iOS 性能优化的探索

起因

我们公司的主 App 在大约 17 年 5 月份前后经历了一次大版本迭代，迭代之后更换了若干个一级和二级页面，首页就在这些个一级页面之内。
17 年大约 11 月份的时候，我们的小程序第一个版本正式上线，然后我们技术的大 Leader 拿来了小程序给我们看看，小程序的首页流畅性确实优于我们客户端，于是我们正式启动了性能优化。

明确优化的目标

优化的第一步，肯定是要明确我们优化具体的 Case，需要达到什么样的流畅度？是 fps 达到 60？还是要内存使用降到一个具体的数字?
讨论之后，我们最终将第一期优化定位为将首页的 fps 优化到 60。

调研过程

虽然说是第一期将目标定位优先优化首页的流畅度，但是本着有第一次就要有第二次的原则，我还是将 App 中的其他流畅度敏感页面也 Review 了一遍代码，算是给自己留一个优化思考的方向。

Review 代码发现一些比较显而易见的问题：

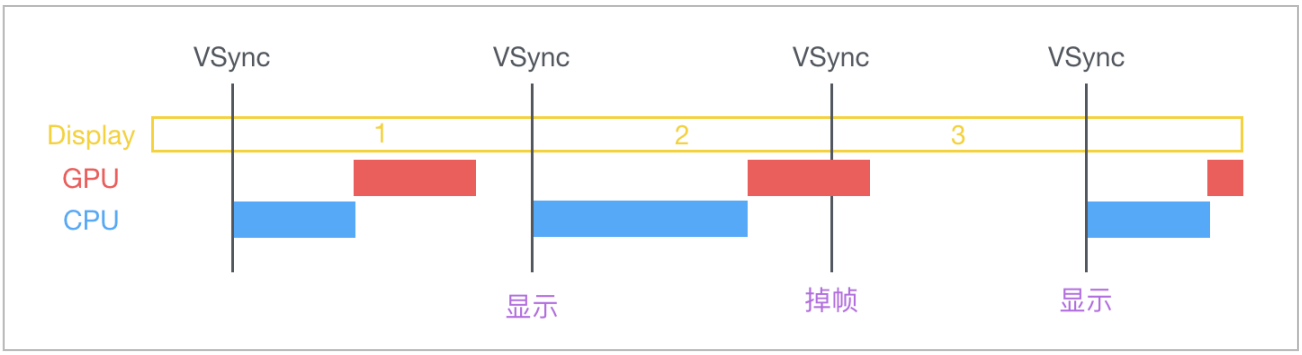
- 肆意调用数据库而没有用 cache
- 复杂 UI 大量使用约束
- 离屏渲染
- 像素混合

PS：因为我们的 IM 系统是我们自己写的，中间又经历了公司分家，人员换了好几茬，于是就导致了在本来架构不合理的基础上，实现业务和功能的代码更是屎一样，所以 IM 的问题更严重。而且我们已经计划了对于 IM 的重构时间表，所以我会在另一篇 Blog 里写一下我的重构思路。

方案整理

影响流畅度的主要原因：

1、文本宽高计算、视图布局计算
2、文本渲染、图片解码、图形绘制
3、对象创建、对象调整、对象销毁



CPU 资源消耗原因以及解决办法：

1、对象的创建：

对象的创建会分配内存、设置属性等，会消耗 CPU 资源。所以尽量使用轻量对象代替，比如能用 CALayer 的时候尽量不用 UIView，敏感位置能不用 IB 尽量使用纯代码手写。

推迟同一时间创建对象，推荐使用懒加载在需要使用时候创建对象。

2、对象调整

对 UIView 的这些属性进行调整时，消耗的资源要远大于一般的属性。对此你在应用中，应该尽量减少不必要的属性修改。

当视图层次调整时，UIView、CALayer 之间会出现很多方法调用与通知，所以在优化性能时，应该尽量避免调整视图层次、添加和移除视图。

3、对象销毁

当前类持有大量对象时候，其销毁时候的资源消耗就非常明显。建议创建销毁的异步队列，将需要销毁的对象放到队列中销毁。

4、布局计算

布局计算在 UITableView 使用中是最常见的消耗资源的地方。建议取到数据之后，异步进行计算布局并缓存下来，当复用 Cell 时候直接调用缓存数据。

5、AutoLayout

Autolayout 对于复杂视图来说常常会产生严重的性能问题，AutoLayout 相对低效的原因是隐藏在底层的命名为”Cassowary“的约束求解系统，随着视图数量的增长，Autolayout 带来的 CPU 消耗会呈指数级上升，当 Cell 内约束超过 25 个的时候，会降低滑动的帧率。

具体：http://pilky.me/36/ (https://link.jianshu.com?t=http%3A%2F%2Fpilky.me%2F36%2F) 。

6、文本的计算以及渲染

UI 中存在大量的对于文本高度的适配，可以参考：用 [NSAttributedString boundingRectWithSize:options:context:] 来计算文本宽高，用 –[NSAttributedString drawWithRect:options:context:] 来绘制文本。尽管这两个方法性能不错，但仍旧需要放到后台线程进行以避免阻塞主线程。

常见的文本控件 （UILabel、UITextView 等），其排版和绘制都是在主线程进行的，当显示大量文本时，CPU 的压力会非常大。解决办法是利用 TextKit 或者是 CoreText 自定义文本控件。详见：YYText (https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fibireme%2FYYText) 。

7、图片解码以及图像的绘制

当你用 UIImage 或 CGImageSource 的那几个方法创建图片时，图片数据并不会立刻解码。图片设置到 UIImageView 或者 CALayer.contents 中去，并且 CALayer 被提交到 GPU 前，CGImage 中的数据才会得到解码。这一步是发生在主线程的，并且不可避免。如果想要绕开这个机制，常见的做法是在后台线程先把图片绘制到 CGBitmapContext 中，然后从 Bitmap 直接创建图片。目前常见的网络图片库都自带这个功能。

个最常见的地方就是 [UIView drawRect:] 里面了。由于 CoreGraphic 方法通常都是线程安全的，所以图像的绘制可以很容易的放到后台线程进行。

8、文件系统的调用

NSFileManager 获取一个目录获取文件信息，进行多次递归计算，stat 几乎瞬间完成，NSFileManager 耗时较长且消耗 CPU。

GPU 资源消耗原因以及解决办法：

1、纹理的渲染

当在较短时间显示大量图片时（比如 TableView 存在非常多的图片并且快速滑动时），CPU 占用率很低，GPU 占用非常高，界面仍然会掉帧。避免这种情况的方法只能是尽量减少在短时间内大量图片的显示，尽可能将多张图片合成为一张进行显示。

2、视图的混合（Blended）

视图结构过于复杂，混合的过程、会消耗很多 GPU 资源。为了减轻这种情况的 GPU 消耗，应用应当尽量减少视图数量和层次，并在不透明的视图里标明 opaque 属性以避免无用的 Alpha 通道合成。当然，这也可以用上面的方法，把多个视图预先渲染为一张图片来显示。

- Blended Layers（视图混合）：在同一个区域内，存在着多个有透明度的图层，那么 GPU 需要更多的计算，混合上下多个图层才能得出最终像素的 RGB 值。
- Misaligned Images（像素对齐）：逻辑像素（point）和 物理像素（pixel）无法相匹配；图片的 size 和显示图片的 imageView 的 size（逻辑像素（point））不相等。

3、图形的生成

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），而离屏渲染通常发生在 GPU 中。可以尝试开启 CALayer.shouldRasterize 属性，但这会把原本离屏渲染的操作转嫁到 CPU 上去。

好的方法是使用图片遮罩等方法，避免使用圆角和隐形等。详细：[iOS 的离屏渲染](https://link.jianshu.com?t=http%3A%2F%2Fwww.imlifengfeng.com%2Fblog%2F%3Fp%3D593)

解决方案：

1、预先计算 UI 布局

获取数据之后，异步计算 Cell 高度以及各控件高度和位置，并储存在 CellLayouModel 中，当每次 Cell 需要高度以及内部布局的时候就可以直接调用，不需要进行重复计算。

2、使用自动缓存高度

iOS 8 之后出现了 UITableView 通过约束自动计算高度，但是因为 iOS 对于约束的算法问题，会导致流畅性降低， FDTemplateLayoutCell (https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fforkingdog%2FUITableView-FDTemplateLayoutCell%2F) 很好的优化了这个问题。

3、异步绘制

Facebook 的开源项目 Texture (https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2FTextureGroup%2FTexture) （原 AsyncDisplayKit），通过利用 ASDisplayNode 封装了 CALayer，实现了异步绘制。

第三方微博客户端 墨客 (https://link.jianshu.com?t=https%3A%2F%2Fitunes.apple.com%2Fcn%2Fapp%2Fmoke-%25E7%25BA%25AF%25E7%25B2%25B9%25E4%25BD%2593%25E9%25AA%258C-for-%25E6%2596%25B0%25E6%25B5%25AA%25E5%25BE%25AE%25E5%258D%259A%2Fid880813963%3Fmt%3D8) 的是现实，当滑动时，松开手指后，立刻计算出滑动停止时 Cell 的位置，并预先绘制那个位置附近的几个 Cell，而忽略当前滑动中的 Cell。但也有缺点，快速滑动的时候有可能会出现大量空白。

3、高效图片加载

- FastImageCache-github (https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fpath%2FFastImageCache)
- SDWebImage-github (https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Frs%2FSDWebImage)
- YYImage-github (https://link.jianshu.com?t=https%3A%2F%2Fgithub.com%2Fibireme%2FYYImage)

4、预加载

列表当中，当滑动到一个可以设定的位置的时候，提前获取下载下一页的数据，并绘制 UI。详见：https://zhuanlan.zhihu.com/p/23418800 (https://link.jianshu.com?t=https%3A%2F%2Fzhuanlan.zhihu.com%2Fp%2F23418800) 。

5、针对 Blended Layers 以及 Misaligned Images

Blended Layers：

- 尽量少的使用具有透明色的图片
- 尽量多的将 UI 部件增加背景色
- 减少同一像素点进行过多的颜色计算

Misaligned Images：

现象：

洋红色：UIView 的 frame 像素不对齐，即不能换算成整数像素值。
黄色：UIImageView 的图片像素大小与其 frame.size 不对齐，图片发生了缩放造成。

解决：

- 尽量使用 ceil()，保证没有小数的 UI 绘制
- 尽量不实用 0.01f 标记 UITableView 或者 UICollectionView 的 header 以及 footer
- 网络上获取的图片没有 @2x 和 @3x 的区别，需要我们缩放图片到与 UIImageView 对应的尺寸，且缩放后的图片的 scale 和 [UIScreen mainScreen].scale 相等，再显示出来。

其他：

下面的情况或操作会引发离屏渲染：

- 为图层设置遮罩（layer.mask）
- 将图层的 layer.masksToBounds / view.clipsToBounds 属性设置为 true
- 将图层 layer.allowsGroupOpacity 属性设置为 YES 和 layer.opacity 小于 1.0
- 为图层设置阴影（layer.shadow *）。
- 为图层设置 layer.shouldRasterize=true
- 具有 layer.cornerRadius，layer.edgeAntialiasingMask，layer.allowsEdgeAntialiasing 的图层
- 文本（任何种类，包括 UILabel，CATextLayer，Core Text 等）。

- 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。

开工

我们综合分析了下现有首页代码的代码结构，发现主要存在问题如下

-
- 较多的使用约束进行布局
- 每次 Cell 滑动进入屏幕都需要根据 DataSource 计算一次 Cell 高度，浪费时间。
- 像素混合等问题严重。
- 存在离屏渲染。
- 无预加载逻辑，导致滑动流畅性降低。

于是我们做了以下措施：

- 使用 frame 布局来替换约束布局。
- 每次拉取到数据之后，异步计算 Cell 高度并做为单独字段缓存在 DataSource 中。
- 按照代码规范，规避像素混合问题和离屏渲染。
- 在相应位置增加了预先拉取数据的逻辑，实现效果是用户没有将 UI 滑动到最后一条的时候，数据以及完成了拉取并刷新 UI 的逻辑。

总结

性能优化这个东西其实很难形成一个具体的方案，为什么这么说？因为之所以称之为优化，是因为要在原有的代码基础上进行优化，原有的代码又有各式各样的原因导致需要依照现有代码来优化，而很难完全脱离现有的情况完全参照某一种的既定方案进行优化。假如说是完全参照某一种的方案优化的话，建议还是将某一个性能敏感的页面利用 Texture 进行完全重写，这样才能算是整体化一的利用了某一种方案。

Refrence

- 如何做优化，UITabelView 才能更加顺滑 (https://link.jianshu.com?t=https%3A%2F%2Fmp.weixin.qq.com%2Fs%3F__biz%3DMjM5OTM0MzIwMQ%3D%3D%26mid%3D402351449%26idx%3D1%26sn%3D3ef1c82a123da76a1e899ab70bb5e9f7%26scene%3

D1%26srcid%3D0118mcBfdWRqJ8w20WOqmbvh%23rd)
(https://link.jianshu.com?
t=https%3A%2F%2Fblog.ibireme.com%2F2015%2F11%2F12%2
Fsmooth_user_interfaces_for_ios%2F)

- iOS 保持界面流畅的技巧 (https://link.jianshu.com?
t=https%3A%2F%2Fblog.ibireme.com%2F2015%2F11%2F12%2
Fsmooth_user_interfaces_for_ios%2F)
- 优化 UITableViewCell 高度计算的那些事
(https://link.jianshu.com?
t=http%3A%2F%2Fblog.sunnyxx.com%2F2015%2F05%2F17%
2Fcell-height-calculation%2F)
- iOS 优化（三）没错我还是滑动优化
(https://www.jianshu.com/p/f3e18bab841e)
- iOS 的离屏渲染 (https://link.jianshu.com?
t=http%3A%2F%2Fwww.imlifengfeng.com%2Fblog%2F%3Fp%
3D593)
- iOS 开发针对对 Masonry 下的 FPS 优化讨论
(https://link.jianshu.com?
t=https%3A%2F%2Fjuejin.im%2Fentry%2F5a111ae86fb9a0452
11e4b62)

另外

- 简书地址 (https://www.jianshu.com/u/ac41d8480d04)
- 掘金地址 (https://link.jianshu.com?
t=https%3A%2F%2Fjuejin.im%2Fuser%2F5730b373f38c8400
67d0d602)

全文完