

UITableView优化技巧

2015-06-10 100000+ 有意思啊



最近在微博上看到一个很好的开源项目VVeboTableViewDemo，是关于如何优化UITableView的。加上正好最近也在优化项目中的类似朋友圈功能这块，思考了很多关于UITableView的优化技巧，相信这块是难点也是痛点，所以决定详细的整理下我对优化UITableView的理解。

UITableView作为iOS开发中最重要的控件之一，其中的实现原理很是考究。Apple在这块的优化水平直接决定了iOS的体验能甩安卓几条街，哈哈，扯淡扯多了。。。好了，废话不多说，直接进入主题。首先来谈谈我对UITableView的认识：

UITableView的简单认识

UITableView最核心的思想就是UITableViewCell的重用机制。简单的理解就是：UITableView只会创建一屏幕（或一屏幕多一点）的UITableViewCell，其他都是从中取出来重用的。每当Cell滑出屏幕时，就会放入到一个集合（或数组）中（这里就相当于一个重用池），当要显示某一位置的Cell时，会先去集合（或数组）中取，如果有，就直接拿来显示；如果没有，才会创建。这样做的好处可想而知，极大的减少了内存的开销。

知道UITableViewCell的重用原理后，我们来看看UITableView的回调方法。UITableView最主要的两个回调方法是tableView:cellForRowAtIndexPath:和tableView:heightForRowAtIndexPath:。理想上我们是会认为UITableView会先调用前者，再

调用后者，因为这和我们创建控件的思路是一样的，先创建它，再设置它的布局。但实际上却并非如此，我们都知道，**UITableView**是继承自**UIScrollView**的，需要先确定它的**contentSize**及每个**Cell**的位置，然后才会把重用的**Cell**放置到对应的位置。所以事实上，**UITableView**的回调顺序是先多次调用**tableView:heightForRowAtIndexPath:**以确定**contentSize**及**Cell**的位置，然后才会调用**tableView:cellForRowAtIndexPath:**，从而来显示在当前屏幕的**Cell**。

举个例子来说：如果现在要显示100个**Cell**，当前屏幕显示5个。那么刷新（reload）**UITableView**时，**UITableView**会先调用100次**tableView:heightForRowAtIndexPath:**方法，然后调用5次**tableView:cellForRowAtIndexPath:**方法；滚动屏幕时，每当**Cell**滚入屏幕，都会调用一次**tableView:heightForRowAtIndexPath:**、**tableView:cellForRowAtIndexPath:**方法。

看到这里，想必大伙也都能隐约察觉到，**UITableView**优化的首要任务是要优化上面两个回调方法。事实也确实如此，下面按照我探讨进阶的过程，来研究如何优化：

优化探索：改进前的代码

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath {
    ContacterTableCell *cell = [tableView
    dequeueReusableCellWithIdentifier:@"ContacterTableCell"];
    if (!cell) {
        cell = (ContacterTableCell *)[[[NSBundle mainBundle]
    loadNibNamed:@"ContacterTableCell" owner:self options:nil] lastObject];
    }
    NSDictionary *dict = self.dataList[indexPath.row];
    [cell setContentInfo:dict];
    return cell;
}

- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath
*)indexPath {
    UITableViewCell *cell = [tableView cellForRowAtIndexPath:indexPath];
    return cell.frame.size.height;
}
```

看到这段代码，对于刚毕业的我来说，觉得还是蛮巧妙的，但巧归巧，当**Cell**非常复杂的时候，直接卡出翔了。。。特别是在我的**Touch4**上，这我能忍？！好吧，依据上面**UITableView**原理的分析，我们先来分析它为什么卡？

这样写，在**Cell**赋值内容的时候，会根据内容设置布局，当然也就可以知道**Cell**的高度，想想如果1000行，那就会调用1000+页面**Cell**个数次**tableView:(UITableView *)tableView**

`cellForRowAtIndexPath:(NSIndexPath *)indexPath`方法，而我们对Cell的处理操作，都是在这个方法里的！什么赋值、布局等等。开销自然很大，这种方案Pass。。。改进代码。

改进后的代码

```
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    NSDictionary *dict = self.dataList[indexPath.row];
    return [ContacterTableCell cellHeightOfInfo:dict];
}
```

思路是：把赋值和计算布局分离。这样让`tableView:cellForRowAtIndexPath:`方法只负责赋值，`tableView:heightForRowAtIndexPath:`方法只负责计算高度。注意：两个方法尽可能的各司其职，不要重叠代码！两者都需要尽可能的简单易算。Run一下，会发现UITableView滚动流畅了很多。

基于上面的实现思路，我们可以在获得数据后，直接先根据数据源计算出对应的布局，并缓存到数据源中，这样在`tableView:heightForRowAtIndexPath:`方法中就直接返回高度，而不需要每次都计算了。

```
- (CGFloat)tableView:(UITableView *)tableView heightForRowAtIndexPath:(NSIndexPath *)indexPath {
    NSDictionary *dict = self.dataList[indexPath.row];
    CGRect rect = [dict[@"frame"] CGRectValue];
    return rect.frame.height;
}
```

其实上面的改进方法并不是最佳方案，但基本能满足简单的界面！记得开头我的任务吗？像朋友圈那样的图文混排，这种方案还是扛不住的！我们需要进入更深层次的探究：自定义Cell的绘制。

我们在Cell上添加系统控件的时候，实质上系统都需要调用底层的接口进行绘制，当我们大量添加控件时，对资源的开销也会很大，所以我们可以索性直接绘制，提高效率。是不是说的很抽象？废话不多说，直接上代码：

首先需要给自定义的Cell添加draw方法，（当然也可以重写`drawRect`）然后在方法体中实现：

```
//异步绘制
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
```

```

CGRect rect = [_data[@"frame"] CGRectValue];
UIGraphicsBeginImageContextWithOptions(rect.size, YES, 0);
CGContextRef context = UIGraphicsGetCurrentContext();
//整个内容的背景
[[UIColor colorWithRed:250/255.0 green:250/255.0 blue:250/255.0 alpha:1] set];
CGContextFillRect(context, rect);
//转发内容的背景
if ([_data valueForKey:@"subData"]) {
    [[UIColor colorWithRed:243/255.0 green:243/255.0 blue:243/255.0 alpha:1] set];
    CGRect subFrame = [_data[@"subData"][@"frame"] CGRectValue];
    CGContextFillRect(context, subFrame);
    [[UIColor colorWithRed:200/255.0 green:200/255.0 blue:200/255.0 alpha:1] set];
    CGContextFillRect(context, CGRectMake(0, subFrame.origin.y, rect.size.width, .5));
}
{
    //名字
    float leftX = SIZE_GAP_LEFT+SIZE_AVATAR+SIZE_GAP_BIG;
    float x = leftX;
    float y = (SIZE_AVATAR-(SIZE_FONT_NAME+SIZE_FONT_SUBTITLE+6))/2-
2+SIZE_GAP_TOP+SIZE_GAP_SMALL-5;
    [_data[@"name"] drawInContext:context withPosition:CGPointMake(x, y)
andFont:FontWithSize(SIZE_FONT_NAME)
andTextColor:[UIColor colorWithRed:106/255.0 green:140/255.0
blue:181/255.0 alpha:1]
andHeight:rect.size.height];
    //时间+设备
    y += SIZE_FONT_NAME+5;
    float fromX = leftX;
    float size = [UIScreen mainScreenWidth]-leftX;
    NSString *from = [NSString stringWithFormat:@"%@@ %@", _data[@"time"],
_data[@"from"]];
    [from drawInContext:context withPosition:CGPointMake(fromX, y)
andFont:FontWithSize(SIZE_FONT_SUBTITLE)
andTextColor:[UIColor colorWithRed:178/255.0 green:178/255.0 blue:178/255.0
alpha:1]
andHeight:rect.size.height andWidth:size];
}
//将绘制的内容以图片的形式返回，并调主线程显示
UIImage *temp = UIGraphicsGetImageFromCurrentImageContext();

```

```

    UIGraphicsEndImageContext();
    dispatch_async(dispatch_get_main_queue(), ^{
        if (flag==drawColorFlag) {
            postBGView.frame = rect;
            postBGView.image = nil;
            postBGView.image = temp;
        }
    })
    //内容如果是图文混排，就添加View，用CoreText绘制
    [self drawText];
}

```

上述代码只贴出来部分功能，但大体的思路都是一样的，各个信息都是根据之前算好的布局进行绘制的。这里是需要异步绘制，但如果在重写drawRect方法就不需要用GCD异步线程了，因为drawRect本来就是异步绘制的。对于图文混排的绘制，可以移步Google，研究下CoreText，这块内容太多了，不便展开。

好了，至此，我们又让UITableView的效率提高了一个等级！但我们的步伐还远远不止这些，下面我们还可以从UIScrollView的角度出发，再次找到突破口。

滑动UITableView时，按需加载对应的内容
直接上代码：

```

//按需加载 - 如果目标行与当前行相差超过指定行数，只在目标滚动范围的前后指定3行加载。
- (void)scrollViewWillEndDragging:(UIScrollView *)scrollView withVelocity:(CGPoint)velocity
targetContentOffset:(inout CGPoint *)targetContentOffset{
    NSIndexPath *ip = [self indexPathForRowAtPoint:CGPointMake(0, targetContentOffset->y)];
    NSIndexPath *cip = [[self indexPathsForVisibleRows] firstObject];
    NSInteger skipCount = 8;
    if (labs(cip.row-ip.row)>skipCount) {
        NSArray *temp = [self indexPathsForRowsInRect:CGRectMake(0, targetContentOffset->y,
self.width, self.height)];
        NSMutableArray *arr = [NSMutableArray arrayWithArray:temp];
        if (velocity.y<0) {
            NSIndexPath *indexPath = [temp lastObject];
            if (indexPath.row+3<datas.count) {
                [arr addObject:[NSIndexPath indexPathForRow:indexPath.row+1 inSection:0]];
                [arr addObject:[NSIndexPath indexPathForRow:indexPath.row+2 inSection:0]];
            }
        }
    }
}

```

```

        [arr addObject:[NSIndexPath indexPathForRow:indexPath.row+3 inSection:0]];
    }
} else {
    NSIndexPath *indexPath = [temp firstObject];
    if (indexPath.row>3) {
        [arr addObject:[NSIndexPath indexPathForRow:indexPath.row-3 inSection:0]];
        [arr addObject:[NSIndexPath indexPathForRow:indexPath.row-2 inSection:0]];
        [arr addObject:[NSIndexPath indexPathForRow:indexPath.row-1 inSection:0]];
    }
}
[needLoadArr addObjectFromArray:arr];
}
}

```

记得在tableView:cellForRowAtIndexPath:方法中加入判断:

```

if (needLoadArr.count>0&&[needLoadArr indexOfObject:indexPath]==NSNotFound) {
    [cell clear];
    return;
}

```

滚动很快时，只加载目标范围内的Cell，这样按需加载，极大的提高流畅度。

写了这么多，也差不多该来个总结了！UITableView的优化主要从三个方面入手：

- 提前计算并缓存好高度（布局），因为heightForRowAtIndexPath:是调用最频繁的方法；
- 异步绘制，遇到复杂界面，遇到性能瓶颈时，可能就是突破口；
- 滑动时按需加载，这个在大量图片展示，网络加载的时候很管用！（SDWebImage已经实现异步加载，配合这条性能杠杠的）。

除了上面最主要的三个方面外，还有很多几乎大伙都很熟知的优化点：


- 正确使用reuseIdentifier来重用Cells
- 尽量使所有的view opaque，包括Cell自身
- 尽量少用或不用透明图层
- 如果Cell内现实的内容来自web，使用异步加载，缓存请求结果
- 减少subviews的数量
- 在heightForRowAtIndexPath:中尽量不使用cellForRowAtIndexPath:，如果你需要用到它，只用一次然后缓存结果
- 尽量少用addSubview给Cell动态添加View，可以初始化时就添加，然后通过hide来控制是否显示

肯定很多人会非常好奇，为什么我都是手动用代码创建Cell的？现在主流不都是Xib、Storyboard什么的嘛？我的回答是：要想提高效率，还是手动写有用！抛开Xib、Storyboard需要系统自动转码，给系统多加了一层负担不谈，自定义Cell的绘制更是无从下手，所以，在我看来，复杂的需要高效的界面，还是手动写代码吧！！！

知识是需要不断学习的，作为刚上路的我，如果有什么理解不到位的，欢迎大伙留言指正，如果你有什么更牛逼的想法，希望一起交流交流。

注明：本篇的分析源码来源于开源项目VVeboTableViewDemo

参考：<https://github.com/johnil/VVeboTableViewDemo>

 IOS开发技巧请关注

有意思啊



阅读原文



微信扫一扫
关注该公众号