Layer Animation

通过之前对UIView Animation的总结，基本的动画已经可以做了，是时候深入对Core Animation 的API进行总结啦！

Layeras VS Views

Layer是一个简单的类，给外界提供一系列的参数来代表了一些基于图像的内容，每个UIView都是由Layer层,所以你可以认为Layer层是当前内容的在屏幕后的类。

Layer与View的区别：

1、一个Layer是一个模态对象，它的数据属性和实现没有逻辑。它没有复杂的自动布局也不处理用户交互的依赖关系。

2、Layer有默认的可视化的特性—一系列的数据属性来影响内容区域显示在屏幕上，比如边框线、百年矿颜色、位置、阴影；

3、最后，Core Animation优化了Layer层的内容的缓存与快速直接的通过GPU绘制；

|  |  |
| --- | --- |
| Views | Layers |
| 复杂的等级关系、Autolayout等 | 简单的层级关系，快速解决布局、快速绘制 |
| 用户交互 | 没有响应者链 |
| 常在CPU的主线程自定义逻辑或执行绘制代码 | 默认没有自定义逻辑，常用的是在GPU上缓存和绘制 |
| 非常灵活、有力、非常多的子类 | 不怎么灵活、少量的子类 |

动画属性

CALayer一些属性直接对应的VIew属性，如：bounds

、position、opacity透明度、transform；

1、borderColor ：边框颜色

2、borderWidth：边框宽度

3、cornerRadius：圆角

4、shadowOffset：阴影距离layer的远近距离

5、shadowOpacity：阴影的透明度

6、shadowPath ：更改Layer的阴影的形状，可以使用不同的3D效果来使Layer看起来漂浮在不同形状、不同位置的阴影之上。

7、shadowRadius：控制阴影的模糊效果

8、contents ：声明一个TIFF或PNG的数据

9、mask：修改这个建立形状或图像，使用mask控制contents的的可见的内容;

10、opacity ：透明度

打开项目，注释heading的viewapper和didapper中的旧的方法，开始添加layer层动画，在viewdidApper()中创建CAbasicAnimatin

let flyRight = CABasicAnimation(keyPath: "position.x") flyRight.fromValue = -view.bounds.size.width/2 flyRight.toValue = view.bounds.size.width/2 flyRight.duration = 0 。5

动画对象在Core Animation中是简单的数据模型，你实例化一个模型，并给它设置对应的数据属性；

实例化的这个CABasicAnimation描述的时一个潜在的layer层的动画，在声明动画模型时你可以指定keypath 作为动画执行的属性，这里只是设置position.x，可以使用Core Animation公开的成员的position,bounds和transform.

接下来就是将设置好的动画添加到Layer层上，

heading.layer.addAnimation(flyRight, forkey:nil)

addAnimation(\_:forKey:)

意味着copy一份新的Animation对象，并且告诉Core Animation在Layer层执行动画，这个key只供你使用，如果你想要改变或停止动画，可以更具这个key来操作；

这个动画执行的时控件在Layer层从屏幕外进入指定的位置。

复杂的Layer Animation

/// CABasicAnimation 设置规则 layer负责展示 addAnimation 开始

1、beginTime

.beginTime = CACurrentMediaTime() + 0.3

beginTime设置的是动画开始的绝对之间，

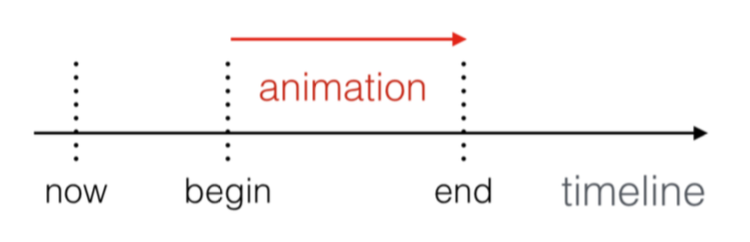
使用CACurrentMediaTime()获得当前时间，添加你想要延迟的秒数。

2、fillMode

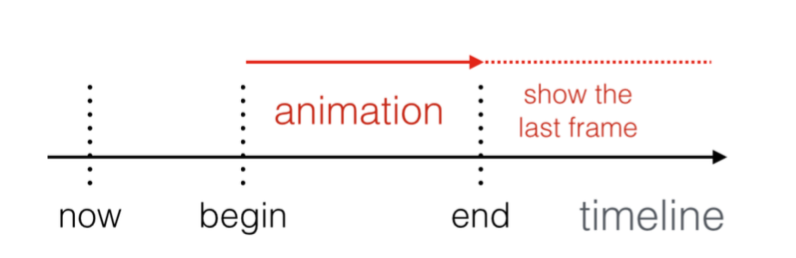
fillMode的作用就是决定当前对象过了非active时间段的行为. 比如动画开始之前,动画结束之后。如果是一个动画CAAnimation,则需要将其removedOnCompletion设置为NO,要不然fillMode不起作用.

下面来讲各个fillMode的意义

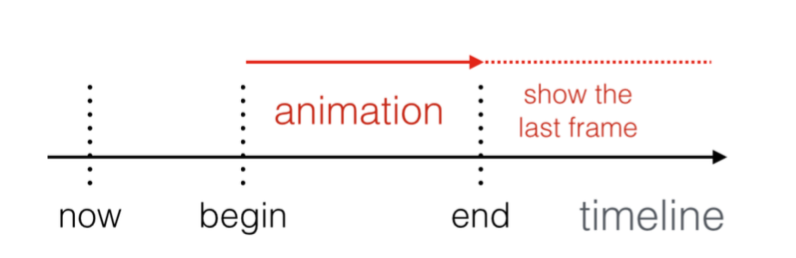
**kCAFillModeRemoved** 这个是默认值,也就是说当动画开始前和动画结束后,动画对layer都没有影响,动画结束后,layer会恢复到之前的状态



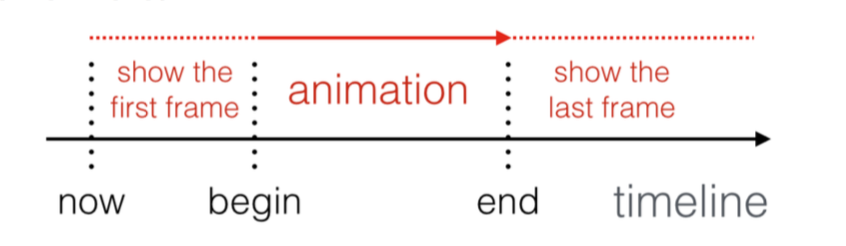
**kCAFillModeForwards** 当动画结束后,layer会一直保持着动画最后的状态



**kCAFillModeBackwards** 这个和kCAFillModeForwards是相对的,就是在动画开始前,你只要将动画加入了一个layer,layer便立即进入动画的初始状态并等待动画开始.你可以这样设定测试代码,将一个动画加入一个layer的时候延迟5秒执行.然后就会发现在动画没有开始的时候,只要动画被加入了layer,layer便处于动画初始状态



**kCAFillModeBoth** 理解了上面两个,这个就很好理解了,这个其实就是上面两个的合成.动画加入后开始之前,layer便处于动画初始状态,动画结束后layer保持动画最后的状态.



调试

为了测试，在viewwillapper()中

username.layer.position.x -= view.bounds.width password.layer.position.x -= view.bounds.width

现在username和password默认在屏幕之外了，意味着你可以在viewDidApper()中可以删除：

~~flyRight.fromValue = -view.bounds.size.width/2~~

运行之后你会发现，动画结束后，username和password消失了；

为什么呢！！

接下来调试APP为了找出UI到底发生了什么

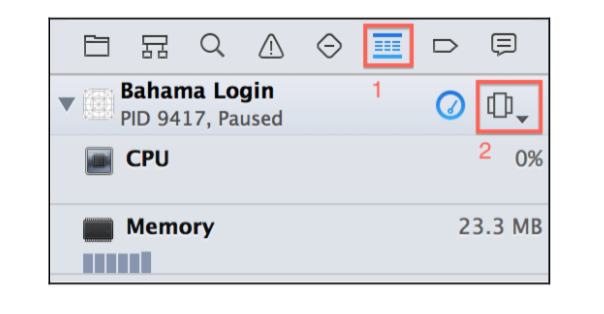
delay(seconds: 5.0, {

println("where are the fields?")

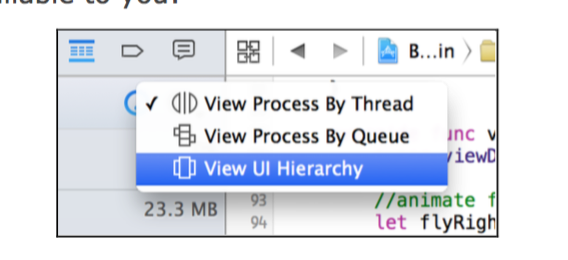
})

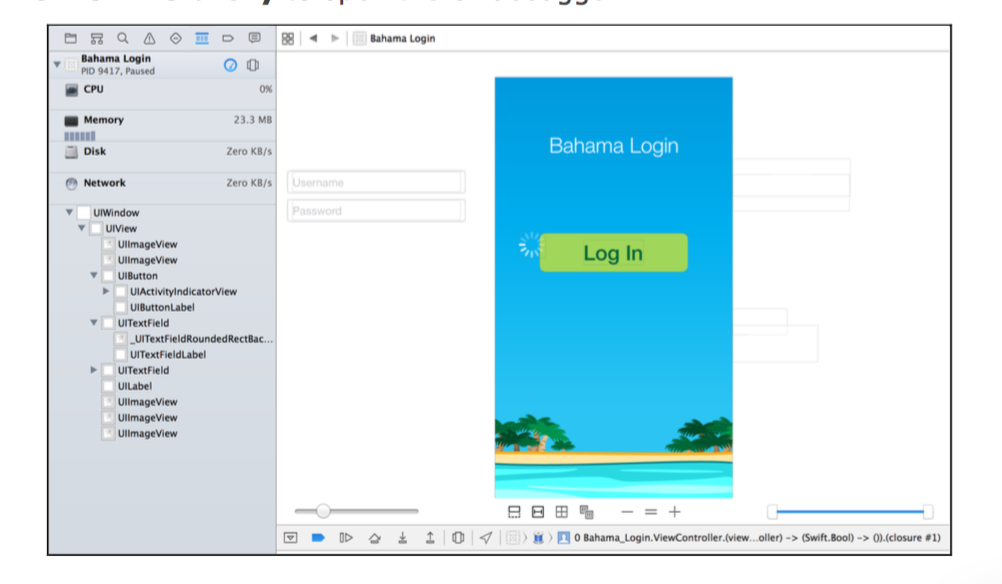
延迟5秒输出文字，在这里打个断点,运行后他会停止在这这个闭包中；

在Xcode的做左侧找到Debug Navigator，找到下图



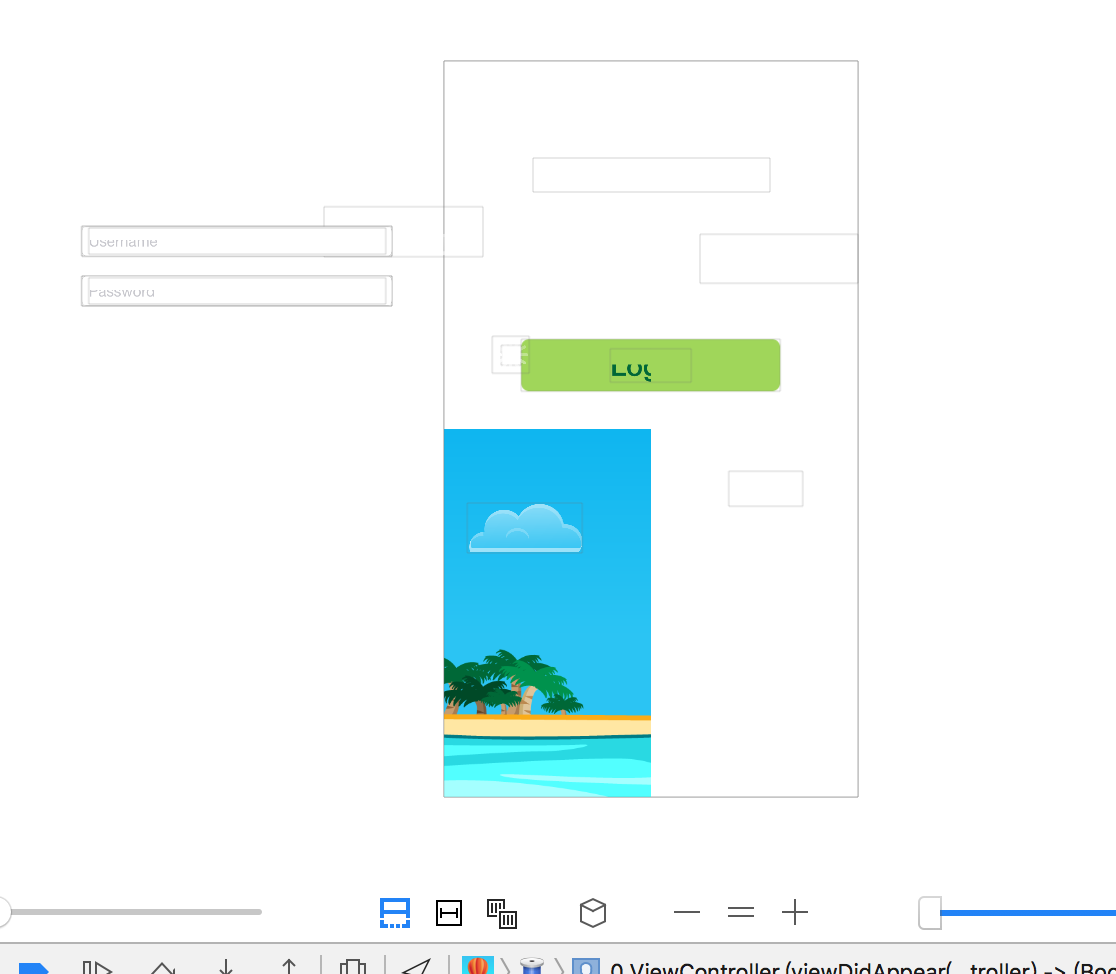
然后找到



打开View的列表：

你会看到UI层次左边UI的调试器,与当前运行时布局显示在屏幕的中心,包括任何隐藏或透明屏幕范围外的视图。

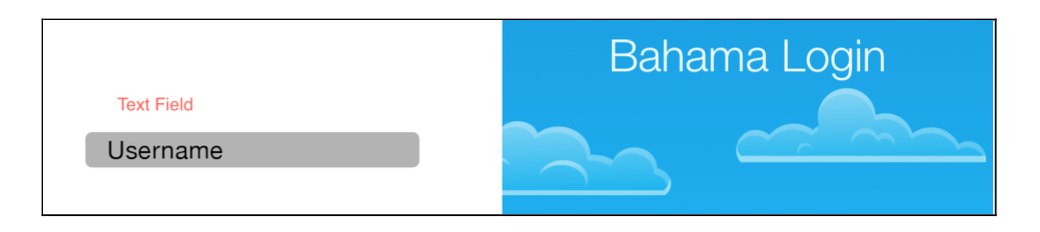
如果没有发现textfielt,如图选中中间最左边的按钮：如图



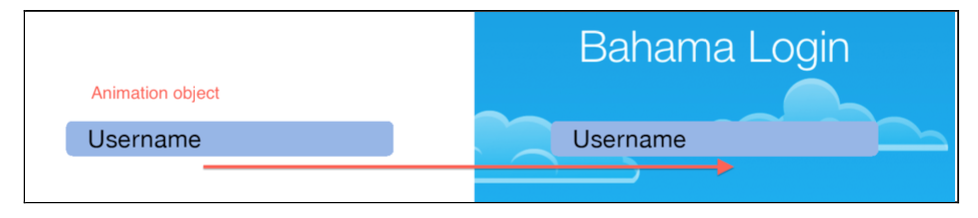
他们居然跑到可视区域的外侧，选中这个区域在辅助编辑器重来检查它的声明周期；

Animation vs 真正的内容区

当你让一个输入框运动时，事实上你没有看到输入框自身的移动，相反的你看到的是缓存的版本作为展示层来显示；表示层从屏幕动画完成后,删除原始层显示本身了。

开始，你将输入框的位置在viewWillApper()中设置在屏幕外

当动画开始后，一个已经渲染的动画对象代替输入框，原始的输入框临时隐藏：



当动画完成，它会从屏幕上消失，原始的输入框显示在你之前指定的位置（在屏幕的左侧）；

如何解决这个谜题，你需要使用另外一个CABasicAnimation的属性：removedOnCompletion

设置动画完成以后的fillMode为kCAFillModeBoth，为了实现这中效果，你需使用相应的removedOnCompletion，结合这两个属性会使动画在屏幕上展示：

在viewDidApper()，在fillmodel之后添加

flyRight.moveedOnCompletion = false

flyRight默认为true,所以动画在完成之后消失;

解下来的问题就是代替输入框不可以交互，其实时Layer层不可以交互，那么我们就要将真实的输入框显示出来，你知道moveedOnCompletion的原理，但是尽量不用使用它，因为它在性能上有点累赘，

flyRight.moveedOnCompletion = false；

你让它们自动删除就好了，只需要更新原始layer层的位置就好了。

添加

username.layer.position.x = view.bounds.size.width/2

password.layer.position.x = view.bounds.size.width/2

flyRight.fromValue = -view.bounds.size.width/2

设计你的层界面构建器与他们的最终值,并使用fromValue开始和中间值。这减少了复杂性的模型和表示层同步。

2、动画的颜色

重新创建login 按钮的颜色动画：

let tintColor = UIColor(red: 0.85, green: 0.83, blue: 0.45, alpha: 1.0)

tintBackgroundColor(layer: loginButton.layer,

toColor: tintColor)

let tint = CABasicAnimation(keyPath: "backgroundColor")

tint.fromValue = layer.backgroundColor

tint.toValue = toColor.CGColor

tint.duration = 1.0

layer.addAnimation(tint, forKey: nil)

layer.backgroundColor = toColor.CGColor

1、初始化CABasicAniamtion

2、设置三个属性：fromValue.toValue/duration

3、layer层添加CABasicAniamtion；

4、最后设置backgroundColor

3、圆角动画类似于背景色动画

let round = CABasicAnimation(keyPath: "cornerRadius")

round.fromValue = layer.cornerRadius

round.toValue = toRadius

round.duration = 0.33

layer.addAnimation(round, forKey: nil)

layer.cornerRadius = toRadius