性能优化

概括

重用

view设置为完全不透明

只要一个视图的不透明度小于1,就会导致blending.blending操作在iOS的图形处理器（GPU）中完成的,blending主要指的是混合像素颜色的计算。举个例子,我们把两个图层叠加在一起,如果第一个图层的有透明效果,则最终像素的颜色计算需要将第二个图层也考虑进来。这一过程即为Blending。

> R = S + D * (1 – Sa)
> 其中，R表示混合结果的颜色，S是源颜色，D是目标颜色，Sa是源颜色的alpha值，即透明度。公式中所有的S和D颜色都假定已经预先乘以了他们的透明度。
> 知道图层混合的基本原理以后，再回到正题说说opaque属性的作用。当UIView的opaque属性被设为YES以后，按照上面的公式，也就是Sa的值为1，这个时候公式就变成了：R = S
> 当opaque属性被设为YES时，GPU就不会再利用图层颜色合成公式去合成真正的色值。因此，如果opaque被设置成YES，而对应UIView的alpha属性不为1.0的时候，就会有不可预料的情况发生

避免过于庞大的XIB

当你加载一个XIB的时候所有内容都被放在了内存里，包括任何图片。如果有一个不会即刻用到的view，你这就是在浪费宝贵的内存资源了。

不要阻塞主线程

调整在Image Views中图片大小

延迟加载(lazy load) Views

更多的view意味着更多的渲染，也就是更多的CPU和内存消耗，对于那种嵌套了很多view在UIScrollView里边的app更是如此。

重用大开销对象

比如NSDateFormatter和NSCalendar

优化Table View（稍后细讲）

图片处理（稍后细讲）

避免日期格式转换

如果你可以控制你所处理的日期格式，尽量选择Unix时间戳。

dealloc异步处理

方法时间消耗大致范围

[NSObject alloc] 143ms
[NSObject dealloc] 132ms
[CollectionCell prepareForReuse] 9ms
[cell.contentView systemLayoutSizeFittingSize:UILayoutFittingCompressedSize] 28ms
dispatch_async 4ms
[UIView layoutSublayersOfLayer:] 524ms

优化Table View

1. 在适合的时候使用重用（以App首页为例，早期使cell中添加了collectionView保证重用，反而影响了性能）
2. view尽可能使用不透明色。
3. 高度计算(fdTemplate与定高)(NTYTableViewProxy: - (CGFloat)tableView:(UITableView*)tableView heightForRowAtIndexPath:(NSIndexPath*)indexPath)
4. 异步线程
5. 图片处理
6. 步局
    i. View尽量少
    ii. 不要给cell动态添加subView:在初始化cell的时候就将所有需要展示的添加完毕，然后根据需要来设置hide属性显示和隐藏。
7. 避免离屏渲染，导致离屏渲染情况如下：
    i. 为图层设置遮罩（layer.mask）
    ii. 将图层的layer.masksToBounds / view.clipsToBounds属性设置为true
    iii. 将图层layer.allowsGroupOpacity属性设置为YES和layer.opacity小于1.0
    iv. 为图层设置阴影（layer.shadow *）。
    v. 为图层设置layer.shouldRasterize=true
    vi. 具有layer.cornerRadius，layer.edgeAntialiasingMask，layer.allowsEdgeAntialiasing的图层
    vii. 文本（任何种类，包括UILabel，CATextLayer，Core Text等）。
    viii. 使用CGContext在drawRect :方法中绘制大部分情况下会导致离屏渲染，甚至仅仅是一

个空的实现

> 离屏渲染意思是iOS要显示一个视图时，需要先在后台用CPU计算出视图的Bitmap，再交给GPU做Onscreen-Rendering显示在屏幕上，因为显示一个视图需要两次计算，所以这种Offscreen-Rendering会导致app的图形性能下降。

```
    // 在下一次runloop时执行nty_setImageWithURL，保证图片大小正确
    [Dispatch async:^{
     [self.myImageView nty_setImageWithURL:viewModel.poster placeholder:nil]
;
    }];
```

图片加载优化

核心思想

1. 所有可以不在主线程执行的逻辑都在多线程中执行
2. 将使用过的图片保证下次可以快速调用(用空间换时间)
3. 可以进行懒加载

细节优化

1. 将网络加载到的图片存储在Cache文件夹，以供下次调用
2. 将调用过的图片存储到NSCache，以供下次调用
3. 将图片截切成需要的模式、大小，再进行存储、调用。（避免多次缩放）
4. 不要无意义的切换多线程
5. 截切的图片的宽高保证是4的倍数

```
// 加载图片的全过程
- (void)nty_loadImageWithURL:(id)url
                 placeholder:(UIImage*)placeholder
                      expect:(CGSize)size
                        mode:(UIViewContentMode)contentMode
                    complete:(void (^)(UIImage*image))complete

// 异步进行裁切
UIImage *resizeImage = [self scaleAndSaveImage:image expect:expectSize mode:
contentMode key:key];

// 异步加载图片
- (void)smoothly_setImage:(UIImage*)image

// 裁切图片的内部实现
```

```
- (UIImage*)imageByExpect:(CGSize)size
                     mode:(UIViewContentMode)contentMode
             clipsToBounds:(BOOL)clips
```

性能测试工具

Allocations, Leaks, Time, Core Animation

> Separate by Thread：按线程分开做分析，这样更容易揪出那些吃资源的问题线程。特别是对于主线程，它要处理和渲染所有的接口数据，一旦受到阻塞，程序必然卡顿或停止响应。
> Invert Call Tree：反向输出调用树。把调用层级最深的方法显示在最上面，更容易找到最耗时的操作。
> Hide System Libraries：隐藏系统库文件。过滤掉各种系统调用，只显示自己的代码调用。
> Flattern Recursion：拼合递归。将同一递归函数产生的多条堆栈（因为递归函数会调用自己）合并为一条。
> Top Functions：找到最耗时的函数或方法

App配置文件中，Debug Information Format 设置为DWARF with dSYM File

模拟器测试

Color Blended Layers

用于检测哪里发生了图层混合，将透明色用红色标记出来。
如果label文字有中文，依然会出现图层混合，这是因为此时label多了一个sublayer，sublayer在label背景四周多了一圈透明边。layer的masksToBounds为YES时，图层将会沿着Bounds进行裁剪。

Color Copied Images

比如应用中有一些从网络下载的图片，而GPU恰好不支持这个格式，这就需要CPU预先进行格式转化。如果有这种图片则会将图片标记为蓝色。

Color Misaligned Images

它表示如果图片需要缩放则标记为黄色，如果没有像素对齐则标记为紫色。

Color Offscreen-Rendered Yellow

会把需要离屏渲染的地方标记为黄色，大部分情况下我们需要尽可能避免黄色的出现。

iOS 内存监控组件

项目中的DebugSettingModule、DQDebugUIComponentManager
FBMemoryProfiler
OOMDectector

参考资料

> UITableView优化：http://blog.csdn.net/hmh007/article/details/54907560
> 离屏渲染优化：http://www.jianshu.com/p/ca51c9d3575b
> UIKit性能调优实战讲解：http://www.cocoachina.com/ios/20160208/15238.html?utm_source=tuicool&utm_medium=referral