# 第九章UIView和CALayer层

窗口和UIView视图是为iOS应用程序构造用户界面的可视化组件。窗口为内容显示提供背景平台，而UIview视图负责绝大部分的内容描绘，并负责响应用户的交互。

UIView的显示依赖于其内部的CALayer层对象。UIView的显示由一个CALayer类来管理，其本身只负责说明要显示的内容。当访问和调整UIView中如frame、bounds等关于绘制的属性，实际上都是在访问和调整它所包含的CALayer的相关属性。

本章将逐步介绍UIview视图及CALayer层的原理和一些具体的使用方法。

## 5.1视图UIview

### 5.1.1UIview概述

UIWindow

NSObject

UILabel

UITableView

UITableViewCell

UITextView

UIScrollView

UICollectionView

UIPickerView

UIResponder

UIProgressView

UIActivationView

UIView

UIButton

UISearchBar

UIDatePicker

UIActionSheet

UIPageControl

UITabBar

UISegmentControl

UIToolBar

UITextField

UIControl

UINavigationBar

UISlider

UISwitch

UIImageView

UIAlertView

UIWebView

图1 UIView的继承层次结构

UIview是UIKit框架里面最基础的视图类。UIview类定义了一个矩形的区域，管理该矩形区域内的所有显示内容。在应用程序中还需要响应该区域中发生的触碰事件。视图是应用程序与用户交互的重要机制。

UIview类定义了视图的基本行为，但并不定义其视觉表现，而是UIKit通过子类来为文本框、按键及工具条这样的标准界面元素定义具体的外观和行为。

UIview继承层次如图1所示。

接下来简单介绍一下UIWindow。UIWindow是一个比较特殊的视图控件，实际的作用就是提供一个展示App内容的窗口。iOS程序启动后，创建的第一个视图控件就是UIWindow，接着创建视图控制器的view，并将该view添加到UIWindow上，于是控制器的view就显示在屏幕上了。

iOS应用程序通常只有一个窗口，表示为一个UIWindow类的实例，应用程序在启动时创建这个窗口，并往窗口中加入一个或多个视图，然后将它显示出来。窗口一旦显示出来，基本上就不会再使用到它了，而更多的是对UIview视图的操作。在iOS应用程序中，窗口对象并没有像关闭框或标题栏这样的区域，所以用户不能直接对其进行关闭或其他操作。

创建新项目，模板为iOS中的【Single View App】,创建完成后在ViewControl- -ler.swift中编写代码来创建一个视图并定义其属性。

1. import UIKit
2. class ViewController: UIViewController {
3. override func viewDidLoad() {
4. super.viewDidLoad()
5. // Do any additional setup after loading the view, typically from a nib.
6. //定义视图位置和大小并创建视图
7. let view = UIView(frame: CGRect(x: 40, y: 100, width: 334, height: 334))
8. //定义其背景颜色为红色
9. view.backgroundColor = UIColor.black
10. //添加视图于根视图
11. self.view.addSubview(view)
12. }
13. }

运行项目，结果如图所示。

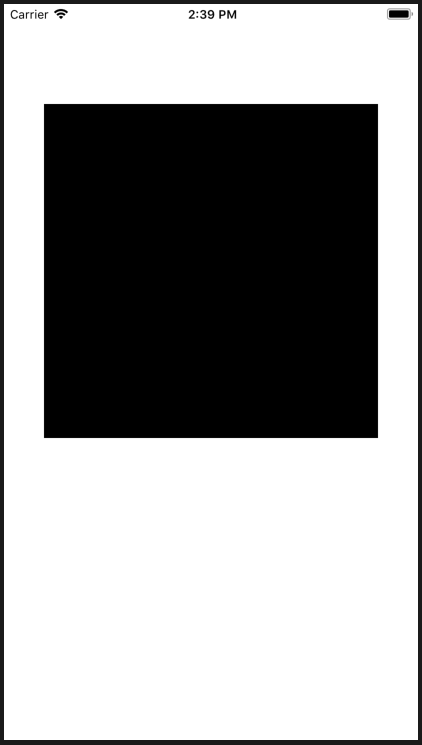


图2 使用UIView创建正方形视图

下面将逐一介绍UIView中的各种属性.

### 5.1.2 UIView的外观属性

UIView类的外观属性常用的主要有背景颜色、切边、透明度、显示与隐藏。

backgroundColor为背景颜色属性，可以用UIColor来设置单一颜色背景。UIColor是UIKit中存储颜色属性的重要类，其中包含一些常用颜色的方法，如红色、白色、黑色等。一个视图的背景还可以设为一个图片，如把5.1.1节中的代码的7~11行换成：：

1. let view = UIView(frame: CGRect(x: 40, y: 100, width: 334, height: 334))
2. let image = UIImage(named: "gezi")
3. view.backgroundColor = UIColor.init(patternImage:image!)
4. self.view.addSubview(view)

点击项目文件夹，在弹出的菜单中选择【Add a File】，添加一张图片到项目中。通过UIImage对象加载资源文件夹的一张图片，然后使用UIColor中的init方法将加载的图片设置为背景。结果如图2所示。

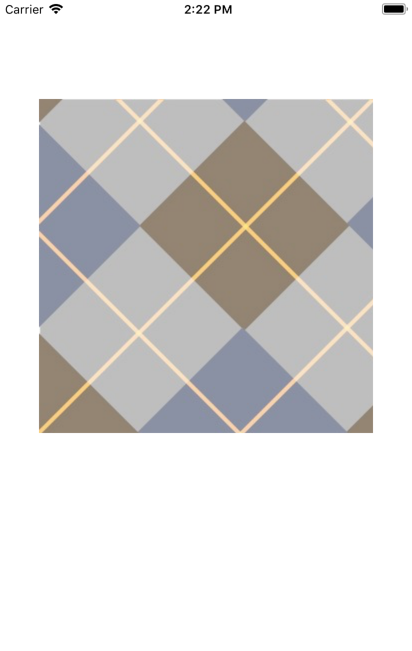


图3 设置格子图片为视图背景

UIView可以通过clipToBounds属性来限制视图的显示区域是否能够超过其父视图区域。如果设置为true，说明不能超过，超过的部分会沿着父视图边界切除，即切边。由于涉及UIView的层次和嵌套，所以切边将在5.1.4做实例展示。

UIView可以通过hidden属性设置视图的显示与隐藏。

视图的hidden属性为布尔值类型，表示UIView的视图是否处于隐藏状态，默认值为false。修改5.1.1代码的7~11行如下：：

1. let view = UIView(frame: CGRect(x: 40, y: 100, width: 334, height: 334))
2. view.isHidden = true
3. view.backgroundColor = UIColor.black
4. self.view.addSubview(view)

这样将不会再屏幕上看到已经设置好的视图。

视图的透明度可根据调节透明度属性alpha的值来实现。

视图的透明度alpha属性是一个浮点数类型，取值范围为0~1.0，从完全透明到完全不透明，默认alpha值为1。修改5.1.1代码的7~11行如下：

1. let view = UIView(frame: CGRect(x: 40, y: 100, width: 334, height: 334))
2. view.alpha = 0.2
3. view.backgroundColor = UIColor.black
4. self.view.addSubview(view)

结果如图3所示。



图4 设置视图透明度为0.2

注：当把alpha值设为0或者isHidden值设为true时，当前的UIView视图及其子视图都会被隐藏，不管子视图的alpha或者isHidden值为多少。

### 5.1.3 UIView的几何属性

在iOS系统中，点的位置是依靠iOS系统的坐标系来确定的，坐标原点位于视图左上角。iOS包含两个坐标系，其中UIKit框架的坐标系是X轴正方向向右，Y轴正方向向下，而标准的Quartz 2D绘图坐标系为X轴正方向向右，Y轴正方向向上。

一个UIView对象的位置和大小属性，是由两个属性值来组成的：CGPoint和CGSize。其中CGPoint由x、y两个值决定，代表视图在父视图上的相对于原点的位置，CGSize由width和height两个值决定，代表视图的大小。在声明一个UIView的对象时，可以用以下三种属性来定义：frame、bounds和center。

frame决定该视图在父视图的位置及其大小。frame使用方法如下：

UIView.frame(CGRect(x:，y:，width:，height:))

UIView.frame(CGRect(origin:CGPoint，size:CGSize))

bounds决定该视图在其本身的坐标系中的位置和大小，通常用于改变视图的大小。其中origin属性默认为（0，0）,实际代表该视图左上角的坐标，即将origin属性值设置为其他值无法改变该视图的位置，但其子视图的位置会发生改变。使用方法如下：

UIView.bounds(CGRect(x:，y:，width:，height:))

UIView.bounds(CGRect(origin:CGPoint，size:CGSize))

比如想改变某视图的大小，就改变width和height的值或者size属性的值：

UIView.bounds(CGRect(x:0，y:0,width:100,height:100))

该行代码就是把视图长宽都设为100。注意，bounds是基于视图中心点位置不变来改变该视图大小。

Center决定该视图的中心在其父视图中的位置。使用方法如下：

UIView.center(CGPoint(x:，y:))

下面将用代码来定义视图坐标和尺寸信息。

1. import UIKit
2. class ViewController: UIViewController {
3. override func viewDidLoad() {
4. super.viewDidLoad()
5. // Do any additional setup after loading the view, typically from a nib.
6. //定义并创建视图,设置其背景颜色为红色
7. let frame = CGRect(x: 40, y: 100, width: 334, height: 334)
8. let view = UIView(frame: frame)
9. view.backgroundColor = UIColor.red
10. //定义并创建另一个视图，设置其背景颜色为黑
11. let subview = UIView(frame: CGRect(x: 0, y: 0, width: 260, height: 260))
12. subview.backgroundColor = UIColor.black
13. //添加第二个视图于第一个视图
14. view.addSubview(subview)
15. //添加第一个视图于根视图
16. self.view.addSubview(view)
17. }
18. }

在17、20行代码中将视图作为子对象添加到另一个视图对象中的方法请查看5.1.4节的介绍。

运行该项目，效果如图5所示。

接下来设置子视图的bounds属性。在第15行代码下添加一行代码。

1. subview.backgroundColor = UIColor.black
2. subview.bounds = CGRect(x: 0, y: 0, width: 200, height: 200)
3. view.addSubview(subview)

运行项目，效果如图6所示。可以看到，子视图中心位置未变。

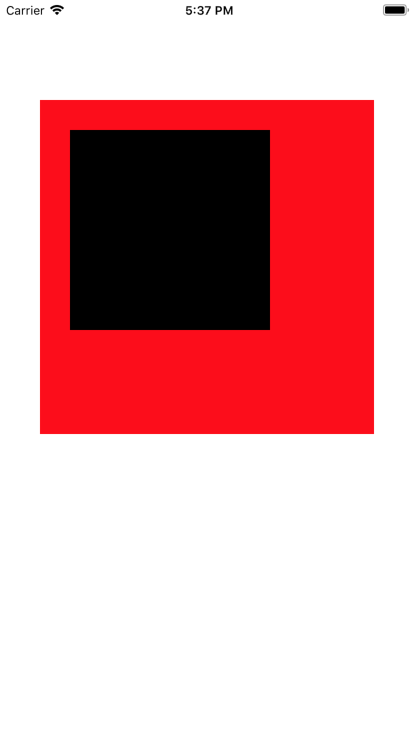
 

图5 在一个视图上添加另一个视图 图6改变子视图大小

接下来设置父视图的bounds属性,在第8行代码下方添加代码。

1. let view = UIView(frame: frame)
2. view.bounds = CGRect(x: -100, y: -100, width: 334, height: 334)
3. view.backgroundColor = UIColor.red

运行项目，效果如图7所示。可以看到subview视图位置改变，这是因为view视图的左上角的坐标在自身的坐标系中变为（-100，-100），subview作为view的子视图要以view的坐标系中的（0，0）为原点。

设置父视图的center属性，在第20行上方添加代码。

1. view.center = self.view.center
2. self.view.addSubview(view)

运行项目，效果如图8所示,把view视图的中心移到跟视图的中心。

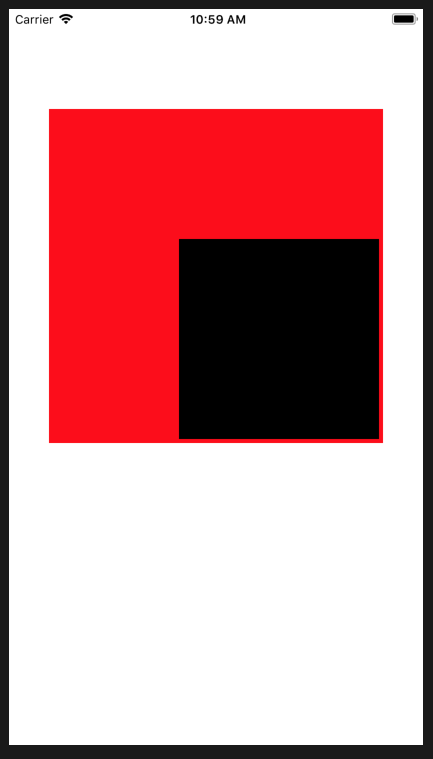
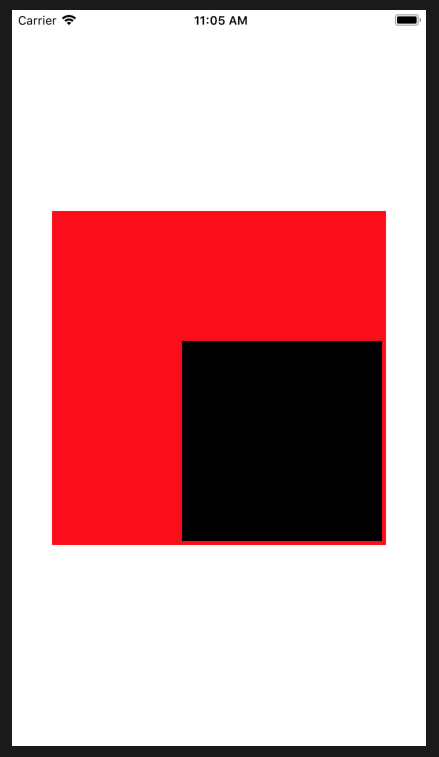
 

图7 图8

### 5.1.4 UIView的嵌套和层次关系

一个视图中可以嵌套多个子视图。而嵌套的多个子视图会 形成复杂的图层结构，视图的这种布局方式被称为视图层次。一个视图可以包含任意数量的子视图，通过为子视图添加子视图的方式，可实现任意深度的嵌套。

对于子视图在屏幕上的显示方式，虽然子视图总是显示在父视图的上方，但是同一层子视图之间可能会发生重叠或者覆盖。这是由视图层次的组织方式来决定的。它还决定了视图对于事件的响应和变化方式。子视图的层次结构由父视图管理。

UIView对于视图层次的管理方法如下表所示。

表 UIView视图层次管理方法

|  |  |
| --- | --- |
| 方法名称 | 方法说明 |
| addSubview(view:) | 添加视图于父视图层级结构的最上层 |
| insertSubview(view:,at:) | 在指定的位置上插入式图 |
| insertSubview(view:,aboveSubview:) | 将试图添加到指定视图的上方 |
| insertSubview(view:,belowSubview:) | 将试图添加到指定视图的下方 |
| bringSubview(toFront:) | 将指定的子视图移动到最前面 |
| sendSubview(toBack:) | 将指定的子视图移动到最后面 |
| exchangeSubview(at:,withSubviewAt:) | 交换两个指定位置的子视图在父视图的位置 |
| removeFromSuperview | 将子视图从父视图中删除 |

接下来将要用代码创建相同尺寸但拥有不同背景颜色和位置的3个视图，并将它们都添加到根视图上，以便展示视图之间的层次转换。

1. import UIKit
2. class ViewController: UIViewController {
3. override func viewDidLoad() {
4. super.viewDidLoad()
5. // Do any additional setup after loading the view, typically from a nib.
6. //创建第一个视图并添加到根视图中
7. let view1 = UIView(frame: CGRect(x: 40, y: 100, width: 200, height: 200))
8. view1.backgroundColor = UIColor.black
9. self.view.addSubview(view1)
10. //创建第二个视图并添加到根视图中
11. let view2 = UIView(frame: CGRect(x: 100, y: 140, width: 200, height: 200))
12. view2.backgroundColor = UIColor.brown
13. self.view.addSubview(view2)
14. //创建第三个视图并添加到根视图中
15. let view3 = UIView(frame: CGRect(x: 160, y: 180, width: 200, height: 200))
16. view3.backgroundColor = UIColor.lightGray
17. self.view.addSubview(view3)
18. }
19. }

运行项目，结果如图9所示。可以看到最初的层次是按照添加到根视图的先后来排的，先添加到根视图中的视图在最后一层，最晚添加的在最上层。

接下来调整view1和view2在父视图中的层次。在第17行代码下面添加一行代码：

1. self.view.addSubview(view3)
2. self.view.insertSubview(view1, aboveSubview: view2)

运行项目，结果如图10所示。可以看到view1的层次在view2之上。继续添加代码：

1. self.view.addSubview(view3)
2. self.view.insertSubview(view1, aboveSubview: view2)
3. self.view.sendSubview(toBack: view3)

运行项目，结果如图11所示。继续添加代码：

1. self.view.sendSubview(toBack: view3)
2. view2.removeFromSuperview()

运行项目，结果如图12所示。

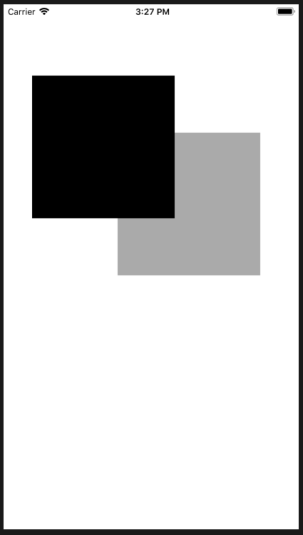
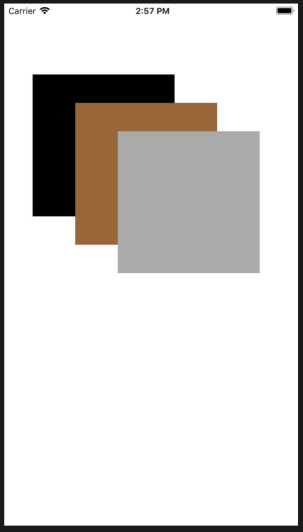
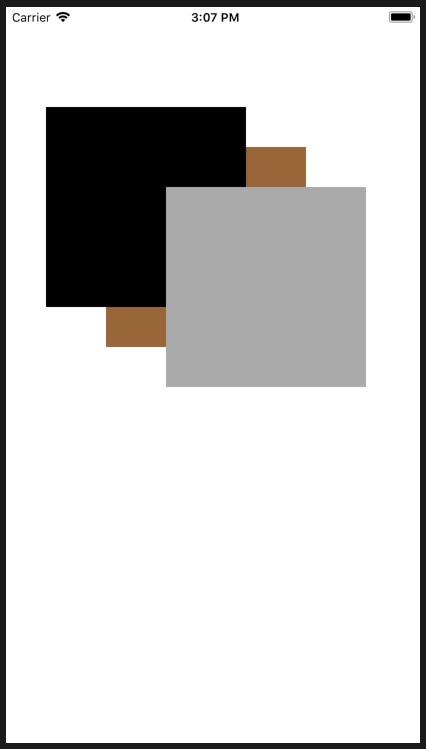
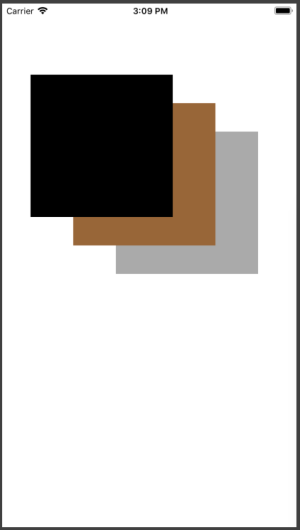


图9 图10 图11 图12

### 5.1.5 UIView的变形操作

UIView的变形操作由CGAffinetransform结构体控制。该结构体代表了一种用于仿射变换的矩阵。结构体的参数指定了从一个坐标系的点转化成另外一个坐标系的点的规则。

仿射变换是一种特殊的映射，保留在一个路径中的平行线，但不一定保留长度或角度。缩放、旋转、平移是最常用的仿射变换。开发者通常不会直接创建一个仿射变换，只需要根据现有的参数，修改现有的仿射变换即可。

常用的几种仿射变换见下表。

表 几种常用的仿射变换

|  |  |
| --- | --- |
| 名称 | 说明 |
| translatedBy（x:,y:） | 对已存在的矩阵进行平移 |
| Invert() | 对已存在的矩阵进行反转 |
| scaledBy(x:,y:) | 对已存在的矩阵进行旋转 |
| Rotated(by:) | 对已存在的矩阵进行反转 |
| Concatenating(t2:) | 对仿射效果进行叠加操作 |

接下来会使用上表的方法实现视图的变形。

创建新项目，模板为iOS中的【Single View App】,创建完成后在ViewControl- -ler.swift中编写代码来创建一个视图并定义其属性。

1. import UIKit
2. class ViewController: UIViewController {
3. override func viewDidLoad() {
4. super.viewDidLoad()
5. // Do any additional setup after loading the view, typically from a nib.
6. //创建一个视图，背景为黑色，并添加到根视图
7. let view = UIView(frame: CGRect(x: 50, y: 200, width: 314, height: 100))
8. view.backgroundColor = UIColor.black
9. self.view.addSubview(view)
10. }
11. }

运行项目，结果如图13所示。在第9行代码后添加代码。

1. self.view.addSubview(view)
2. view.transform = view.transform.translatedBy(x: 50, y: 0)

结果如图14所示，可以看到视图view向右平移了40.

将第10行代码改为：

1. view.transform = view.transform.scaledBy(x: 0.5, y: 0.5)

运行结果如图15所示，可以看到view视图在x水平方向和y垂直方向上各缩小了0.5倍。其中心位置没有变化。

将第10行代码改为：

1. view.transform = view.transform.rotated(by: CGFloat(Double.pi/4))

运行结果如图16所示。可以看到view视图以顺时针方向旋转了45°。Double.pi在swift语言中表示圆周率。

以上的对视图的平移、缩放和旋转都是采用已封装好的方法，而对于斜切却没有。开发者只能通过设置参数的值来进行斜切变形。参数说明见下表。

表 变形参数

|  |  |  |  |
| --- | --- | --- | --- |
| 参数 | 说明 | 参数 | 说明 |
| a | 水平方向上的缩放因子 | d | 垂直方向上的缩放因子 |
| b | 水平方向上的斜切因子 | tx | 水平方向上的位移因子 |
| c | 垂直方向上的斜切因子 | ty | 水平方向上的位移因子 |

在第10行代码下方添加代码：

1. view.transform = view.transform.rotated(by: CGFloat(Double.pi/4))
2. view.transform.b = 0.3
3. view.transform.c = 0.3

运行结果如图17所示。

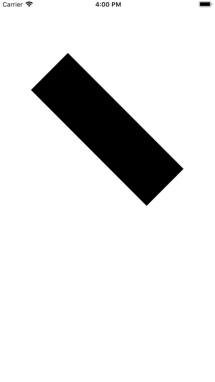
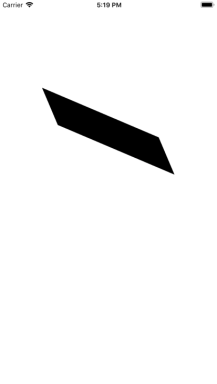
    

图13 图14 图15 图16 图17

## 5.2 CALayer层

在本章开始就已说明，UIView是对视图内容的管理，而UIView真正的绘图部分，是由CALayer（Core Animation Layer）类来管理的。对于创建的视图或者控件，有时不会满足于其只是矩形的视觉效果。通过CALayer可以给视图添加边框、阴影、圆角、颜色渐变等效果。

### 5.2.1 CAlayer边框

创建新项目，模板为iOS中的【Single View App】,创建完成后在ViewControl- -ler.swift中编写以下代码。

1. import UIKit
2. class ViewController: UIViewController {
3. override func viewDidLoad() {
4. super.viewDidLoad()
5. // Do any additional setup after loading the view, typically from a nib.
6. //创建一个视图，背景为黑色，并添加到根视图
7. let view = UIView(frame: CGRect(x: 100, y: 200, width: 214, height: 214))
8. view.backgroundColor = UIColor.black
9. //设置边框效果，边框宽为20，颜色为灰色
10. view.layer.borderWidth = 20
11. view.layer.borderColor = UIColor.gray.cgColor
12. //把该视图添加到根视图
13. self.view.addSubview(view)
14. }
15. }

对于第11行，由于CALayer层的颜色使用CGColor类型，所以要使用UIColor中的cgColor属性。运行项目，结果如图18所示。

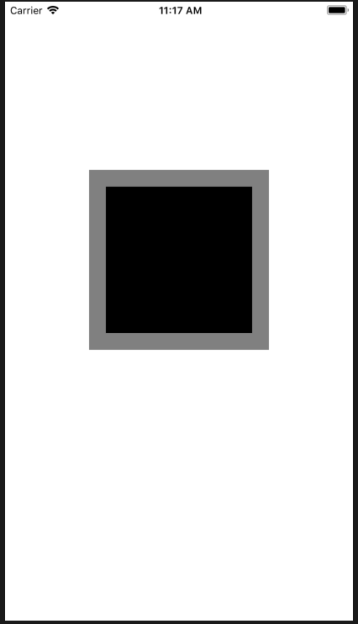
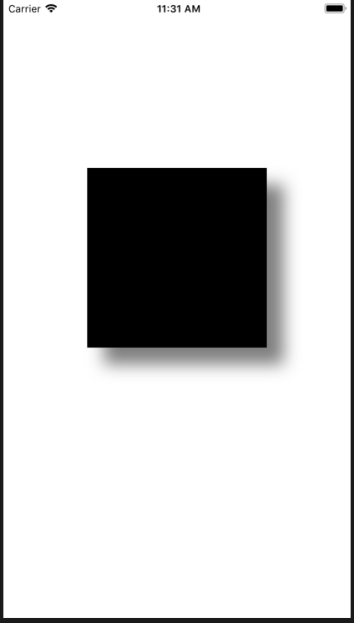
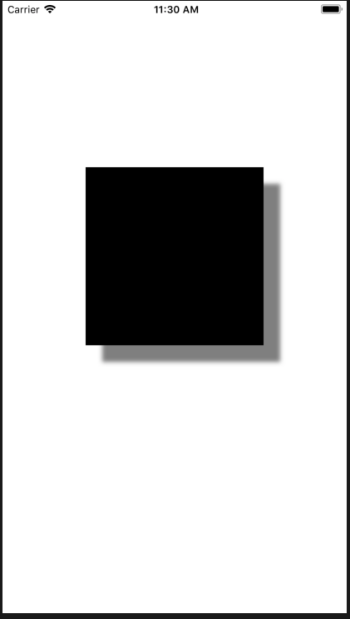
  

图18 图19 图20

### 5.2.2 CALayer阴影

将5.2.1中的代码的9~11行改为：

1. //设置阴影效果，颜色为黑色,阴影偏移量为（20，20）
2. view.layer.shadowOffset = CGSize(width: 20, height: 20)
3. view.layer.shadowColor = UIColor.black.cgColor
4. //设置阴影透明度为0.5，圆角半径为10
5. view.layer.shadowOpacity = 0.5
6. view.layer.shadowRadius = 10

运行项目，结果如图19所示。shadowOffset属性为阴影偏移量，类型为CGSize.shadowOpacity默认值为0.0，shadowRadius是为了实现阴影的模糊效果，使阴影更自然平滑,其默认值为3.0。不设置shadowRadius的效果如图20所示。

### 5.2.3 CALayer圆角

将5.2.1中的代码的9~11行改为：

1. //设置圆角
2. view .layer.cornerRadius = 40

运行项目，结果如图21所示。可以看到在层的四个角处绘制了指定半径的圆角。

把cornerRadius的值改为正方形视图宽的一半，效果如图22所示。可以看到视图变为正圆形。继续扩大cornerRadius的值为150，大于正方形宽的一半，效果如图23所示。

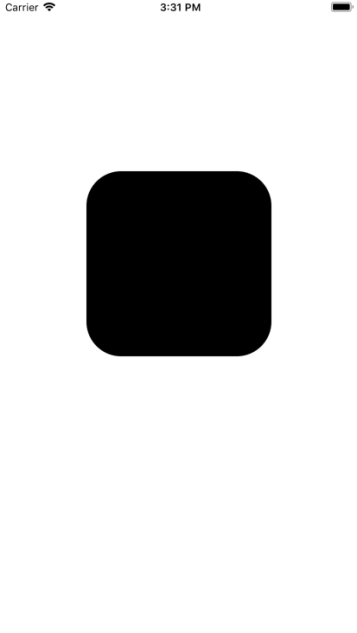
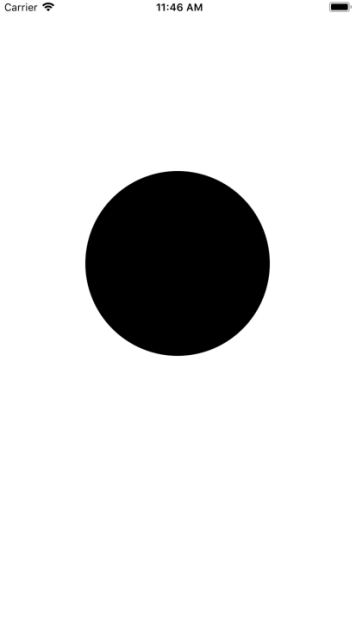
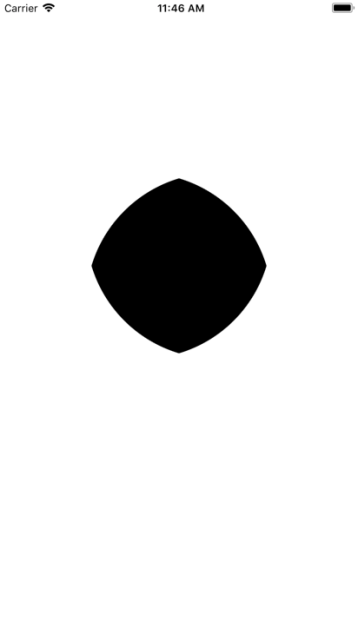
  

图21 图22 图23

### 5.2.4 颜色渐变

CALayer层也有嵌套和层级关系。通过对各层的绘图控制可以实现多种效果。CALayer层的颜色渐变是由其CAGradientLayer层控制的。通过该层可以实现多种颜色在某一视图区域的颜色渐变，然后将其添加到某CALayer层实现在iOS设备界面的显示。

CAGradientLayer层主要是通过在该层设置一个渐变线，然后设置在渐变线上的对应位置的颜色，相邻位置间就会产生颜色的过渡效果。

创建新项目，模板为iOS中的【Single View App】,创建完成后在ViewControl- -ler.swift中编写以下代码。

1. import UIKit
2. class ViewController: UIViewController {
3. override func viewDidLoad() {
4. super.viewDidLoad()
5. // Do any additional setup after loading the view, typically from a nib.
6. //创建第一个视图并添加到根视图中
7. //创建一个视图和一个渐变层，使渐变层位置和大小与视图一样大
8. let rect = CGRect(x: 50, y: 100, width: 214, height: 214)
9. let view = UIView(frame: rect)
10. let layer = CAGradientLayer()
11. layer.frame=view.frame
12. //设置渐变颜色数组
13. let firstColor = UIColor.white.cgColor
14. let secondColor = UIColor.red.cgColor
15. layer.colors = [firstColor,secondColor,firstColor]
16. //设置渐变线的起点和终点
17. layer.startPoint = CGPoint(x: 0, y: 0.5)
18. layer.endPoint = CGPoint(x: 1, y: 0.5)
19. //设置渐变层各颜色点在渐变线的位置分布
20. layer.locations = [0,0.5,1]
21. //添加视图和层到各自的根中
22. view.layer.addSublayer(layer)
23. self.view.addSubview(view)
24. }
25. }

运行项目，结果如图24所示。

注意，CALayer层的坐标与UIView中的坐标系不一样，其坐标系中x、y的取值范围为0~1，比如CGPoint(x: 0.5, y: 0.5)表示的是该CALayer层的中心点。渐变线的位置范围也是0~1。其次，在设置渐变层的locations数组时，其数组中元素与colors数组中元素是一一对应的。如果数组中元素个数超过colors数组的元素个数，那么默认locations数组剩余元素对应的的颜色为white；如果locations数组中元素个小于colors数组的元素个数，那么只显示对应到的colors数组中的颜色。



图24 颜色渐变