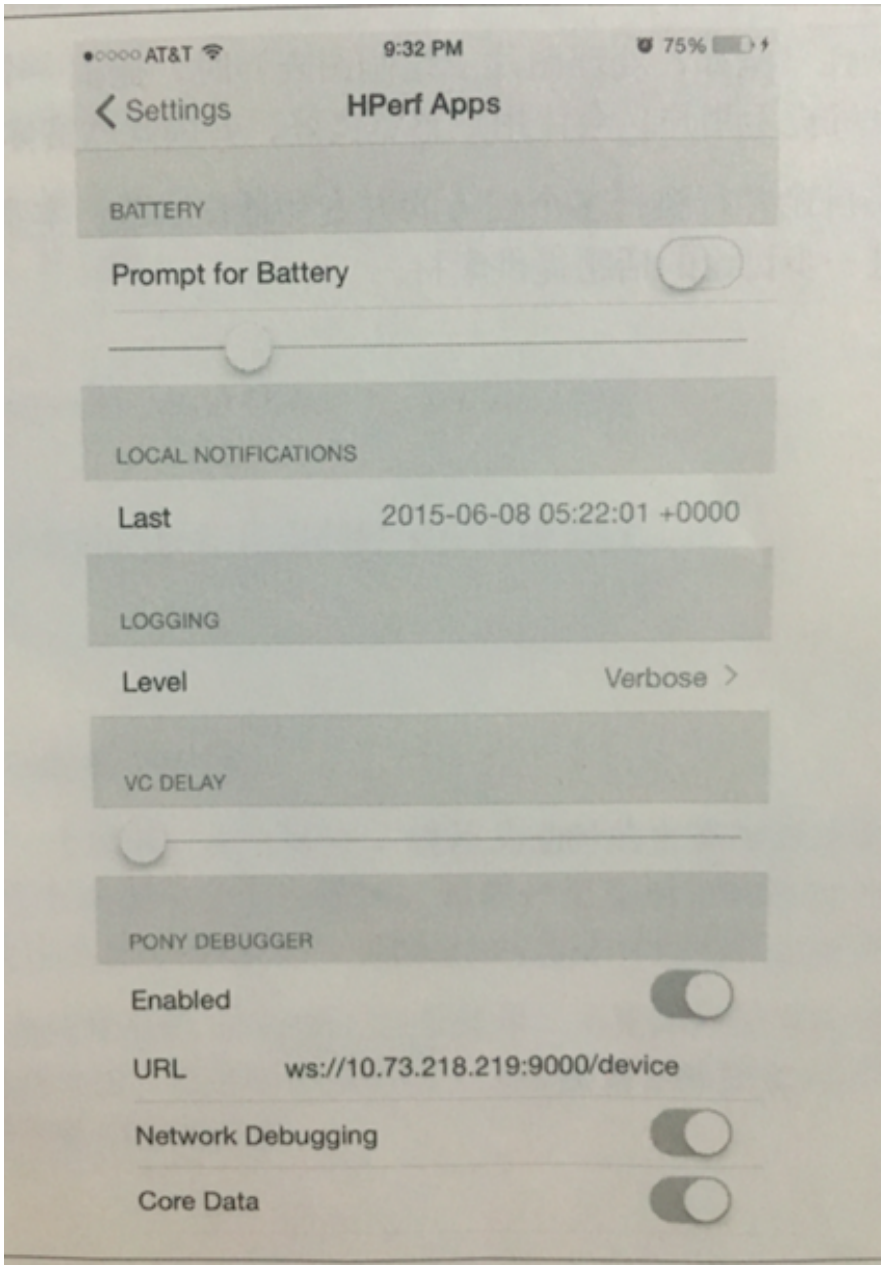
以下的最佳实践可以确保对电量的谨慎使用, 遵循以下要点,应用可以实现对电量的高效使用.

- 最小化硬件使用. 换句话说,尽可能晚的与硬件打交道, 并且一旦完成任务立即结束使用
- 在进行密集型任务前, 检查电池电量和充电状态
- 在电量低时, 提示用户是否确定要执行任务,并在用户同意后再执行
- 或提供设置的选项,允许用户定义电量的阈值,以便在执行秘籍型操作前提示用户

下边代码展示了设置电量的阈值以提示用户.

```
- (IBAction)onIntensiveOperationButtonClick:(id)sender {

    NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];

    BOOL prompt = [defaults boolForKey:@"promptForBattery"];
    int minLevel = [defaults integerForKey:@"minBatteryLevel"];

    BOOL canAutoProceed = [self shouldProceeWithMinLevel:minLevel];

    if (canAutoProceed) {
        [self executeIntensiveOperation];
    }else{

        if (prompt) {
            UIAlertView *view = [[UIAlertView alloc]initWithTitle:@"提
示" message:@"电量低于最小值,是否继续执行" delegate: self cancelButtonTitle:@
"取消" otherButtonTitles:@"确定"];
            [view show];
        }else{

            [self queueIntensiveOperation];
        }
    }
}

- (void)alertView:(UIAlertView *)alertView clickedButtonAtIndex:(NSInteger)buttonIndex{

    if (buttonIndex == 0) {
        [self queueIntensiveOperation];
    }else{
        [self executeIntensiveOperation];
    }
}
```

代码对应的配图如下



- 设置由两个条目组成:`promptForBattery`(应用设置中的拨动开关,表明是否要在低电量时给予提示)和`miniBatteryLevel`(区间为0~100的一个滑块,表明了最低电量------在此示例中,用户可以自行调整),在实际项目中应用的开发人员通常根据操作的复杂性和密集性对阈值进行预设.不同的密集型操作可能会有不同的最低电量需求
- 在实际执行密集操作之前,检查当前电量是否足够, 或者手机是否正在充电.这就是我们判断是否可以进行后续处理的逻辑,图中你可以有自己的定制---最低电量和充电状态

> 用户总是随身携带者手机,所以编写省电的代码就格外重要, 毕竟手机的移动电源并不是随处可见,
>
> 不过现在北京的街电共享充电宝好像很不错 本人逛街会经常使用街电充电宝,但还是要尽可能的为
>
> 用户省电 在无法降低任务复杂性时, 提供一个对电池电量保持敏感的方案并在适当的时机提示用
>
> 户, 会让用户感觉很良好, 并且因此会成为你 APP 的永久用户