用UIKit Dynamics模仿UIScrollView

(https://www.tuicool.com/)

时间 2016-10-21 16:52:48 哈士狍 (/sites/vaqeUjj)

原文

http://philcai.com/2016/03/15/用UIKit-Dynamics模仿UIScrollView/ (http://philcai.com/2016/03/15/用UIKit-Dynamics模仿UIScrollView/?utm_source=tuicool&utm_medium=referral)

主题 UIScrollView (/topics/10200292)

饿了么在上个版本的时候对餐厅页做了很大的改动, 无论是视觉上还是交互上都有很不错的效果. 为了实现这种效果, 我们自己用UIPanGestureRecognizer和UIKit Dynamics模拟了系统的UIScrollView, 包括惯性滚动, 弹性, 橡皮筋(RubberBanding)效果.

在刚接到这个任务的时候, 有过几种想法:

1. 这个效果很像是UITableView加上Header的Parallel效果
2. 可以在一个UIScrollView上面嵌套一个UITableView作为子视图

这些方案都被否决了. 第一种方案, 因为当前页面不仅有两个TableView(食物类别和菜单), 而且要支持左右滚动在"商品", "评价", "详情"三个页面切换. 用TableView的header做视差效果是不太可能做到的. 对第二种方案, 是在 `-[UIScrollViewDelegate scrollViewDidScroll:]` 中再手动修改其中一个ScrollView的 `contentOffset` , 使得当前的两个scrollView的 `contentOffset` 都是正确的, 但是难点是很难去指定手指在屏幕上滑动的时候, 是父view还是子view的UIPanGestureRecognizer手势被响应. 而考虑先禁用其中的一个手势(比如子view的), 先让父View的手势可以驱动父View的 `contentOffset` 改变, 直到父view的 `contentOffset` 到了某个位置再启用子view的手势, 禁用父view的. 这带来一个问题, 在切换手势的enable的时候, 即使手指没有离开屏幕, 但是手势已经禁用, 导致滚动中断, 除非手指离开屏幕后重新触摸才能再次滚动, 这样的效果比较不流畅, 并且其中的逻辑比较复杂, 不太容易处理. 或者子类UIScrollView和UITableView, 在手势代理 `gestureRecognizer:shouldRecognizeSimultaneouslyWithGestureRecognizer` 返回YES, 使得两个ScrollView可以同时滚动, 然后在 `scrollViewDidScroll:` 中还原其中一个ScrollView的contentOffset. 但是这样导致逻辑变得复杂, 因为视图中的手势太多. 把ScrollView添加为另一个ScrollView的子视图并不推荐.

最后, 考虑到这个效果订制程度很高, 于是自己去模仿一下UIScrollView的特性.

首先说明一下视图的结构:

```
<ParentViewController.View>
    | <Container> //UIView
    |     | <SegmentView>
    |     | <ScrollView> //仅左右滑动(pagingEnabled)
    |     |     | <ChildViewController1.View>
    |     |     |     | <CategoryListView>
    |     |     |     | <FoodListView>
    |     |     | <ChildViewController2.View>
    |     |     |     | <RatingListView>
    |     |     | <ChildViewController3.View>
    |     |     |     | <SummaryListView>
```

ParentViewController就是从首页Push进入的ViewController, 在它的View上放置了一个Container(一个普通的UIView), Container的上方是SegmentView,下方是一个左右滑动的ScrollView; 在ScrollView上, 从左往右放置了三个ViewController的View; 所有的tableView视图的bounce都禁用. 由于使用了AutoLayout,

ViewController

```
[Container mas_makeConstraints:^(MASConstraintMaker *make) {
   make.left.right.bottom.equalTo(ParentViewController.View);
   make.top.equalTo(ParentViewController.View).offset(topOffset);
}];
```

Container的left, right, bottom都对应ParentViewController.View的left,right,bottom, 我们只需要修改topOffset对应的约束, 就可以做出如下的效果:

为了让交互的所有细节都可控, 我们要把FoodListView, RatingListView, SummaryListView的scrollEnabled设置为NO, 也就是将UIScrollView自带的panGestureRecognizer禁用, 然后在Container上加上自己的PanGestureRecognizer. 这样之后只要是和上下滚动相关的交互(tableView的滚动和Container的top的约束)都由自己实现的PanGestureRecognizer完成. 这么做有两点优势, 一是当在上下滑动的时候PanGestureRecognizer一定会触发, 并且在滑动的时候, 可以精确的控制当前手势的位移是修改Container的顶部约束还是修改当前页面的tableView的contentOffset; 二是在手势结束的时候, 可以获取最后手势的速度 `-[UIPanGestureRecognizer velocityInView:]` 这方便了之后模拟惯性效果.

在模拟ScrollView的三个特性里面, 最简单的是RubberBanding(橡皮筋效果), 惯性滚动和弹性原理是类似的.

RubberBanding

因为只启用了自定义的pan手势, 在普通情况下, 要修改tableView的contentOffset 或者修改Container的顶部约束, 只需要在 `pan.state == UIGestureRecognizerStateChanged` , 根据 `[pan translationInView: Container].y` 获取垂直方向的手势位移, 修改contentOffset或者约束的变化等于手势位移. 至于RubberBanding, 在垂直方向上有两种可能: Container距离顶部超过某个预设的值, 手势继续向下拖动; 或者tableView的拉到底部之后手势继续向上. 这个时候修改 `contentOffset` 或者顶部约束的变化小于手势位移(比如乘以一个小于1的因数), 就可以模仿出RubberBanding效果.

惯性 & 弹性

这里说的惯性效果不仅包括模仿tableView自身的惯性减速修改 `contentOffset` .

还包括:

- 在手势结束之后, Container根据惯性的效果动态改变它的顶部约束.
- Container按照惯性效果到顶部后(top约束减小, Container向上移动), 惯性效果没有消失, 继续驱动tableView的 `contentOffset` 修改. (速度传递)
- tableView按照惯性减小 `contentOffset.y` 到0后, 惯性效果继续驱动Container修改顶部约束. (速度传递)
  同样, 弹性效果也不只是tableView到达超过底部之后放手回弹, 也包括Container距离顶部超过一定距离之后放手回弹效果, 以及可能因为速度传递后导致的回弹.

先简单的考虑只在手势结束后发生的惯性和弹性, 很幸运的是可以获取手势最后一刻的速度 `[pan velocityInView:Container].y` . 第一反应是使用UIView的springAnimation, 因为它接受传入速度. 但是其他参数比如duration, 其实没有太好的方案去指定, 如果加上速度传递的效果, 它就更无能为力了. 反复滑动系统的ScrollView, 在调用栈发现它是由CADisplayLink驱动的, 发现它的行为和UIKit Dynamics的动画很符合, 而且UIKit Dynamics背后也是CADisplayLink,加上UIDynamicBehavior有个action属性:

When running, the dynamic animator calls the action block on every animation step.

在每一帧动画的时候都会调用下. 这些组合起来, 足够去模拟ScrollView的各种行为了.

一般我们使用UIKit Dynamics的时候, 我们是把各种Behaviour直接添加到UIView上, 然后视图就会在它到作用下动起来. 但在现在的情况下, 并不能够直接对视图添加Behaviour. 由于Behaviour实际是对遵循UIDynamicItem协议的对象做物理动画, 所以可以把contentOffset或者顶部约束的值做一层抽象.

DynamicItem

```
@interface DynamicItem : NSObject<UIDynamicItem>
@property (nonatomic, readwrite) CGPoint center;
@property (nonatomic, readonly) CGRect bounds;
@property (nonatomic, readwrite) CGAffineTransform transform;
@end

@implementation DynamicItem
- (instancetype)init {
  if (self = [super init]) {
        _bounds = CGRectMake(0, 0, 1, 1);
  }
  return self;
}
@end
```

DynamicItem的实例可以看作是一个质点, 在垂直方向上, 它的位置(center)可以用来代表Container的位置(top), 也可以用来代表tableView的 `contentOffset.y` , 它的 `transform` 属性可以不用考虑.

无论是修改Container的位置还是tableView的 `contentOffset` , 在惯性或弹性效果的情况下, 只要在action中将约束的值或者 `contentOffset.y` 设置为DynamicItem的 `center.y` 就可以. UIKit Dynamics自己会在每一帧去修改.

比如惯性效果下修改Container的顶部约束大概是这样的:

ViewController

```
// when pan.state == UIGestureRecognizerStateEnded
NVMDynamicItem *item = [NVMDynamicItem new];
// topOffset表示当前Container距离顶部的距离
item.center = CGPointMake(0, topOffset);
// velocity是在手势结束的时候获取的竖直方向的手势速度
UIDynamicItemBehavior *inertialBehavior = [[UIDynamicItemBehavior alloc] initWithItems:@[ item ]];
  [inertialBehavior addLinearVelocity:CGPointMake(0, velocity) forItem:item];
  // 通过尝试取2.0比较像系统的效果
  inertialBehavior.resistance = 2.0;
  inertialBehavior.action = ^{
    CGFloat itemTop = item.center.y;
    [Container mas_updateConstraints:^(MASConstraintMaker *make) {
      make.top.equalTo(ParentViewController.View).offset(itemTop);
  }];
};
[self.animator addBehavior:inertialBehavior];
```

类似的弹性效果只需使用UIAttachmentBehavior并且设置合适的值, 尝试下来length = 0, damping = 1, frequency = 1.6, 就有不错的回弹效果.

修改contentOffset也是类似, 只不过是将在action中修改约束的部分改为修改contentOffset.

对于速度传递, 完全一样的原理, 唯一的变化就是从获取手势的速度变为获取 `-[UIDynamicItemBehavior linearVelocityForItem]` 的线速度, 然后UIDynamicAnimator移除不需要的动画, 按照上面的例子传入速度再次做惯性动画.

甚至还可以把UIAttachmentBehavior和UIDynamicItemBehavior同时使用, 模仿有初速度的回弹效果.

大致的思路就是这样, 只需要注意什么时候调用 `-[UIDynamicAnimator removeBehavior]` 停止动画(比如手势刚开始的时候), 以及action中注意retain cycle.

有个2014年的 博客 (http://holko.pl/2014/07/06/inertia-bouncing-rubber-banding-uikit-dynamics/) 已经有了类似的例子, 只是交互简单一些, 原理是一样的.

而在运用自己的手势去实现ScrollView之后, 碰到了一些细节问题.

1. 自己加到Container上的手势, 很容易误触发tableView的 `-tableView:didSelectRowAtIndexPath:indexPath` 协议方法, 导致很容易Push到下一个页面, 很影响使用. 解决的原理比较简单, 在合适的时机将当前的tableView.userInteractionEnabled设置为NO, 之后在需要的时候恢复. 正好UIDynamicAnimatorDelegate提供了动画将要开始 `dynamicAnimatorWillResume:` 和暂停(包括移除bahaviour) `dynamicAnimatorDidPause:` 的回调. 就在这两个地方分别设置, 效果还可以接受.
2. 当tableView在UIKit Dynamics的作用下滚动时, 或者是快速上下滑动的时候, 很容易触发左右滑动的ScrollView切换页面. 解决方案比较tricky: 自定义了UIScrollView的子类, 在子类中将gestureRecognizerShouldBegin:重写, 对于panGestureRecognizer的情况, 在它的水平速度和垂直速度的夹角在一定范围内强制返回NO. 这样就大大减小了误触发左右滚动的操作. 但是还是希望有更好的解决方案.
3. 还有一个很常见的问题, 点击状态栏, 正常情况下系统能够将ScrollView滚动到顶部, 而在一个Window中有多个ScrollView的时候, 它是不一定成功的. 正确的解决方案应该是将当前页面需要响应系统statusBar点击的ScrollView的 `scrollsToTop` 设置为YES, 其他都设置为NO, 并且 `scrollsToTop` 为YES的只能有一个, 这种情况下理论上是可以work的. 但是在解决第一个问题的时候, 导致了这种解决方法有时候不成功. 因为发现在一个UIScrollView的 `userInteractionEnabled == NO` 的时候, 状态栏点击返回顶部效果是无效的(比如正在惯性滚动的时候, 状态还是NO, 这个时候点击statusBar); 加上在最左边的页面有两个tableView需要同时滚动到顶部. 只能换个解决方案. 子类化了全局的UIWindow, 重写它的 `-pointInside:withEvent:` , 在statusBar区域被点击的时候发出通知, 监听到后手动设置contentOffset到0.
4. 由于之前很多控件是AutoLayout写的, 比如cell, 因为现在实现的方案会频繁修改约束, 导致滑动很卡(之前在iPhone 6就感受到卡顿了). 之后用手动布局改了一部分cell, 确实流畅了很多.

虽然UIKit Dynamics平时很少用到, 不过在关键时刻也发挥了巨大的作用, 很好奇Apple在实现UIScrollView会不会也用到了它.

Written by 饿了么iOS组 – PhilCai (https://github.com/philcai1993)

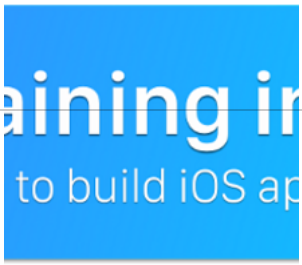
分享

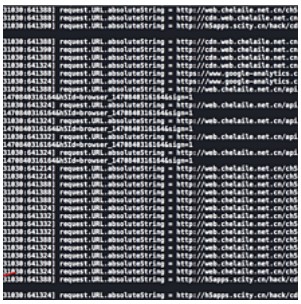
☆ 收藏 ⚠ 纠错

推荐文章

- 1. 学习OpenGL ES之粒子效果 (/articles/mi6rU3M)
- 2. iOS 手写签名的简单实用封装 (/articles/nMFzauY)
- 3. iOS第三方平台集成组件化 (/articles/6fuueei)
- 4. iOS开发小技巧及小知识点 (/articles/rYBRVb7)
- 5. iOS开发之 Xcode 9 和iOS11 适配问题 (/articles/rEV3meu)
- 6. 关于iOS11中estimatedRowHeight (/articles/Y3MJziQ)

相关推刊



- *《匿名收藏》 14*
- by 一直很发烧  (/kans/4261721568) 《技巧》 (/kans/4261721568) *47*

我来评几句

请输入评论内容...

登录后评论

已发表评论数(0)

相关站点

哈士狍 (/sites/vaqeUjj)

+ 订阅

热门文章

1. 学习OpenGL ES之粒子效果 (/articles/mi6rU3M)
2. iOS 手写签名的简单实用封装 (/articles/nMFzauY)
3. iOS第三方平台集成组件化 (/articles/6fuueei)
4. iOS开发小技巧及小知识点 (/articles/rYBRVb7)
5. iOS开发之 Xcode 9 和iOS11 适配问题 (/articles/rEV3meu)