AsyncDisplayKit的使用

一、简介

AsyncDisplayKit 是一个UI框架，最初诞生于 Facebook 的 Paper 应用程序。它是为了解决 Paper 团队面临的核心问题之一：如何尽可能缓解主线程的压力?

ASDK 的作者是 Scott Goodson (Linkedin),
他曾经在苹果工作，负责 iOS 的一些内置应用的开发，比如股票、计算器、地图、钟表、设置、Safari 等，当然他也参与了 UIKit framework 的开发。后来他加入 Facebook 后，负责 Paper 的开发，创建并开源了 AsyncDisplayKit。目前他在 Pinterest 和 Instagram 负责 iOS 开发和用户体验的提升等工作。

1.解决的问题

很多时候用户在操作app的时候，会感觉到不适那么流畅，有所卡顿。
ASDK主要就是解决的问题就是操作页面过程中的保持帧率在60fps（理想状态下）的问题。

造成卡顿的原因有很多，总结一句话基本上就是:
CPU或GPU消耗过大，导致在一次同步信号之间没有准备完成，没有内容提交，导致掉帧的问题。
具体的原理，在提升 iOS 界面的渲染性能文章中介绍的十分详细了，这里也不多阐述了。

2.优化原理

- 布局：
  iOS自带的Autolayout在布局性能上存在瓶颈，并且只能在主线程进行计算。（参考Auto Layout Performance on iOS）因此ASDK弃用了Autolayout，自己参考自家的ComponentKit设计了一套布局方式。
- 渲染
  对于大量文本，图片等的渲染，UIKit组件只能在主线程并且可能会造成GPU绘制的资源紧张。ASDK使用了一些方法，比如图层的预混合等，并且异步的在后台绘制图层，不阻塞主线程的运行。
- 系统对象创建与销毁
  UIKit组件封装了CALayer图层的对象，在创建、调整、销毁的时候，都会在主线程消耗资源。ASDK自己设计了一套Node机制，也能够调用。

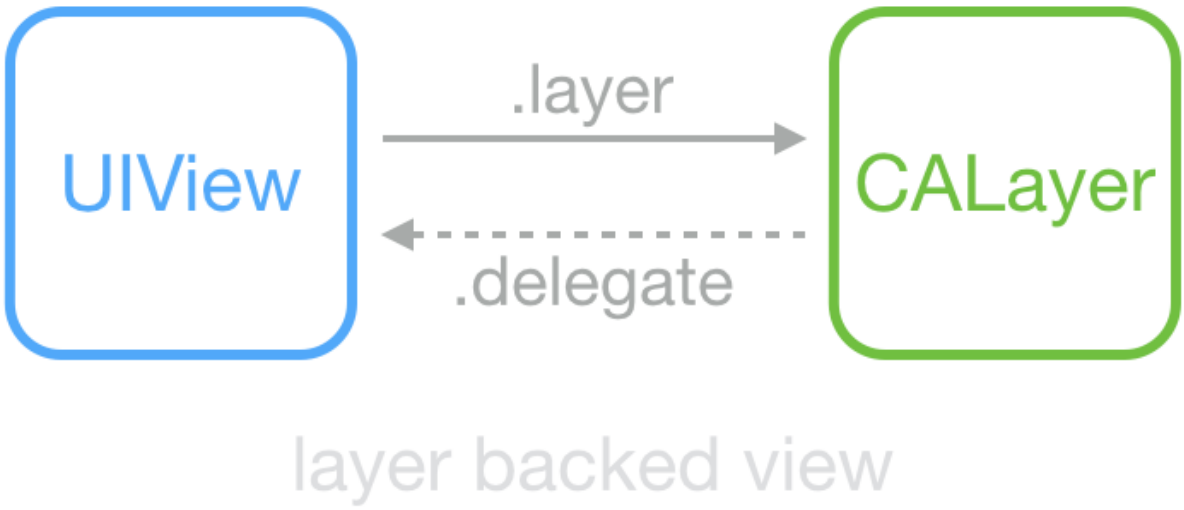
> 实际上，从上面的一些解释也可以看出，ASDK最大的特点就是”异步”。
> 将消耗时间的渲染、图片解码、布局以及其它 UI 操作等等全部移出主线程，这样主线程就可以对用户的操作及时做出反应，来达到流畅运行的目的。

ASDK 认为，阻塞主线程的任务，主要分为上面这三大类。

为了尽量优化性能，ASDK 尝试对 UIKit 组件进行封装：

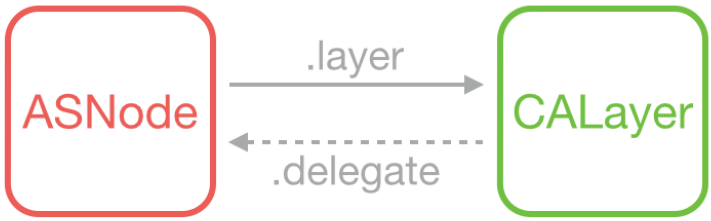
3.Nodes节点

- 如果你之前使用过views，那么你应该已经知道如何使用nodes，大部分的方法都有一个等效的node，大部分的UIView和CALayer的属性都有类似的可用的。任何情况都会有一点点命名差异（例如，clipsToBounds和masksToBounds），node基本上都是默认使用UIView的名字，唯一的例外是node使用position而不是center
- 当然，你也可以直接访问底层view和layer，使用node.view和node.layer
- 关系图



这是常见的 UIView 和 CALayer 的关系：View 持有 Layer 用于显示，View 响应触摸事件。

view backed node



layer backed node

- Node控件

ASDK	UIKit
ASDisplayNode	UIView
ASCellNode	UITableViewCell/UICollectionViewCell
ASTextNode	UILabel
ASImageNode/ASNetworkImageNode	UIImageView
ASVideoNode	AVPlayerLayer
ASControlNode	UIControl
ASScrollNode	UIScrollView
ASEditableTextNode	UITextView
ASMultiplexImageNode（图片管理）	UIImageView

4.安装

CocoaPods安装

```
pod 'AsyncDisplayKit'
```

Carthage安装

AsyncDisplayKit可以使用Carthage安装，将下面的代码添加进入 `Cartfile`

```
github "facebook/AsyncDisplayKit"
```

在终端执行carthage update来构建AsyncDisplayKit库，会自动在项目根目录下生成Carthage名字的文件夹，里面有个build文件夹，可以用来framework到你打算使用的项目中

静态库

AsyncDisplayKit可以当做静态库引入

> - 拷贝整个工程到你的目录下，添加AsyncDisplayKit.xcodeproj到你的workspace
> - 在build phases中，在Target Dependencies下添加AsyncDisplayKit Library
> - 在build phases中，添加libAsyncDisplayKit.a, AssetsLibrary, Photos等框架到Link Binary With Libraries中
> - 在build settings中，添加-lc++和-ObjC到 project linker flags

二、使用

主要介绍常用控件ASTableNode/ASCollectionNode的使用，代码放在GitHub上的ASDK_Demo。

1.ASImageNode

> - 使用ASNetworkImageNode的URL设置网络图片。
> - ASNetworkImageNode有图片下载的ASNetworkImageNodeDelegate
> - ASImageNode使用ASDK的图片管理类**PINCache,PINRemoteImage**
> - 如果不打算引入PINRemoteImage和PINCache，你会失去对jpeg的更好的支持，你需要自行引入你自己的cache系统，需要遵从ASImageCacheProtocol

2.ASTextNode

ASTextNode没有text属性，只能使用attributedText

```
//居中
NSMutableParagraphStyle * paragraphStyle = [[NSMutableParagraphStyle allo
c] init];
paragraphStyle.alignment = NSTextAlignmentCenter;

labelNode.attributedText = [[NSAttributedString alloc] initWithString:@"居
中文字" attributes:@{NSFontAttributeName: [UIFont systemFontOfSize:18],

                              NSForegroundColorAttributeName:[UIColor blac
kColor],

                              NSBackgroundColorAttributeName : [UIColor cl
earColor],

                              NSParagraphStyleAttributeName:paragraphStyl
e}];
```

3.ASTableNode/ASCollectionNode

- ASTableNode/ASCollectionNode不支持复用机制，每次滚动都会重新创建cell。
- ASTableNode并不提供类似UITableview的-tableView:heightForRowAtIndexPath:方法，这是因为节点基于自己的约束来确定自己的高度，就是说你不再需要写代码来确定这个细节，一个node通过-layoutSpecThatFits:方法返回的布局规则确定了行高，所有的节点只要提供了约束大小，就有能力自己确定自己的尺寸
- 使用 Batch Fetching 进行无限滚动，即预加载功能

三、布局

引用1
引用2

ASDK 拥有自己的一套成熟布局方案，虽然学习成本略高，但至少比原生的 AutoLayout 写起来舒服，重点是性能比起 AutoLayout 好的不是一点点。（ASDK不支持autoLayout）

```
//下面这个方法就是用来建立布局规则对象，产生 node 大小以及所有子 node 大小的地方，你
创建的布局规则对象一直持续到这个方法返回的时间点，经过了这个时间点后，它就不可变了。尤其
重要要记住的一点事，千万不要缓存布局规则对象，当你以后需要他的时候，请重新创建。
//调用时机：ASDisplayNode 在初始化之后会检查是否有子视图，如果有就会调用
- (ASLayoutSpec *)layoutSpecThatFits:(ASSizeRange)constrainedSize
```

1. 布局类

- ASAbsoluteLayoutSpec（绝对布局约束)
- ASBackgroundLayoutSpec（背景布局规则）
- ASInsetLayoutSpec（边距布局规则）
- ASOverlayLayoutSpec（覆盖布局规则）
- ASRatioLayoutSpec（比例布局规则）
- ASRelativeLayoutSpec（相对布局规则）
- ASCenterLayoutSpec（中心布局规则）
- ASStackLayoutSpec（堆叠布局规则）
- ASWrapperLayoutSpec （填充布局规则）

2.示例

ASAbsoluteLayoutSpec

```
- (ASLayoutSpec *)layoutSpecThatFits:(ASSizeRange)constrainedSize{
  self.childNode.style.layoutPosition = CGPointMake(100, 100);
  self.childNode.style.preferredLayoutSize = ASLayoutSizeMake(ASDimension
Make(100), ASDimensionMake(100));

  ASAbsoluteLayoutSpec *absoluteLayout = [ASAbsoluteLayoutSpec absoluteLa
youtSpecWithChildren:@[self.childNode]];
  return absoluteLayout;
}
```

> 使用方法和原生的绝对布局类似

ASBackgroundLayoutSpec

```
- (ASLayoutSpec *)layoutSpecThatFits:(ASSizeRange)constrainedSize{
  ASBackgroundLayoutSpec *backgroundLayout = [ASBackgroundLayoutSpec back
groundLayoutSpecWithChild:self.childNodeB background:self.childNodeA];
  return backgroundLayout;
}
```

> 把childNodeA 做为 childNodeB 的背景，也就是 childNodeB 在上层，要注意的是 ASBackgroundLayoutSpec 事实上根本不会改变视图的层级关系

ASInsetLayoutSpec

```
- (ASLayoutSpec *)layoutSpecThatFits:(ASSizeRange)constrainedSize{
    ASInsetLayoutSpec *inset = [ASInsetLayoutSpec insetLayoutSpecWithInse
ts:UIEdgeInsetsZero child:_childNode];
    return insetLayout;
}
```

> _childNode 相对于父视图边距都为 0，相当于填充整个父视图。

ASOverlayLayoutSpec

```
- (ASLayoutSpec *)layoutSpecThatFits:(ASSizeRange)constrainedSize{

  _photoNode.style.preferredSize = CGSizeMake(USER_IMAGE_HEIGHT*2, USER_I
MAGE_HEIGHT*2);

  // INIFINITY(插入无边界)
  UIEdgeInsets insets = UIEdgeInsetsMake(INFINITY, 12, 12, 12);
  ASInsetLayoutSpec *textInsetSpec = [ASInsetLayoutSpec insetLayoutSpecWi
thInsets:insets child:_titleNode];

  return [ASOverlayLayoutSpec overlayLayoutSpecWithChild:_photoNode
                                                 overlay:textInsetSpec];

}
```

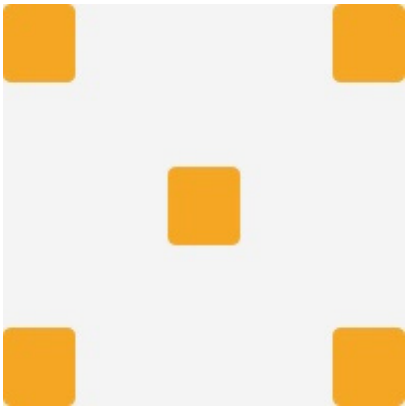
> 类似于ASBackgroundLayoutSpec，都是设置层级关系

ASRatioLayoutSpec

```
- (ASLayoutSpec *)layoutSpecThatFits:(ASSizeRange)constrainedSize{
    ASRatioLayoutSpec *ratioLayout = [ASRatioLayoutSpec ratioLayoutSpecWi
thRatio:1.0f child:self.childNodeA];
    return ratioLayout;
}
```

> 比较常用的一个类，作用是设置自身的高宽比，例如设置正方形的视图

ASRelativeLayoutSpec



```
//把 childNodeA 显示在右上角。
    self.childNodeA.style.preferredSize = CGSizeMake(100, 100);
    ASRelativeLayoutSpec *relativeLayout = [ASRelativeLayoutSpec relative
PositionLayoutSpecWithHorizontalPosition:ASRelativeLayoutSpecPositionEnd
 verticalPosition:ASRelativeLayoutSpecPositionStart sizingOption:ASRelati
veLayoutSpecSizingOptionDefault child:self.childNodeA];
    return relativeLayout;
}
```

> 它可以把视图布局在：左上、左下、右上、右下四个顶点以外，还可以设置成居中布局。

ASCenterLayoutSpec

```
- (ASLayoutSpec *)layoutSpecThatFits:(ASSizeRange)constrainedSize{
    self.childNodeA.style.preferredSize = CGSizeMake(100, 100);
    ASCenterLayoutSpec *relativeLayout = [ASCenterLayoutSpec centerLayout
SpecWithCenteringOptions:ASCenterLayoutSpecCenteringXY sizingOptions:ASCe
nterLayoutSpecSizingOptionDefault child:self.childNodeA];
    return relativeLayout;
}
```

ASWrapperLayoutSpec

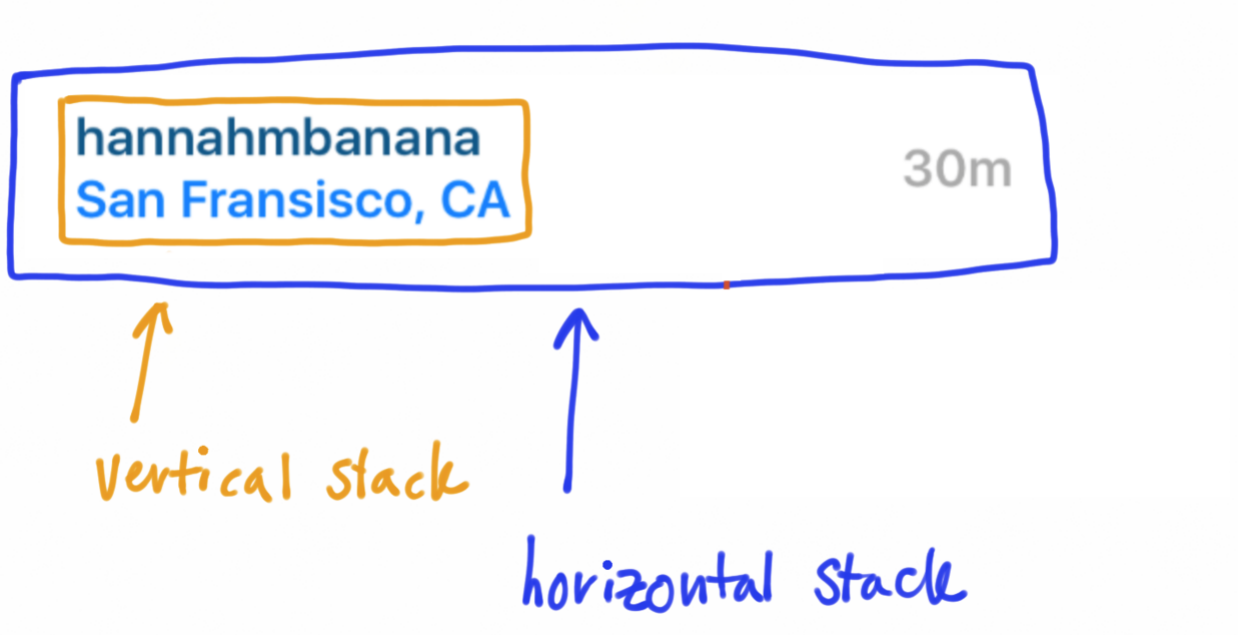
```
- (ASLayoutSpec *)layoutSpecThatFits:(ASSizeRange)constrainedSize{
    ASWrapperLayoutSpec *wrapperLayout = [ASWrapperLayoutSpec wrapperWith
LayoutElement:self.childNodeA];
    return wrapperLayout;
}
```

> 填充整个视图

ASStackLayoutSpec

> 可以说这是最常用的类，而且相对于其他类来说在功能上是最接近于 AutoLayout 的。
> 之所以称之为盒子布局是因为它和 CSS 中 Flexbox 很相似，不清楚 Flexbox 的可以看下这篇博客(HTML布局)。

示例



```
- (ASLayoutSpec *)layoutSpecThatFits:(ASSizeRange)constrainedSize{

  // 当用户名和位置信息文本太长时,收缩堆放视图来适应屏幕,而不是将所有内容向右堆放
  ASStackLayoutSpec *nameLocationStack = [ASStackLayoutSpec verticalStack
LayoutSpec];
  nameLocationStack.style.flexShrink = 1.0;
  nameLocationStack.style.flexGrow = 1.0;

  //如果从服务器获取位置信息,并检查位置信息是否可用
  if (_postLocationNode.attributedText) {
    nameLocationStack.children = @[_usernameNode, _postLocationNode];
  } else {
    nameLocationStack.children = @[_usernameNode];
  }

  //水平堆放
  ASStackLayoutSpec *headerStackSpec = [ASStackLayoutSpec   stackLayoutSp
ecWithDirection:ASStackLayoutDirectionHorizontal
                      spacing:40
               justifyContent:ASStackLayoutJustifyContentStart
                   alignItems:ASStackLayoutAlignItemsCenter
                     children:@[nameLocationStack, _postTimeNode]];

  //插入堆放
  return [ASInsetLayoutSpec insetLayoutSpecWithInsets:UIEdgeInsetsMake(0,
 10, 0, 10)
                                                child:headerStackSpec];

}
```

简单的说明下各个参数的作用：

1. `direction`： 主轴的方向，有两个可选值：

- 纵向： `ASStackLayoutDirectionVertical` (默认)
- 横向： `ASStackLayoutDirectionHorizontal`

2. `spacing`：主轴上视图排列的间距，比如有四个视图，那么它们之间的存在三个间距值都应该是 `spacing`

3. `justifyContent`：主轴上的排列方式，有五个可选值：

- `ASStackLayoutJustifyContentStart` 从前往后排列
- `ASStackLayoutJustifyContentCenter` 居中排列
- `ASStackLayoutJustifyContentEnd` 从后往前排列
- `ASStackLayoutJustifyContentSpaceBetween` 间隔排列，两端无间隔
- `ASStackLayoutJustifyContentSpaceAround` 间隔排列，两端有间隔

4. `alignItems`：交叉轴上的排列方式，有五个可选值：

- `ASStackLayoutAlignItemsStart` 从前往后排列
- `ASStackLayoutAlignItemsEnd` 从后往前排列
- `ASStackLayoutAlignItemsCenter` 居中排列
- `ASStackLayoutAlignItemsStretch` 拉伸排列
- `ASStackLayoutAlignItemsBaselineFirst` 以第一个文字元素基线排列（主轴是横向才可用）
- `ASStackLayoutAlignItemsBaselineLast` 以最后一个文字元素基线排列（主轴是横向才可用）

5. `children`：包含的视图。数组内元素顺序同样代表着布局时排列的顺序

四、优缺点

- 导入的ASDK库有30+M
- 不支持大家常用的storyboard、xib、autoLayout，影响开发效率
- 代码没有UIKit使用熟练
- 网上资源少
- 但是可以和UIKit混合开发