```
加载资源,对象创建,对象调整,对象销毁,布局计算,Autolayout,文本计算,文
                        本渲染,图片的解码, 图像的绘制(Core Graphics)都是在CPU上面进行的
                 CPU
                        性能优化的一个重要基础就是GPU 和 CPU 的任务协助均衡
                        GPU是一个专门为图形高并发计算而量身定做的处理单元
                 GPU
                        GPU的浮点计算能力,渲染性能要比CPU高效很多
                         尽量让CPU负责主线程的UI调动,把图形显示相关的工作交给GPU来处理
                        CPU和GPU中任何一个压力过大,都会导致掉帧现象
                                通常情况下,屏幕会保持60hz/s的刷新速度,每次刷新时会发出一个屏幕
                                刷新信号,CADisplayLink允许我们注册一个与刷新信号同步的回调处理。
                                可以通过屏幕刷新机制来展示fps值:
                                        – (void)startFpsMonitoring {
                                          _link = [CADisplayLink displayLinkWithTarget: self selector: @selector(displayFps:)];
                                          [ link addToRunLoop: [NSRunLoop mainRunLoop] forMode: NSRunLoopCommonModes];
                                        – (void)displayFps: (CADisplayLink *)fpsDisplay {
                                          self.count++;
                                代码:
                                          CFAbsoluteTime threshold = CFAbsoluteTimeGetCurrent() - _lastTime;
                 卡顿
                                          if (threshold >= 1.0) {
                        FPS
                                            _fps = (_count / threshold);
                                            _lastTime = CFAbsoluteTimeGetCurrent();
                                            _fpsLbe.text = [NSString stringWithFormat:@"FPS:%.0f",_fps];
                                            self.count = 0;
                                            NSLog(@"count = %d,_lastTime = %f, _fps = %.0f",_count, _lastTime, _fps);
                                (1).通过打印,我们知道每次刷新时会发出一个屏幕刷新信号,则与刷新信号同步的回调方法`fpsDisplayLinkAction:`
                                会调用,然后count加一。
                                (2).每隔一秒,我们计算一次 `fps `值,用一个变量_lastTime记录上一次计算 fps 值的时间,然后将 count 的值除以
                                时间间隔,就得到了 fps 的值,在将_lastTime重新赋值,_count置成零。
                                (3).正常情况下,屏幕会保持60hz/s的刷新速度,所以1秒内`fpsDisplayLinkAction:`方法会调用60次。fps 计算的值为
                                0,就不卡顿,流畅。
                                (4).如果1秒内`fpsDisplayLinkAction:`只回调了50次,计算出来的fps就是 _count / delta(时间间隔)。
                            TableViewCell 复用
                 TableView
                            TableViewCell高度缓存(当 cell 高度需要自适应时下次最好缓存高度)
                                       减少多余的绘制操作
                          减少视图层级
                                       减少subviews个数,用layer绘
                 视图层
                                       制元素
                          少用 clearColor, maskToBounds, 阴影效果
                                     图片绝对是使用最频繁的资源之一,我们知道磁盘和网络的
                                     加载速度和内存比要慢很多,而一般图片都比较大, I/O十分
                        图片重要性
                                     耗时。而且图片还涉及解码,也是一项十分消耗CPU的工
                                     作,因此图片的优化对app的性能有着十分关键的作用
                                加载image涉及I/O,解码缓存等等,在使用SDWeblmage、
                        实现
                                FastImageCache、YYImage等框架时可以了解下其内部实现原理
                                重要的地方使用png图片,苹果
                                对png支持更优于jpg,有些地方
                        png
                                必须是png图
                                有些时候涉及大量帧图,或者文
                                件较大时需要压缩,不需要透明
                        jpg
                                度效果,等等可以适当考虑jpg
                 图片
                        子线程预解码(Decode),主线程直接渲染。因为当image没有
                        Decode, 直接赋值给imageView会进行一个Decode操作
                        优化图片大小,尽量不要动态缩放(contentMode)
                        切图注意: UllmageView的image含有alpha channel(即使
                        UllmageView的alpha是1,但只要image含有透明通道,则仍会导致
                        blending)
                        小图片,重复使用可使用imageNamed:这个方法会缓存图片
                        超大图片不要用imageNamed:(占用过多内存), 通过
                        imageWithContentsOfFile取
                                            UllmageView的Frame大小跟加载的图片大
                                            小不一样, 显示的时候就会去缩放图片,在主
                        显示图片前先 缩放图片
                                            线程操作必定影响,可以在后台子线程操作
                            透明view会引起blending(叠加混合像素颜色的计算)
                            UllmageView的image减少含有alpha channel
                 透明 view
                            opaque设置为YES,减少性能消耗,因为GPU将不会做任何合
                            成,而是简单从这个层拷贝
                                 Dictionaries: 存储键值对。 用
                 使用NSDictionary
                                 键来查找比较快。
                 NSSets
                          无序的一组值。用值来查找很快,插入/删除很快
                                          CALayer 的 border、圆角、阴影、遮罩(mask),CASharpLayer 的
                                          矢量图形显示,通常会触发离屏渲染(offscreen rendering)
                           产生来源/原因
                                          当一个列表视图中出现大量圆角的 CALayer, 并且快速滑动时, 可以观
                                          察到 GPU 资源已经占满,而 CPU 资源消耗很少。这时界面仍然能正常
                                          滑动,但平均帧数会降到很低
                           离屏渲染指的是在图像在绘制到当前屏幕前,需要先进行一次离屏渲染,之后才绘制到当前屏幕
                                                当前屏幕渲染On-Screen
                                                                     GPU的渲染操作是在当前用于显示的屏幕缓冲区中
                                                                     进行, 当即完成即可显示
                                                Rendering
                           OpenGL中GPU屏幕渲染
                                                                  GPU在当前屏幕缓冲区以外新开辟一个缓冲区进行渲染
                                                离屏渲染Off-Screen
                                                                   操作, 渲染完成后需要切换上下文, 回到当前屏幕显示
                                                Rendering
                                            1 先从当前屏幕(On-Screen)
                                                                     2 离屏创建新的缓冲区
                                                                                        3 等待离屏渲染结束以后
                                            切换到离屏(Off-Screen)
                                            4 将上下文环境从离屏切换到
                                                                   5 将离屏缓冲区的渲染结果显
                           离屏渲染的个过程
                                            当前屏幕
                                                                   示到屏幕上
                                            损耗:来回切换上下文,创建新的缓冲区等等都
                                            是损耗.付出很大代价
                                            layer.shouldRasterize, 光栅化
                                            layer.mask, 遮罩
                                            layer.allowsGroupOpacity为YES
                                            layer.opacity的值小于1.0
                           触发离屏渲染操作
                                            layer.cornerRadius,并且设置
                                            layer.masksToBounds为YES 来切圆角
                                            layer.shadows,(或表示相关的shadow
                离屏渲染
                                            开头的属性),(使用shadowPath代替)
                                          使用ShadowPath指定layer阴影效果路径
                                          使用异步进行layer渲染(Facebook开源的异步绘制框架AsyncDisplayKit)
                                          设置layer的opaque值为YES,减少复杂图层合成
                                          尽量使用不包含透明(alpha)通道的图片资源
                                          尽量设置layer的大小值为整形值
                                           直接让美工把图片切成圆角进行显示,或者圆角图覆盖,这是效率最高的一种方案
                                          使用代码手动生成圆角image设置到要显示的View上,利用
                                          UIBezierPath (Core Graphics框架) 画出来圆角图片
                                                          光栅化是把GPU的操作转到CPU上,生成位图缓存,直接
                                                          读取复用
                           离屏渲染的优化
                                                          优点: CALayer会被光栅化为bitmap, shadows、
                                                          cornerRadius等效果会被缓存
                                          合理使用光栅化
                                                             :更新已经光栅化的layer,会造成离屏渲染,bitmap
                                          shouldRasterize
                                                          超过100ms没有使用就会移除,缓存的大小受限制
                                                          光栅化shouldRasterize 适合静态页面显示,动态页面如
                                                              。如果设置了shouldRasterize为 YES,那也要记住
                                                          设置rasterizationScale为contentsScale
                                                     dispatch_async(dispatch_get_global_queue(0,0), ^{
                                                       //执行渲染代码 ...
                                                       dispatch_async(dispatch_get_main_queue(), ^{
                                          异步渲染
                                                         //回到主线程,执行UI刷新操作
                                                       });
                           参考:
                                   iOS 关于 热启动和冷启动,以及
                                                             https://www.jianshu.com/p/
                                   热启动和冷启动的时间的优化
                                                             82c566a19075
                                          (1):解析Info.plist
                                                         1加载相关信息,例如如闪屏
                                                         2 沙箱建立、权限检查
                                                          1 如果是胖二进制文件,寻找合适当前CPU类别的部分
                                                         2 加载所有依赖的Mach-O文件(递归调用Mach-O加载的方法)
                                          (2):Mach-O加载
                                                         3 定位内部、外部指针引用,例如字符串、函数等
                           App启动过程:
                                                         4 执行声明为__attribute__((constructor))的C函数
                                                         5 加载类扩展(Category)中的方法
                                                         6 C++静态对象加载、调用ObjC的 +load 函数
                                                         1调用main()
                                          (3)程序执行
                                                         2 调用UIApplicationMain()
                                                         3 调用applicationWillFinishLaunching
                                                 App不在后台运行时被启动(如 重启
                                                                            启动需要重新加载很多配置,文
                                       冷启动:
                                                 过,手动关掉进程,系统杀死进程)
                                                                            件等等
                           冷/热启动
                                       热启动:
                                                 App还在后台运行时被启动
                                                                       启动基本不需要做什么
                           冷启动:
                                     冷启动涉及 上面介绍的 "App启动过程" 所以步骤, 所以也是性能优化的一个重点对象
                                                      在Xcode的菜单中选择Project→Scheme→Edit Scheme...,然后找到
                                                      Run→ Environment Variables →+, 添加name为
                                              设置:
                                                      DYLD_PRINT_STATISTICSvalue为1的环境变量
                                                      设置完成后,在Xcode运行App时,会在console中得到一个报告
iOS性能优化
                                              耗时控制在300-400ms以内
                           main()函数之前 耗时:
                                                         * 动态库加载越多,启动越慢。(>50)
                                                         * ObjC类越多,启动越慢 (> 1000)
                                              耗时因素:
                                                         * C的constructor函数越多,启动越慢
                                                         * C++静态对象越多,启动越慢
                                                         * ObjC的+load越多,启动越慢
                                              耗时控制:
                                                          中小项目500ms以内
                                                                            大项目1s以内
                 APP启动
                                              *执行main()函数的耗时
                           main()函数之后 耗时:
                                              * 执行applicationWillFinishLaunching的耗时
                                              * rootViewController及其childViewController的加载、view及其subviews的加载
                                               applicationWillFinishLaunching的耗时
                                         1. main()函数之前的加载
                                                              (动态库,类等等加载)
                                         2. main()函数之后至applicationWillFinishLaunching完成 之间的加载
                                                                                           (rootVC,推送,bugly等等注册)
                           APP展示阶段:
                                         3. App完成所有本地数据的加载并将相应的信息展示给用户
                                                                                    (缓存, cell预加载,或者部分界面展示,加载动效等等)
                                          . App完成所有联网数据的加载并将相应的信息展示给用户
                                                                                    (加载成功后刷新界面数据)
                                    目标:
                                            针对前面涉及的步骤进行优化,减少不必要的加载,或延迟加载等等
                                    移除不需要用到的动态库
                                                     有些产品随意变更,开发不会真正去删除暂时不会被使用的类和方
                                                     法, 久而久之,造成冗余. [不可信的产品经理]
                                    移除不需要用到的类
                                                     搜索未使用类工具:
                                                                       fui(Find Unused Imports)的开源项目
                                    整理合并功能类似的类和扩展(Category)
                                                 删除未使用图标,缩小切图大小,减少使用透明通道等等
                           优化:
                                    压缩资源图片
                                                                                  2 https://github.com/
                                                          l、https://github.com/
                                                 工具:
                                                          tinymind/LSUnusedResources
                                                                                  jeffhodnett/Unused.git
                                    一些耗时加载,不影响启动的情况下异步处理
                                                                         NSUserDefaults实际上是在Library文
                                   NSUserDefaults不要存储大数据,存储量不能太大
                                                                         件夹下会生产一个plist文件,加载的时候
                                                                         是整个plist配置文件全部load到内存中
                                   有些不紧急的处理不要放在ViewDidLoad:里
                                   有些只需加载一次的代码使用dispatch_once(),不要重复处理
                                   首页启动请求的并发量不能太大,可以分批,延迟请求
                                          首页可考虑展示启动广告
                           用户体验优化:
                                          预加载无数据填充样式
                                          缓存上次已加载数据
                                                       timer = [NSTimer
                           timer单次(repeats:NO)时会自动结束,
                                                       scheduledTimerWithTimeInterval:1.0
                           但是循环(repeats:YES)要注意循环引
                                                       target:self selector:@selector(action:)
                           用问题:
                                                       userInfo:nil repeats:YES];
                                         将计时器的reperts设置为YES时, timer的代理会强引用
                                        self, 而self本身又引用timer, 所以会导致内存泄漏,
                                         dealloc方法不会执行,所以释放timer的代码不能放在
                           内存泄漏问题
                                         dealloc方法里面,而应该提前执行,比如
                                        viewDidDisappear:(BOOL)animated
                                      //页面消失,进入后台不显示该页面,关闭定时器
                 NSTimer
                                      -(void)viewDidDisappear:(BOOL)animated
                                       //关闭并销毁定时器
                                       if (_timer) {
                                         [_timer setFireDate:[NSDate distantFuture]];
                           timer销毁
                                         [_timer invalidate];
                                         _timer = nil;
                                         DLog(@"timer del");
                                      - (void)dealloc {
                                       DLog(@"timer dealloc");
                              刷新一个cell能解决不要刷新整个tableView
                              利用runloop提高滑动流畅性,在滑动停止的时候再加载内容,像那种一闪而过的(快
                              速滑动),就没有必要加载,可以使用默认的占位符填充内容
                 tableView加载
                              预加载cell高度
                              预加载 即将显示的cell中图片
                              缓存图片
                                不要把耗时操作直接在主线程
                 不要阻塞主线程
                                运行, 因为UI显示必须在主线程,
                               一旦被占用,页面基本卡死
                           从远端下载XML, JSON, HTML或者其它格式数据,
                           而数据比较多时
                 gzip压缩
                           减小文档的一个方式就是在服务端和你的app中打开gzip。这对于
                           文字这种能有更高压缩率的数据来说会有更显著的效用
                           (AFNetworking 也支持)
                        懒加载/重用
                                 缓存不大可能改变但是需要经常读取的东西:cell高度,请求
                        Cache
                                 头,常用又不怎么改变的图片,计算结果等等
                                   当你收到didReceiveMemoryWarning时需要去释
                        内存警告
                                   放不用的缓存
                                         尽量使用原生js
                        减少使用Web特性
                                         Sprite sheet可以让渲染速度加快,甚至比
                        使用Sprite Sheets
                                         标准的屏幕渲染方法节省内存
                        图片解码操作
                                                           必要时 用TextKit或最底层的
                                   UILabel和UITextview都是在主
                                                           CoreText对文本异步绘制
                        文本渲染
                                   线程渲染的, 当显示大量文本
                                                           [NSAttributedString boundingRectWithSize:options:context:]
                                   时, CPU的压力会非常大
                                                           可以在子线程计算
                                                     在团队合作时,个人觉得代码布
                                                     局更方便
                                                     可视化较差,这个在Swift中等到补充
                                              代码
                                                     封装: 封装让代码开发更快,比如使用链式表达,
                                                     RAC等等不比xib和sb慢,尤其是一些已经封装好
                                                      了的功能,要更快
                                                     扩展和继承等更方便
                        代码 or xib 和 storyboard
                                                     Xib在加载时会一起缓存,如何文件包含图片等
                                                     静态资源,内存会是一个考虑因素
                                              xib
                                                     xib尽量拆分,单个文件不宜过大
                                                          相对代码更适用于一些简单的页面绘制
                                                          团队合作要协作好,容易冲突
                                              storyboard
                                                          对电脑性能要求高一点
                                                 在滑动列表(UITableView和
                                                                          随着视图数量的快速增长,
                                                 UICollectionView) 中建议使用
                                                                          Autolayout带来的 CPU 消耗会
                                                 Autolayout 和 Frame 混合布
                                                                          快速上升
                                                 局。
                                                 Cell设计时,将能计算高度的列
                        混合布局(frame VS autolayout)
                                                 表块和需要Autolayout的文本
                                                  信息拆分开
                                                 列表中将设计中相同功能的模
                                                 块单独创建cell, 重用
                                               在复杂多层界面中,不需要处理触摸事件的场景可以
                                               考虑使用CAlayer
                        UIView (UIView VS CAlayer)
                                               UIView层级太多,会导致创建、布局等较耗时
                                                                           尽量使用
                                         新手尽量不要使用animationImages显示帧动
                                                                           imageWithContentsOfFile
                                         画,操作不当内存会飙涨
                                                                           .animationImages = nil 用完清理
                                         替代: 较小的gif替代
                                         替代: 可通过定时器来设置图
                                          片简单达到类似效果
                                                                 CAKeyframeAnimation *animationSequence = [CAKeyframeAnimation
                                                                 animationWithKeyPath: @"contents"];
                                                                 animationSequence.calculationMode = kCAAnimationLinear;
                        动画UllmageView
                                                                 animationSequence.autoreverses = YES;
                        animationImages
                                                                 animationSequence.duration = 0.3;
                                                                 animationSequence.repeatCount = 6;
                                         使用 CAKeyframeAnimation
                                                                 NSMutableArray *animationSequenceArray = [[NSMutableArray alloc] init];
                                                                 for (Ullmage *image in self.animationImages)
                                                                   [animationSequenceArray addObject:(id)image.CGImage];
                                                                 animationSequence.values = animationSequenceArray;
                                                                 [self.layer addAnimation:animationSequence forKey:@"contents"];
                        CAAnimation 动画
                                                     大多数场景下,都可以使用CAShapeLayer替代drawRect
                                                     使用GPU硬件加速,更快
                                       CAShapeLayer
                                                     矢量图绘制,不会出现像素化
                                                     占用内存更少,不会创建寄宿图
                               优化
                                                  drawRect使用CPU绘图,相比之下会很慢,而且十分耗CPU,
                                                  尤其是一些动态页面绘制
                                       drawRect
                                                  drawRect图层每次重绘的时候都需要重新抹掉内存然后重新分
                                                  配,十分占用内存
```