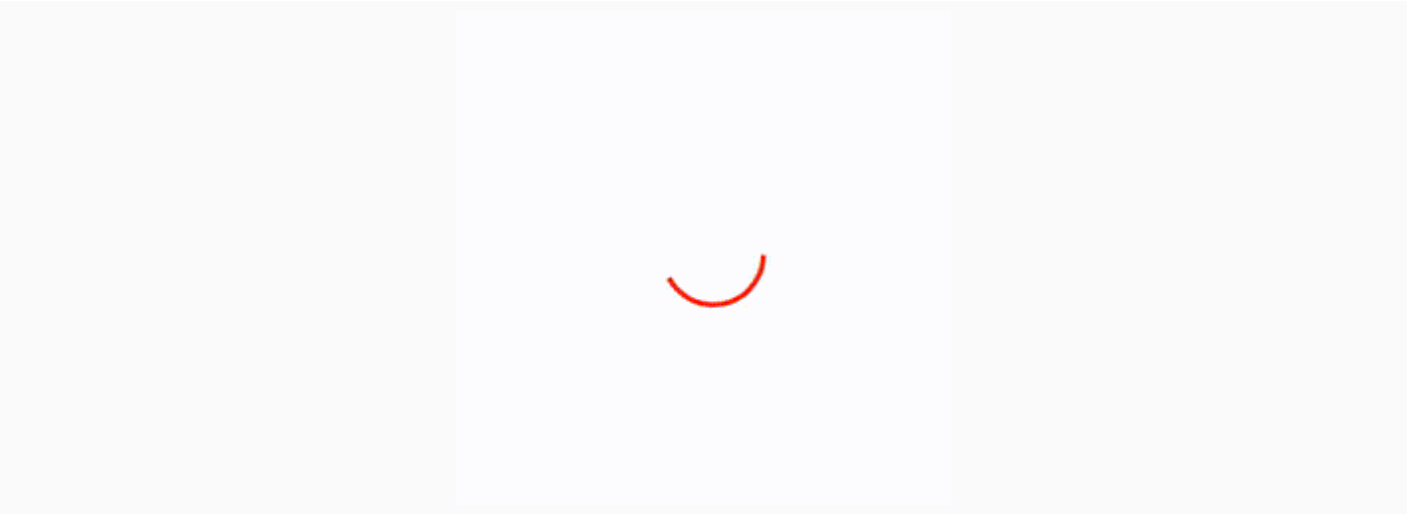
几个星期之前，Michael Villar在Motion试验中创建一个非常有趣的加载动画。

下面的GIF图片展示这个加载动画，它将一个圆形进度指示器和圆形渐现动画结合。这个组合的效果有趣，独一无二和有点迷人。



这个教程将会教你如何使用Swift和Core Animatoin来重新创建这个效果。让我们开始吧！

基础

首先下载这个教程的启动项目，然后编译和运行。过一会之后，你应该看到一个简单的image显示：

这个启动项目已经预先在恰当的位置将views和加载逻辑编写好了。花一分钟来浏览来快速了解这个项目；那里有一个ViewController，ViewController里有一个命名为CustomImageView的UIImageView子类, 还有一个SDWebImage的方法被调用来加载image。

你可能注意到当你第一次运行这个app的时候，当image下载时这个app似乎会暂停几秒，然后image会显示在屏幕。当然，此刻没有圆形进度指示器 – 你将会在这个教程中创建它！

你会在两个步骤中创建这个动画：

1. 圆形进度。首先，你会画一个圆形进度指示器，然后根据下载进度来更新它。
2. 扩展圆形图片。第二，你会通过扩展的圆形窗口来揭示下载图片。

紧跟着下面步骤来逐步实现！

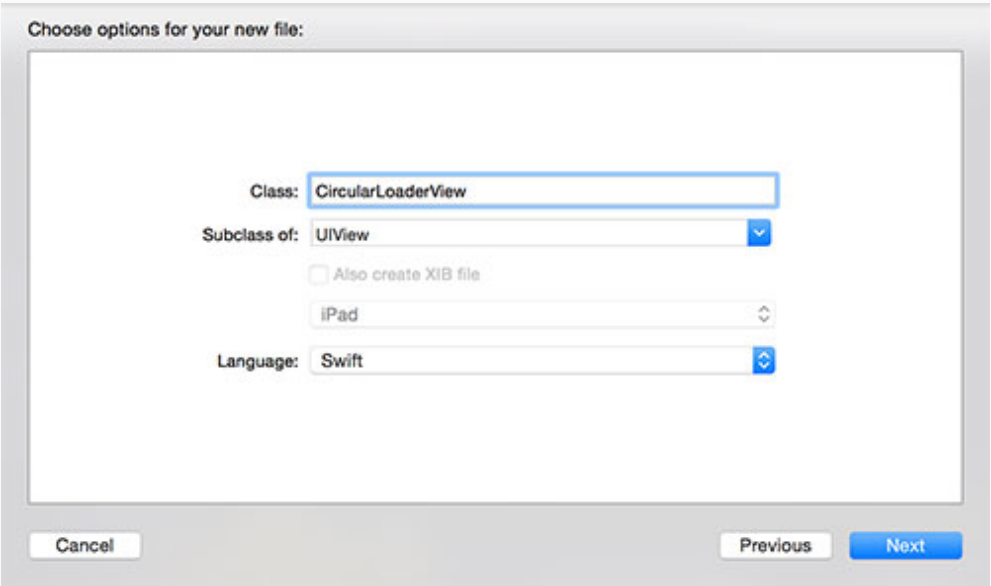
创建圆形指示器

想一下关于进度指示器的基本设计。这个指示器一开始是空来展示0%进度，然后逐渐填满直到image完成下载。通过设置CAShapeLayer的path为circle来实现是相当简单。

注意：如果你不熟悉CAShapeLayer(或CALayers)的基本概念，可以查看Scott Gardner的CALayer in iOS with Swift文章。

你可以通过CAShapeLayer的strokeStart和strokeEnd属性来控制开始和结束位置的外观。通过改变strokeEnd的值在0到1之间，你可以恰当地填充下载进度。

让我们试一下。通过iOS\Source\Cocoa Touch Class template来创建一个新的文件，文件名为CircularLoaderView。设置它为UIView的子类。



点击Next和Create。新的子类UIView将用来保存动画的代码。

打开CircularLoaderView.swift和添加以下属性和常量到这个类：

```
let circlePathLayer = CAShapeLayer()
let circleRadius: CGFloat = 20.0
```

circlePathLayer表示这个圆形路径，而circleRadius表示这个圆形路径的半径。

添加以下初始化代码到CircularLoaderView.swift来配置这个shape layer:

```
override init(frame: CGRect) {
  super.init(frame: frame)
  configure()
}

required init(coder aDecoder: NSCoder) {
  super.init(coder: aDecoder)
  configure()
}

func configure() {
  circlePathLayer.frame = bounds
  circlePathLayer.lineWidth = 2
  circlePathLayer.fillColor = UIColor.clearColor().CGColor
  circlePathLayer.strokeColor = UIColor.redColor().CGColor
  layer.addSublayer(circlePathLayer)
  backgroundColor = UIColor.whiteColor()
}
```

两个初始化方法都调用configure方法，configure方法设置一个shape layer的line width为2，fill color为clear,stroke color为red。将添加circlePathLayer添加到view's main layer。然后设置view的 backgroundColor 为white，那么当image加载时，屏幕的其余部分就忽略掉。

添加路径

你会注意到你还没赋值一个path给layer。为了做到这点，添加以下方法(还是在CircularLoaderView.swift文件)：

```
func circleFrame() -> CGRect {
  var circleFrame = CGRect(x: 0, y: 0, width: 2*circleRadius, height: 2*circleRadius)
  circleFrame.origin.x = CGRectGetMidX(circlePathLayer.bounds) - CGRectGetMidX(circleFra
  circleFrame.origin.y = CGRectGetMidY(circlePathLayer.bounds) - CGRectGetMidY(circleFra
  return circleFrame
}
```

上面那个方法返回一个CGRect的实例来界定指示器的路径。这个边框是2*circleRadius宽和2*circleRadius高，放在这个view的正中心。

每次这个view的size改变时，你会需要都重新计算circleFrame，所以你可能将它放在一个独立的方法。

现在添加以下方法来创建你的路径：

```
func circlePath() -> UIBezierPath {
  return UIBezierPath(ovalInRect: circleFrame())
}
```

这只是根据circleFrame限定来返回圆形的UIBezierPath。由于circleFrame()返回一个正方形，在这种情况下”椭圆“会最终成为一个圆形。

由于layers没有autoresizingMask这个属性，你需要在layoutSubviews方法更新circlePathLayer的frame来恰当地响应view的size变化。

下一步，覆盖layoutSubviews()方法：

```
override func layoutSubviews() {
  super.layoutSubviews()
  circlePathLayer.frame = bounds
  circlePathLayer.path = circlePath().CGPath
}
```

由于改变了frame，你要在这里调用circlePath()方法来触发重新计算路径。

现在打开CustomImageView.swift文件和添加以下CircularLoaderView实例作为一个属性：

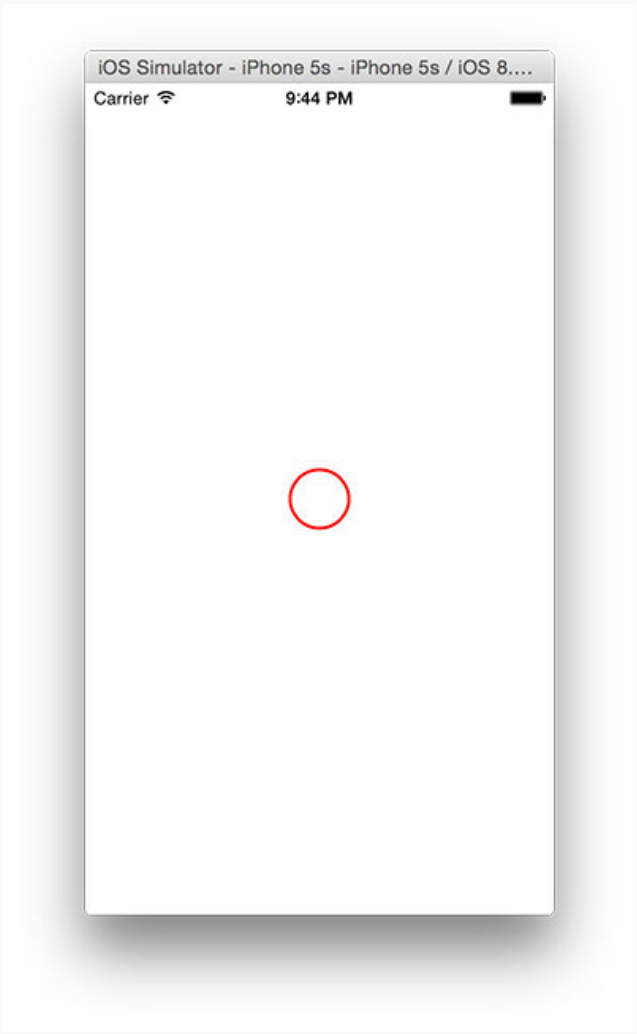
```
let progressIndicatorView = CircularLoaderView(frame: CGRectZero)
```

下一步，在之前下载图片的代码添加这几行代码到init(coder:)方法：

```
addSubview(self.progressIndicatorView)
progressIndicatorView.frame = bounds
progressIndicatorView.autoresizingMask = .FlexibleWidth | .FlexibleHeight
```

上面代码添加进度指示器作为一个subview添加到自定义的image view。autoresizingMask确保进度指示器view保持与image view的size一样。

编译和运行你的项目；你会看到一个红的、空心的圆形出现，就像这样：



好的 – 你已经有进度指示器画在屏幕上。你的下一个任务就是根据下载进度变化来stroke。

修改Stroke长度

回到CircularLoaderView.swift文件和在这个文件的其他属性直接添加以下代码：

```
var progress: CGFloat {
  get {
    return circlePathLayer.strokeEnd
  }
  set {
    if (newValue > 1) {
      circlePathLayer.strokeEnd = 1
    } else if (newValue < 0) {
      circlePathLayer.strokeEnd = 0
    } else {
      circlePathLayer.strokeEnd = newValue
    }
  }
}
```

以上代码创建一个computed property – 也就是一个属性没有任何后背的变量 – 它有一个自定义的setter和getter。这个getter只是返回circlePathLayer.strokeEnd，setter验证输入值要在0到1之间，然后恰当地设置layer的strokeEnd属性。

在第一次运行的时候，添加下面这行代码到configure()来初始化进度：

```
progress = 0
```

编译和运行工程；除了一个空白的屏幕，你应该什么也没看到。相信我，这是一个好消息。设置progress为0，反过来会设置strokeEnd也为0，这就意味着shape layer什么也没画。

唯一剩下要做的就是你的指示器在image下载回调方法中更新progress。

回到CustomImageView.swift文件和用以下代码来代替注释Update progress here：

```
self!.progressIndicatorView.progress = CGFloat(receivedSize)/CGFloat(expectedSize)
```

这主要通过receivedSize除以expectedSize来计算进度。

注意：你会注意到block使用weak self引用 – 这样能够避免retain cycle。

编译和运行你的工程；你会看到进度指示器像这样开始移动：

即使你自己没有添加任何动画代码，CALayer在layer轻松地发现任何animatable属性和当属性改变时平滑地animate。

上面已经完成第一个阶段。现在进入第二和最后阶段。

创建Reveal动画

reveal阶段在window显示image然后逐渐扩展圆形环的形状。如果你已经读过前面教程，那个教程主要讲创建一个Ping风格的view controller动画，你就会知道这是一个很好的关于CALayer的mask属性的使用案例。

添加以下方法到CircularLoaderView.swift文件：

```
func reveal() {

  // 1
  backgroundColor = UIColor.clearColor()
  progress = 1
  // 2
  circlePathLayer.removeAnimationForKey("strokeEnd")
  // 3
  circlePathLayer.removeFromSuperlayer()
  superview?.layer.mask = circlePathLayer
}
```

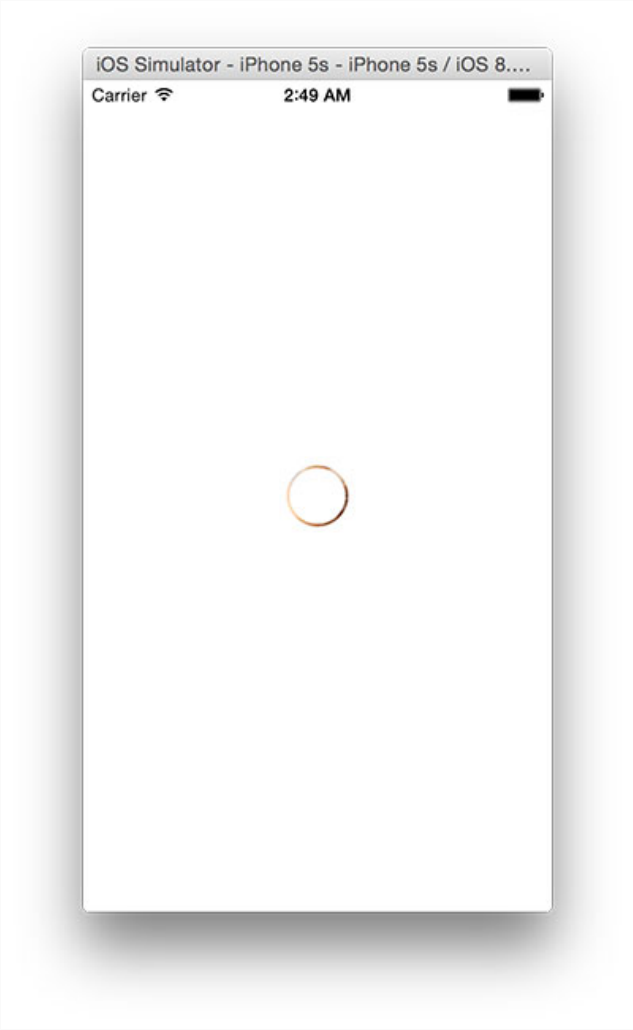
这是一个很重要的方法需要理解，让我们逐段看一遍：

1. 设置view的背景色为clear，那么在view后面的image不再隐藏，然后设置progress为1或100%。
2. 使用strokeEnd属性来移除任何待定的implicit animations，否则干扰reveal animation。关于implicit animations的更多信息，请查看iOS Animations by Tutorials.
3. 从它的superLayer移除circlePathLayer，然后赋值给superView的layer maks，借助circular mask “hole”，image是可见的。这样让你复用已存在的layer和避免重复代码。

现在你需要在某个地方调用reveal()。在CustomImageView.swift文件用以下代码替换Reveal image here注释：

```
self!.progressIndicatorView.reveal()
```

编译和运行你的app；一旦image开始下载，你会看见一部分小的ring在显示。



你能在背景看到你的image – 但几乎什么也没有！

扩展环

你的下一步就是在内外扩展这个环。你可以两个分离的、同轴心的UIBezierPath来做到，但你也可以一个更加有效的方法，只是使用一个Bezier path来完成。

怎样做呢？你只是增加圆的半径(path属性)来向外扩展，同时增加line的宽度(lineWidth属性)来使环更加厚和向内扩展。最终，两个值都增长到足够时就在下面显示整个image。

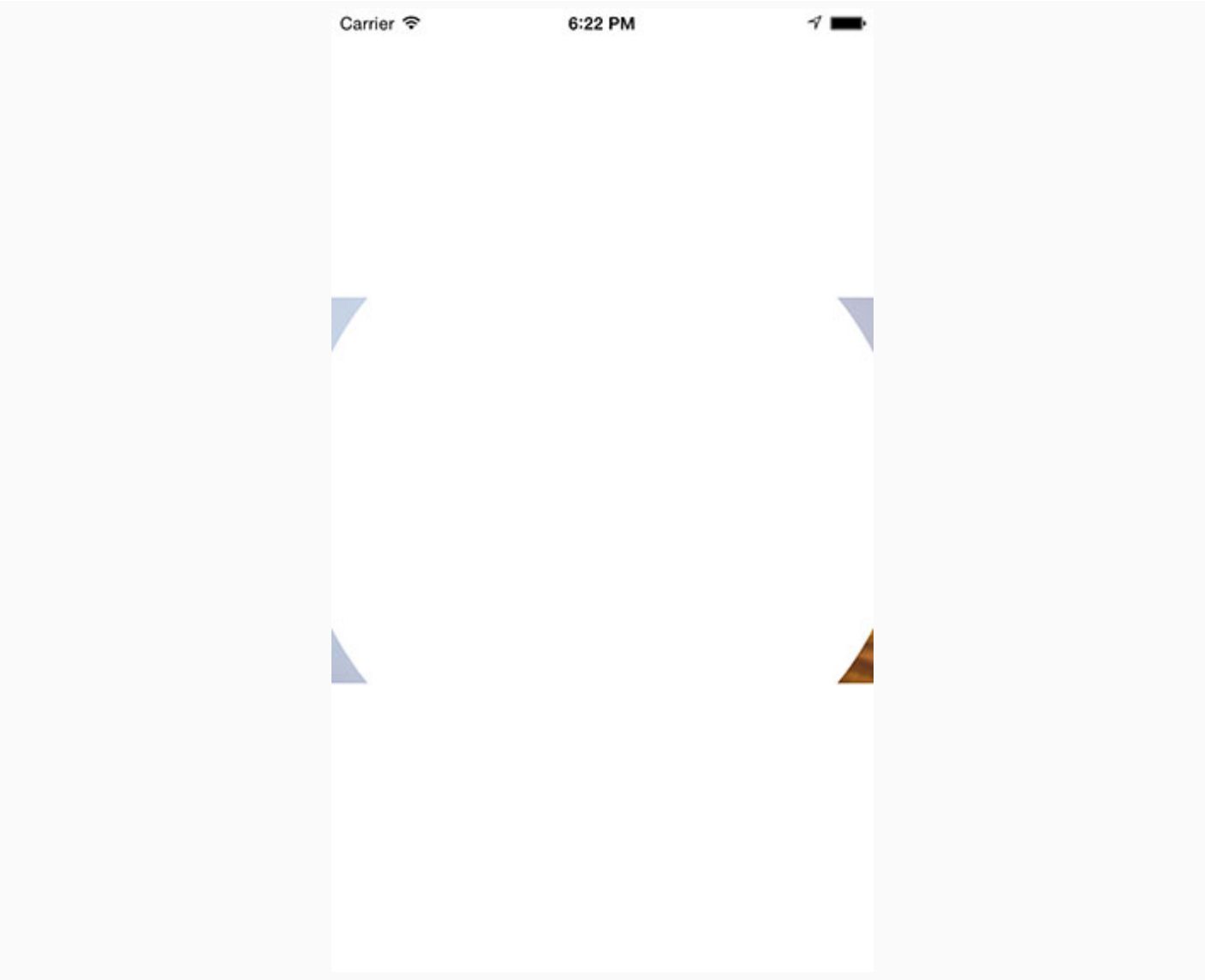
回到CircularLoaderView.swift文件和添加以下代码到reveal()方法的最后：

```
// 1
let center = CGPoint(x: CGRectGetMidX(bounds), y: CGRectGetMidY(bounds))
let finalRadius = sqrt((center.x*center.x) + (center.y*center.y))
let radiusInset = finalRadius - circleRadius
let outerRect = CGRectInset(circleFrame(), -radiusInset, -radiusInset)
let toPath = UIBezierPath(ovalInRect: outerRect).CGPath

// 2
let fromPath = circlePathLayer.path
let fromLineWidth = circlePathLayer.lineWidth

// 3
CATransaction.begin()
CATransaction.setValue(kCFBooleanTrue, forKey: kCATransactionDisableActions)
circlePathLayer.lineWidth = 2*finalRadius
circlePathLayer.path = toPath
CATransaction.commit()

// 4
let lineWidthAnimation = CABasicAnimation(keyPath: "lineWidth")
lineWidthAnimation.fromValue = fromLineWidth
lineWidthAnimation.toValue = 2*finalRadius
let pathAnimation = CABasicAnimation(keyPath: "path")
pathAnimation.fromValue = fromPath
pathAnimation.toValue = toPath

// 5
let groupAnimation = CAAnimationGroup()
groupAnimation.duration = 1
groupAnimation.timingFunction = CAMediaTimingFunction(name: kCAMediaTimingFunctionEaseI
groupAnimation.animations = [pathAnimation, lineWidthAnimation]
groupAnimation.delegate = self
circlePathLayer.addAnimation(groupAnimation, forKey: "strokeWidth")
```

现在逐段解释以上代码是究竟做了什么：

1. 确定圆形的半径之后就能完全限制image view。然后计算CGRect来完全限制这个圆形。toPath表示CAShapeLayer mask的最终形状。
2. 设置lineWidth和path初始值来匹配当前layer的值。
3. 设置lineWidth和path的最终值；这样能防止它们当动画完成时跳回它们的原始值。CATransaction设置kCATransactionDisableActions键对应的值为true来禁用layer的implicit animations。
4. 创建一个两个CABasicAnimation的实例，一个是路径动画，一个是lineWidth动画，lineWidth必须增加到两倍跟半径增长速度一样快，这样圆形向内扩展与向外扩展一样。
5. 将两个animations添加到一个CAAnimationGroup，然后添加animation group到layer。将self赋值给delegate，等下你会使用到它。

编译和运行你的工程；你会看到一旦image完成下载，reveal animation就会弹出来。但即使reveal animation完成，部分圆形还是会保持在屏幕上。

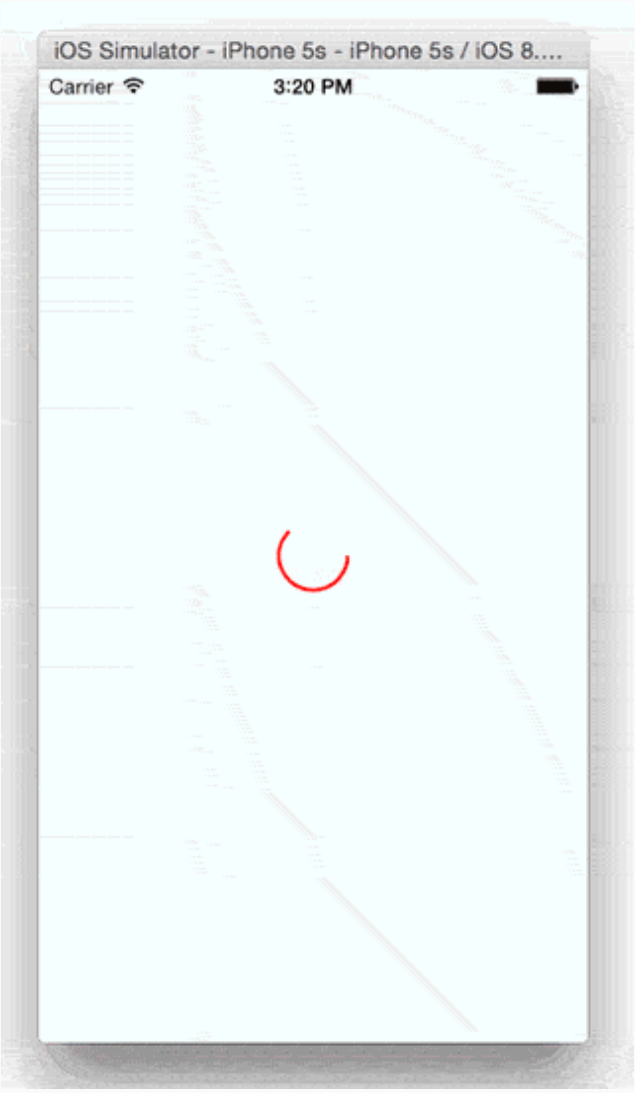


为了修复这种情况，添加以下实现animationDidStop(_:finished:) 到 CircularLoaderView.swift：

```
override func animationDidStop(anim: CAAnimation!, finished flag: Bool) {
  superview?.layer.mask = nil
}
```

这些代码从super layer上移除mask，这会完全地移除圆形。

再次编译和运行你的工程，和你会看到整个动画的效果：



恭喜你，你已经完成创建圆形图像加载动画!

下一步

你可以在这里下载整个工程。

基于本教程，你可以进一步来微调动画的时间、曲线和颜色来满足你的需求和个人设计美学。一个可能需要改进就是设置shape layer的lineCap属性值为kCALineCapRound来四舍五入圆形进度指示器的尾部。你自己思考还有什么可以改进的地方。

如果你喜欢这个教程和愿意学习怎样创建更多像这样的动画，请查看Marin Todorov的书iOS Animations by Tutorials。它是从基本的动画开始，然后逐步讲解layer animations, animating constraints, view controller transitions和更多

如果你有什么关于这个教程的问题或评论，请在下面参与讨论。我很乐意看到你在你的App中添加这么酷的动画。