

iOS开发系列--让你的应用“动”起来

--iOS核心动画

概览

在iOS中随处都可以看到绚丽的动画效果，实现这些动画的过程并不复杂，今天将带大家一窥iOS动画全貌。在这里你可以看到iOS中如何使用图层精简非交互式绘图，如何通过核心动画创建基础动画、关键帧动画、动画组、转场动画，如何通过UIView的装饰方法对这些动画操作进行简化等。在今天的文章里您可以看到动画操作在iOS中是如何简单和高效，很多原来想做但是苦于没有思路的动画在iOS中将变得越发简单：

CALayer

CALayer简介

CALayer常用属性

CALayer绘图

Core Animation

基础动画

关键帧动画

动画组

转场动画

逐帧动画

[UIView动画封装](#)

基础动画

关键帧动画

转场动画

目录

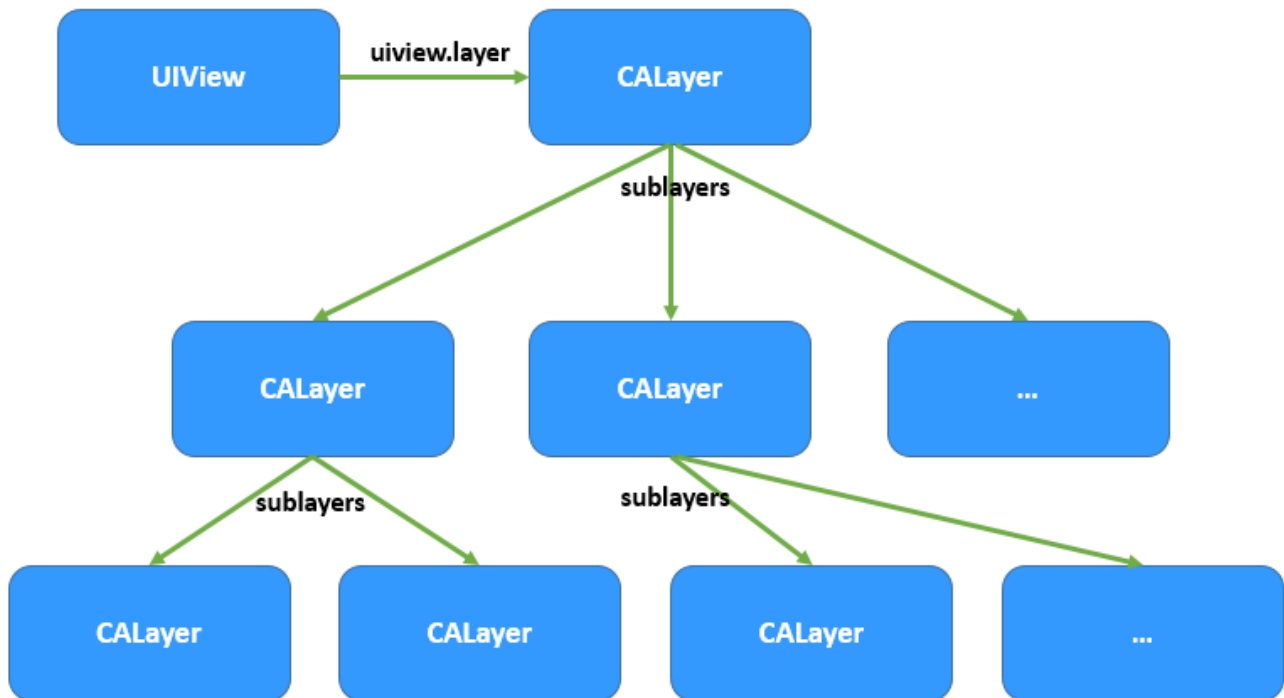
CALayer

CALayer简介

在介绍动画操作之前我们必须先来了解一个动画中常用的对象CALayer。CALayer包含在QuartzCore框架中，这是一个跨平台的框架，既可以用在iOS中又可以用在Mac OS X中。在使用Core Animation开发动画的本质就是将CALayer中的内容转化为位图从而供硬件操作，所以要熟练掌握动画操作必须先来熟悉CALayer。

在上一篇文章中使用Quartz 2D绘图时大家其实已经用到了CALayer，当利用drawRect:方法绘图的本质就是绘制到了UIView的layer（属性）中，可是这个过程大家在上一节中根本

体会不到。但是在Core Animation中我们操作更多的则不再是UIView而是直接面对CALayer。下图描绘了CALayer和UIView的关系，在UIView中有一个layer属性作为根图层，根图层上可以放其他子图层，在UIView中所有能够看到的内容都包含在layer中：



CALayer常用属性

在iOS中CALayer的设计主要是为了内容展示和动画操作，CALayer本身并不包含在UIKit中，它不能响应事件。由于CALayer在设计之初就考虑它的动画操作功能，CALayer很多属性在修改时都能形成动画效果，这种属性称为“隐式动画属性”。但是对于UIView的根图层而言属性的修改并不形成动画效果，因为很多情况下根图层更多的充当容器的做用，如果它的属性变动形成动画效果会直接影响子图层。另外，UIView的根图层创建工作完全由iOS负责完成，无法重新创建，但是可以往根图层中添加子图层或移除子图层。

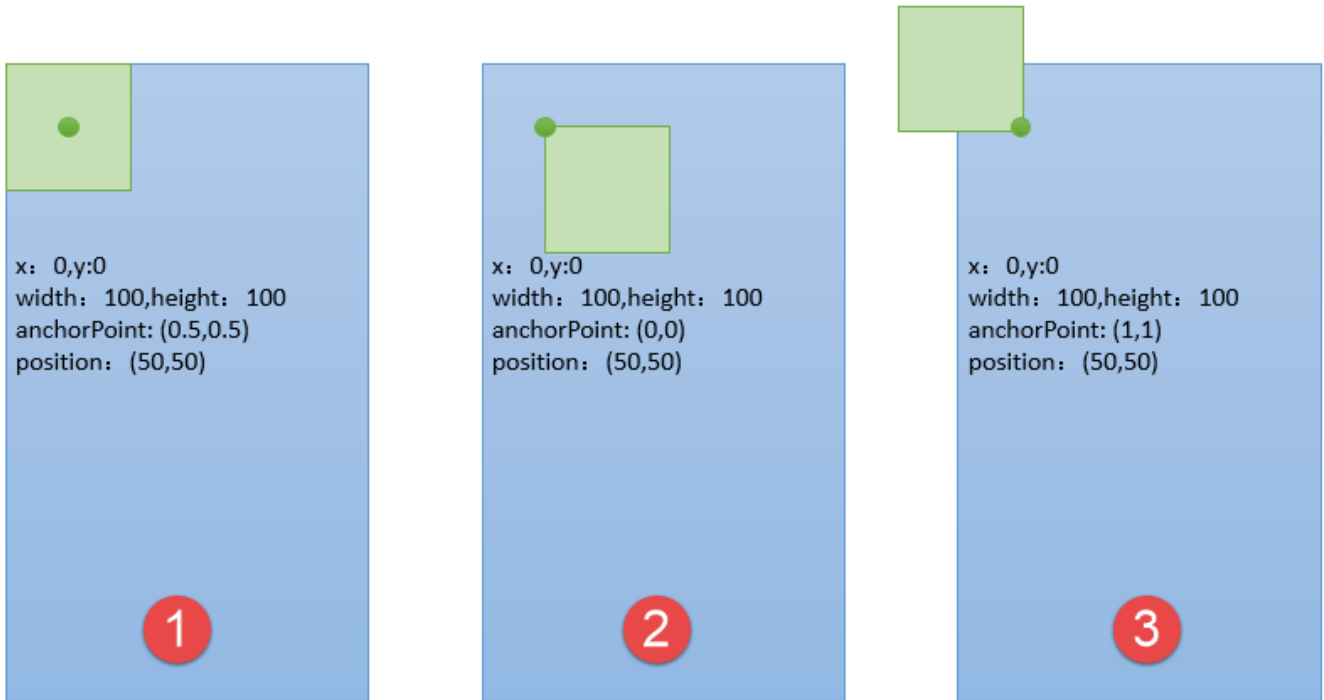
下表列出了CALayer常用的属性：

属性	说明
anchorPoint	和中心点position重合的一个点，称为“锚点”，锚点的描述是相对于x、y位置默认在图像中心点(0.5,0.5)的位置
backgroundColor	图层背景颜色
borderColor	边框颜色
borderWidth	边框宽度
bounds	图层大小
contents	图层显示内容，例如可以将图片作为图层内容显示
contentsRect	图层显示内容的大小和位置

cornerRadius	圆角半径
doubleSided	图层背面是否显示，默认为YES
frame	图层大小和位置，不支持隐式动画，所以CALayer中很少使用frame，通常使用position代替
hidden	是否隐藏
mask	图层蒙版
maskToBounds	子图层是否剪切图层边界，默认为NO
opacity	透明度，类似于UIView的alpha
position	图层中心点位置，类似于UIView的center
shadowColor	阴影颜色
shadowOffset	阴影偏移量
shadowOpacity	阴影透明度，注意默认为0，如果设置阴影必须设置此属性
shadowPath	阴影的形状
shadowRadius	阴影模糊半径
sublayers	子图层
sublayerTransform	子图层形变
transform	图层形变

- 隐式属性动画的本质是这些属性的变动默认隐含了CABasicAnimation动画实现，详情大家可以参照Xcode帮助文档中“Animatable Properties”一节。
- 在CALayer中很少使用frame属性，因为frame本身不支持动画效果，通常使用bounds和position代替。
- CALayer中透明度使用opacity表示而不是alpha；中心点使用position表示而不是center。
- anchorPoint属性是图层的锚点，范围在（0~1,0~1）表示在x、y轴的比例，这个点永远可以同position（中心点）重合，当图层中心点固定后，调整anchorPoint即可达到调整图层显示位置的作用（因为它永远和position重合）

为了进一步说明anchorPoint的作用，假设有一个层大小100*100，现在中心点位置（50,50），由此可以得出frame（0,0,100,100）。上面说过anchorPoint默认为（0.5,0.5），同中心点position重合，此时使用图形描述如图1；当修改anchorPoint为（0,0），此时锚点处于图层左上角，但是中心点position并不会改变，因此图层会向右下角移动，如图2；然后修改anchorPoint为（1,1），position还是保持位置不变，锚点处于图层右下角，此时图层如图3。



下面通过一个简单的例子演示一下上面几个属性，程序初始化阶段我们定义一个正方形，但是圆角路径调整为正方形边长的一般，使其看起来是一个圆形，在点击屏幕的时候修改图层的属性形成动画效果（注意在程序中没有直接修改UIView的layer属性，因为根图层无法形成动画效果）：

```
//  
// KCMainViewController.m  
// CALayer  
//  
// Created by Kenshin Cui on 14-3-22.  
// Copyright (c) 2014年 Kenshin Cui. All rights reserved.  
//  
  
#import "KCMainViewController.h"  
#define WIDTH 50  
  
@interface KCMainViewController ()  
  
@end  
  
@implementation KCMainViewController  
  
- (void)viewDidLoad {  
    [super viewDidLoad];  
    // Do any additional setup after loading the view.  
    [self drawMyLayer];  
}  
  
#pragma mark 绘制图层
```

```
-(void)drawMyLayer{
    CGSize size=[UIScreen mainScreen].bounds.size;

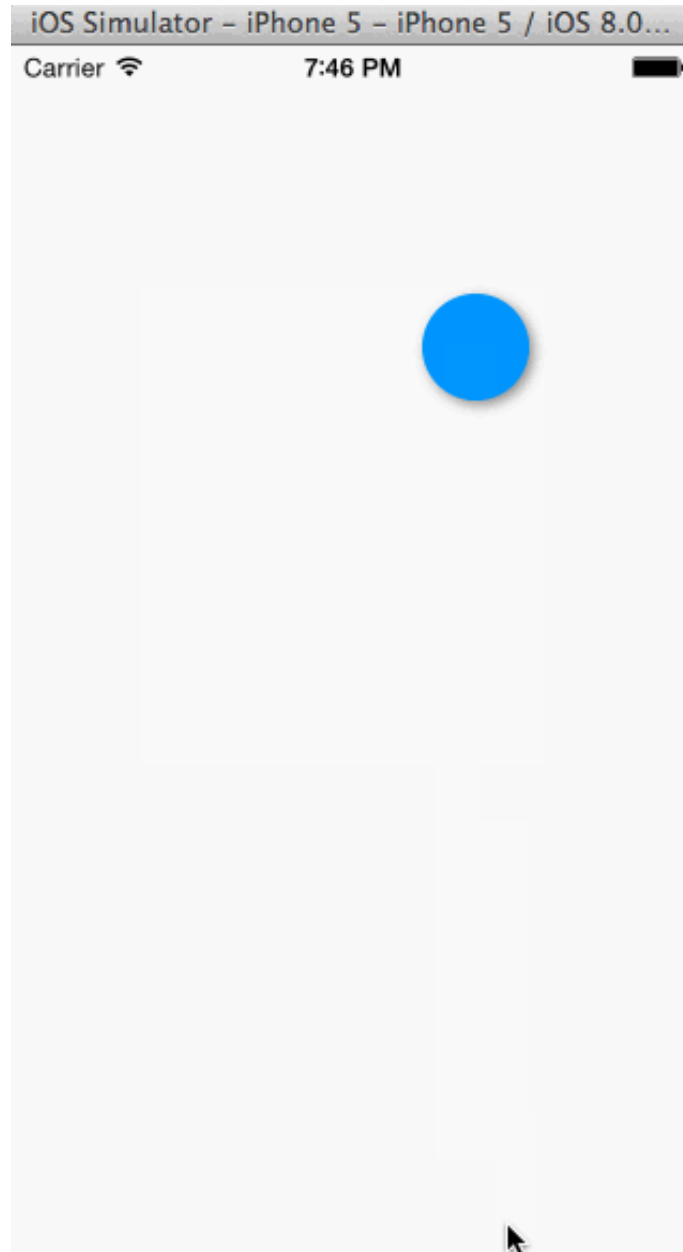
    //获得根图层
    CALayer *layer=[[CALayer alloc]init];
    //设置背景颜色,由于QuartzCore是跨平台框架,无法直接使用UIColor
    layer.backgroundColor=[UIColor colorWithRed:0 green:146/255.0 blue:1.0 alpha:1.0].CGColor;
    //设置中心点
    layer.position=CGPointMake(size.width/2, size.height/2);
    //设置大小
    layer.bounds=CGRectMake(0, 0, WIDTH,WIDTH);
    //设置圆角,当圆角半径等于矩形的一半时看起来就是一个圆形
    layer.cornerRadius=WIDTH/2;
    //设置阴影
    layer.shadowColor=[UIColor grayColor].CGColor;
    layer.shadowOffset=CGSizeMake(2, 2);
    layer.shadowOpacity=.9;
    //设置边框
    // layer.borderColor=[UIColor whiteColor].CGColor;
    // layer.borderWidth=1;

    //设置锚点
    // layer.anchorPoint=CGPointZero;

    [self.view.layer addSublayer:layer];
}

#pragma mark 点击放大
-(void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch *touch=[touches anyObject];
    CALayer *layer=self.view.layer.sublayers[0];
    CGFloat width=layer.bounds.size.width;
    if (width==WIDTH) {
        width=WIDTH*4;
    }else{
        width=WIDTH;
    }
    layer.bounds=CGRectMake(0, 0, width, width);
    layer.position=[touch locationInView:self.view];
    layer.cornerRadius=width/2;
}
@end
```

运行效果:



CALayer绘图

上一篇文章中重点讨论了使用Quartz 2D绘图，当时调用了UIView的drawRect:方法绘制图形、图像，这种方式本质还是在图层中绘制，但是这里会着重介绍一下如何直接在图层中绘图。在图层中绘图的方式跟原来基本没有区别，只是drawRect:方法是由UIKit组件进行调用，因此里面可以使用一些UIKit封装的方法进行绘图，而直接绘制到图层的方法由于并非UIKit直接调用因此只能用原生的Core Graphics方法绘制。

图层绘图有两种方法，不管使用哪种方法绘制完必须调用图层的setNeedsDisplay方法（注意是图层的方法，不是UIView的方法，前面我们介绍过UIView也有此方法）

1. 通过图层代理**drawLayer: inContext:**方法绘制
2. 通过自定义图层**drawInContext:**方法绘制

使用代理方法绘图

通过代理方法进行图层绘图只要指定图层的代理，然后在代理对象中重写-

(void)drawLayer:(CALayer *)layer inContext:(CGContextRef)ctx方法即可。需要注意这个方法虽然是代理方法但是不用手动实现CALayerDelegate，因为CALayer定义中给NSObject做了分类扩展，所有的NSObject都包含这个方法。另外设置完代理后必须要调用图层的setNeedsDisplay方法，否则绘制的内容无法显示。

下面的代码演示了在一个自定义图层绘制一张图像并将图像设置成圆形，这种效果在很多应用中很常见，如最新版的手机QQ头像就是这种效果：

```
//
//  KCMainViewController.m
//  CALayer
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"
#define PHOTO_HEIGHT 150

@interface KCMainViewController ()

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //自定义图层
    CALayer *layer=[[CALayer alloc]init];
    layer.bounds=CGRectMake(0, 0, PHOTO_HEIGHT, PHOTO_HEIGHT);
    layer.position=CGPointMake(160, 200);
    layer.backgroundColor=[UIColor redColor].CGColor;
    layer.cornerRadius=PHOTO_HEIGHT/2;
    //注意仅仅设置圆角，对于图形而言可以正常显示，但是对于图层中绘制的图片无法正确显示
    //如果想要正确显示则必须设置masksToBounds=YES，剪切子图层
    layer.masksToBounds=YES;
    //阴影效果无法和masksToBounds同时使用，因为masksToBounds的目的就是剪切外边框，
    //而阴影效果刚好在外边框
    //    layer.shadowColor=[UIColor grayColor].CGColor;
    //    layer.shadowOffset=CGSizeMake(2, 2);
    //    layer.shadowOpacity=1;
    //设置边框
    layer.borderColor=[UIColor whiteColor].CGColor;
```

```
layer.borderWidth=2;

//设置图层代理
layer.delegate=self;

//添加图层到根图层
[self.view.layer addSublayer:layer];

//调用图层setNeedDisplay,否则代理方法不会被调用
[layer setNeedsDisplay];
}

#pragma mark 绘制图形、图像到图层, 注意参数中的ctx是图层的图形上下文, 其中绘图位置也是相对图层而言的
-(void)drawLayer:(CALayer *)layer inContext:(CGContextRef)ctx{
//    NSLog(@"%@", layer); //这个图层正是上面定义的图层
    CGContextSaveGState(ctx);

    //图形上下文形变, 解决图片倒立的问题
    CGContextScaleCTM(ctx, 1, -1);
    CGContextTranslateCTM(ctx, 0, -PHOTO_HEIGHT);

    UIImage *image=[UIImage imageNamed:@"photo.png"];
    //注意这个位置是相对于图层而言的不是屏幕
    CGContextDrawImage(ctx, CGRectMake(0, 0, PHOTO_HEIGHT, PHOTO_HEIGHT), image.CGImage)

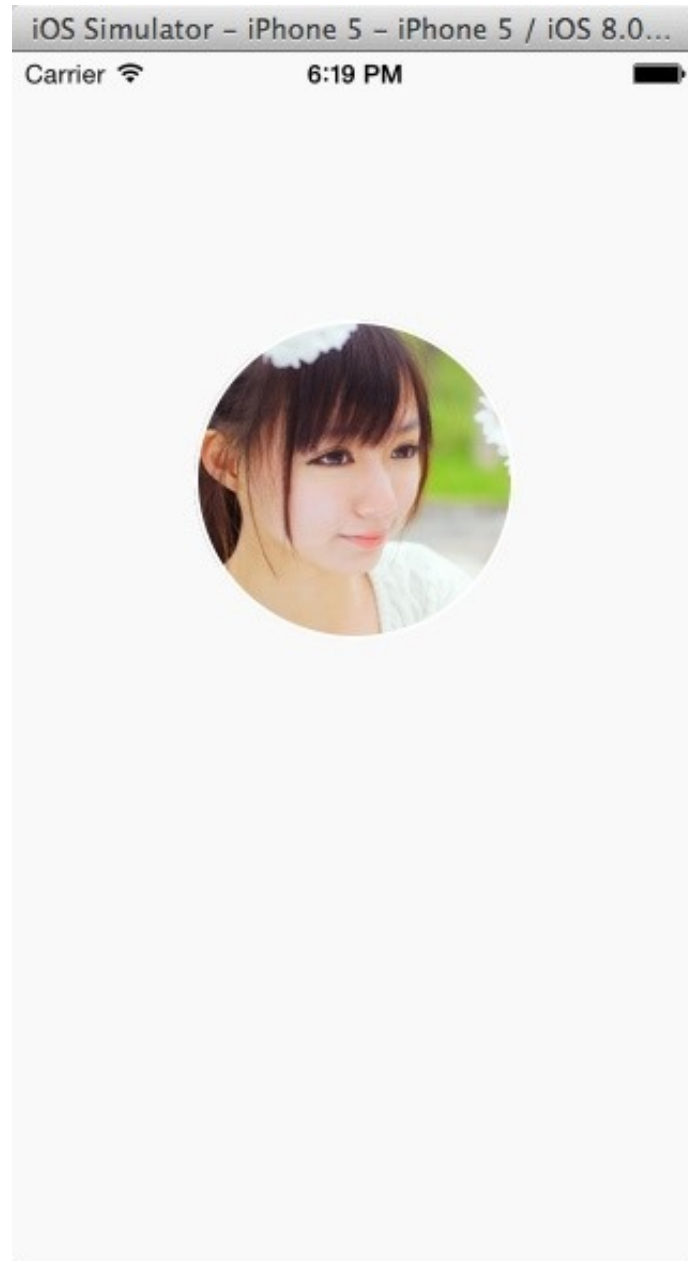
//    CGContextFillRect(ctx, CGRectMake(0, 0, 100, 100));
//    CGContextDrawPath(ctx, kCGPathFillStroke);

    CGContextRestoreGState(ctx);
}

@end
```



运行效果：



使用代理方法绘制图形、图像时在**drawLayer:inContext:**方法中可以通过事件参数获得绘制的图层和图形上下文。在这个方法中绘图时所有的位置都是相对于图层而言的，图形上下文指的也是当前图层的图形上下文。

需要注意的是上面代码中绘制图片圆形裁切效果时如果不设置**masksToBounds**是无法显示圆形，但是对于其他图形却没有这个限制。原因就是当绘制一张图片到图层上的时候会重新创建一个图层添加到当前图层，这样一来如果设置了圆角之后虽然底图层有圆角效果，但是子图层还是矩形，只有设置了**masksToBounds**为**YES**让子图层按底图层剪切才能显示圆角效果。同样的，有些朋友经常在网上提问说为什么使用**UIImageView**的**layer**设置圆角后图片无法显示圆角，只有设置**masksToBounds**才能出现效果，也是类似的问题。

扩展1--带阴影效果的圆形图片裁切

如果设置了masksToBounds=YES之后确实可以显示图片圆角效果，但遗憾的是设置了这个属性之后就无法设置阴影效果。因为masksToBounds=YES就意味着外边框不能显示，而阴影恰恰作为外边框绘制的，这样两个设置就产生了矛盾。要解决这个问题不妨换个思路：使用两个大小一样的图层，下面的图层负责绘制阴影，上面的图层用来显示图片。

```
//
//  KCMainViewController.m
//  CALayer
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"
#define PHOTO_HEIGHT 150

@interface KCMainViewController ()

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    CGPoint position= CGPointMake(160, 200);
    CGRect bounds=CGRectMake(0, 0, PHOTO_HEIGHT, PHOTO_HEIGHT);
    CGFloat cornerRadius=PHOTO_HEIGHT/2;
    CGFloat borderWidth=2;

    //阴影图层
    CALayer *layerShadow=[[CALayer alloc]init];
    layerShadow.bounds=bounds;
    layerShadow.position=position;
    layerShadow.cornerRadius=cornerRadius;
    layerShadow.shadowColor=[UIColor grayColor].CGColor;
    layerShadow.shadowOffset=CGSizeMake(2, 1);
    layerShadow.shadowOpacity=1;
    layerShadow.borderColor=[UIColor whiteColor].CGColor;
    layerShadow.borderWidth=borderWidth;
    [self.view.layer addSublayer:layerShadow];

    //容器图层
    CALayer *layer=[[CALayer alloc]init];
    layer.bounds=bounds;
    layer.position=position;
    layer.backgroundColor=[UIColor redColor].CGColor;
    layer.cornerRadius=cornerRadius;
    layer.masksToBounds=YES;
```

```
layer.borderColor=[UIColor whiteColor].CGColor;
layer.borderWidth=borderWidth;

//设置图层代理
layer.delegate=self;

//添加图层到根图层
[self.view.layer addSublayer:layer];

//调用图层setNeedDisplay,否则代理方法不会被调用
[layer setNeedsDisplay];
}

#pragma mark 绘制图形、图像到图层,注意参数中的ctx是图层的图形上下文,其中绘图位置也是相对图层而言的
-(void)drawLayer:(CALayer *)layer inContext:(CGContextRef)ctx{
    //    NSLog(@"%@",layer);//这个图层正是上面定义的图层
    CGContextSaveGState(ctx);

    //图形上下文形变,解决图片倒立的问题
    CGContextScaleCTM(ctx, 1, -1);
    CGContextTranslateCTM(ctx, 0, -PHOTO_HEIGHT);

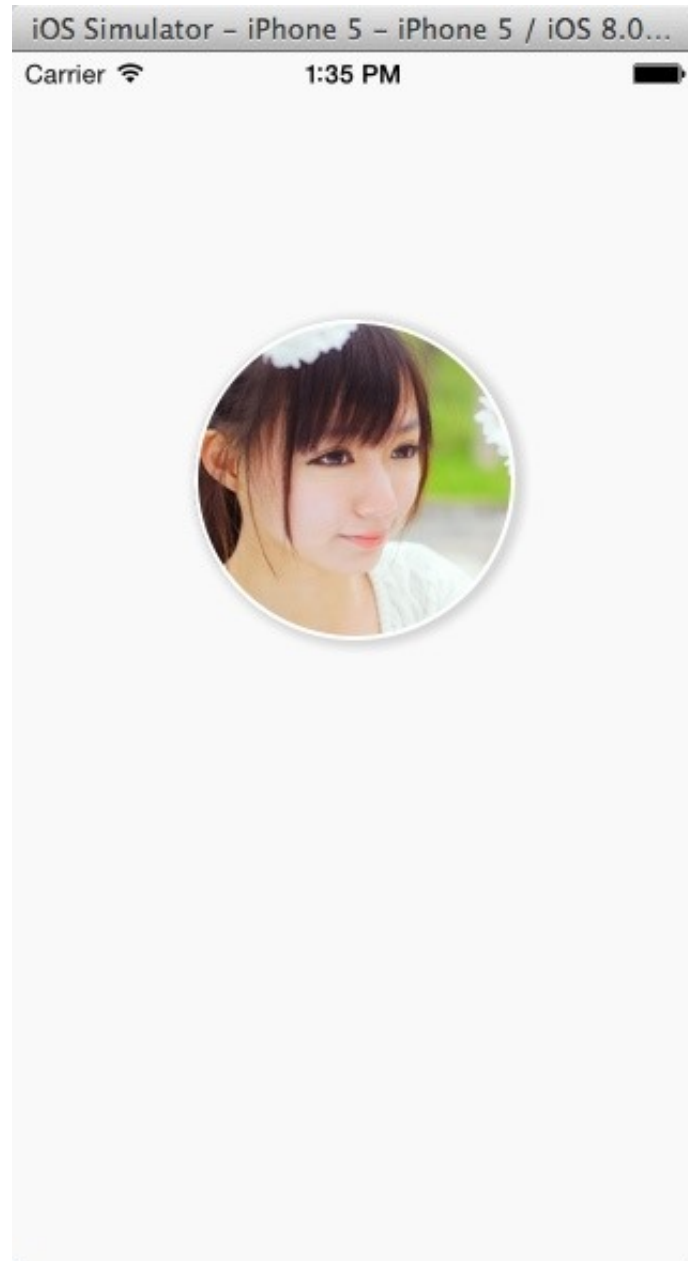
    UIImage *image=[UIImage imageNamed:@"photo.jpg"];
    //注意这个位置是相对于图层而言的不是屏幕
    CGContextDrawImage(ctx, CGRectMake(0, 0, PHOTO_HEIGHT, PHOTO_HEIGHT), image.CGImage)

    //    CGContextFillRect(ctx, CGRectMake(0, 0, 100, 100));
    //    CGContextDrawPath(ctx, kCGPathFillStroke);

    CGContextRestoreGState(ctx);
}
@end
```

运行效果:





扩展2--图层的形变

从上面代码中大家不难发现使用Core Graphics绘制图片时会倒立显示，对图层的图形上下文进行了反转。在前一篇文章中也采用了类似的方法去解决这个问题，但是在那篇文章中也提到过如果直接让图像沿着x轴旋转180度同样可以达到正确显示的目的，只是当时的旋转靠图形上下文还无法绕x轴旋转。今天学习了图层之后，其实可以控制图层直接旋转而不用借助于图形上下文的形变操作，而且这么操作起来会更加简单和直观。对于上面的程序，只需要设置图层的transform属性即可。需要注意的是transform是CATransform3D类型，形变可以在三个维度上进行，使用方法和前面介绍的二维形变是类似的，而且都有对应的形变设置方法（如：CATransform3DMakeTranslation()、CATransform3DMakeScale()、CATransform3DMakeRotation()）。下面的代码通过

CATransform3DMakeRotation()方法在x轴旋转180度解决倒立问题:

```
//
// 形变演示
// CALayer
//
// Created by Kenshin Cui on 14-3-22.
// Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"
#define PHOTO_HEIGHT 150

@interface KCMainViewController ()

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    CGPoint position= CGPointMake(160, 200);
    CGRect bounds=CGRectMake(0, 0, PHOTO_HEIGHT, PHOTO_HEIGHT);
    CGFloat cornerRadius=PHOTO_HEIGHT/2;
    CGFloat borderWidth=2;

    //阴影图层
    CALayer *layerShadow=[[CALayer alloc]init];
    layerShadow.bounds=bounds;
    layerShadow.position=position;
    layerShadow.cornerRadius=cornerRadius;
    layerShadow.shadowColor=[UIColor grayColor].CGColor;
    layerShadow.shadowOffset=CGSizeMake(2, 1);
    layerShadow.shadowOpacity=1;
    layerShadow.borderColor=[UIColor whiteColor].CGColor;
    layerShadow.borderWidth=borderWidth;
    [self.view.layer addSublayer:layerShadow];

    //容器图层
    CALayer *layer=[[CALayer alloc]init];
    layer.bounds=bounds;
    layer.position=position;
    layer.backgroundColor=[UIColor redColor].CGColor;
    layer.cornerRadius=cornerRadius;
    layer.masksToBounds=YES;
    layer.borderColor=[UIColor whiteColor].CGColor;
    layer.borderWidth=borderWidth;

    //利用图层形变解决图像倒立问题
```

```
layer.transform=CATransform3DMakeRotation(M_PI, 1, 0, 0);

//设置图层代理
layer.delegate=self;

//添加图层到根图层
[self.view.layer addSublayer:layer];

//调用图层setNeedDisplay,否则代理方法不会被调用
[layer setNeedsDisplay];
}

#pragma mark 绘制图形、图像到图层, 注意参数中的ctx时图层的图形上下文, 其中绘图位置也是相对图层而言的
-(void)drawLayer:(CALayer *)layer inContext:(CGContextRef)ctx{
    // NSLog(@"%@", layer); //这个图层正是上面定义的图层
    UIImage *image=[UIImage imageNamed:@"photo.jpg"];
    //注意这个位置是相对于图层而言的不是屏幕
    CGContextDrawImage(ctx, CGRectMake(0, 0, PHOTO_HEIGHT, PHOTO_HEIGHT), image.CGImage)
}

@end
```

事实上如果仅仅就显示一张图片在图层中当然没有必要那么麻烦, 直接设置图层contents就可以了, 不牵涉到绘图也就没有倒立的问题了。

```
//
// 图层内容设置
// CALayer
//
// Created by Kenshin Cui on 14-3-22.
// Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"
#define PHOTO_HEIGHT 150

@interface KCMainViewController ()

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    CGPoint position= CGPointMake(160, 200);
    CGRect bounds=CGRectMake(0, 0, PHOTO_HEIGHT, PHOTO_HEIGHT);
    CGFloat cornerRadius=PHOTO_HEIGHT/2;
```

```
CGFloat borderWidth=2;

//阴影图层
CALayer *layerShadow=[[CALayer alloc]init];
layerShadow.bounds=bounds;
layerShadow.position=position;
layerShadow.cornerRadius=cornerRadius;
layerShadow.shadowColor=[UIColor grayColor].CGColor;
layerShadow.shadowOffset=CGSizeMake(2, 1);
layerShadow.shadowOpacity=1;
layerShadow.borderColor=[UIColor whiteColor].CGColor;
layerShadow.borderWidth=borderWidth;
[self.view.layer addSublayer:layerShadow];

//容器图层
CALayer *layer=[[CALayer alloc]init];
layer.bounds=bounds;
layer.position=position;
layer.backgroundColor=[UIColor redColor].CGColor;
layer.cornerRadius=cornerRadius;
layer.masksToBounds=YES;
layer.borderColor=[UIColor whiteColor].CGColor;
layer.borderWidth=borderWidth;
//设置内容（注意这里一定要转换为CGImage）
UIImage *image=[UIImage imageNamed:@"photo.jpg"];
// layer.contents=(id)image.CGImage;
[layer setContents:(id)image.CGImage];

//添加图层到根图层
[self.view.layer addSublayer:layer];
}

@end
```

既然如此为什么还大费周章的说形变呢，因为形变对于动画有特殊的意义。在动画开发中形变往往不是直接设置transform，而是通过keyPath进行设置。这种方法设置形变的本质和前面没有区别，只是利用了KVC可以动态修改其属性值而已，但是这种方式在动画中确实很常用的，因为它可以很方便的将几种形变组合到一起使用。同样是解决动画旋转问题，只要将前面的旋转代码改为下面的代码即可：

```
[layer setValue:@M_PI forKeyPath:@"transform.rotation.x"];
```

当然，通过key path设置形变参数就需要了解有哪些key path可以设置，这里就不再一一列举，大家可以参照Xcode帮助文档中“CATransform3D Key Paths”一节，里面描述的很

详细。

使用自定义图层绘图

在自定义图层中绘图时只要自己编写一个类继承于CALayer然后在**drawInContext:**中绘图即可。同前面在代理方法绘图一样，要显示图层中绘制的内容也要调用图层的**setNeedDisplay**方法，否则**drawInContext**方法将不会调用。

前面的文章中曾经说过，在使用Quartz 2D在UIView中绘制图形的本质也是绘制到图层中，为了说明这个问题下面演示自定义图层绘图时没有直接在视图控制器中调用自定义图层，而是在一个UIView将自定义图层添加到UIView的根图层中（例子中的UIView跟自定义图层绘图没有直接关系）。从下面的代码中可以看到：UIView在显示时其根图层会自动创建一个CGContextRef（CALayer本质使用的是位图上下文），同时调用图层代理（UIView创建图层会自动设置图层代理为其自身）的**draw: inContext:**方法并将图形上下文作为参数传递给这个方法。而在UIView的**draw:inContext:**方法中会调用其**drawRect:**方法，在**drawRect:**方法中使用**UIGraphicsGetCurrentContext()**方法得到的上下文正是前面创建的上下文。

KCLayer.m

```
//
//  KCLayer.m
//  CALayer
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCLayer.h"

@implementation KCLayer

-(void)drawInContext:(CGContextRef)ctx{
    NSLog(@"3-drawInContext:");
    NSLog(@"CGContext:%@",ctx);
    //    CGContextRotateCTM(ctx, M_PI_4);
    CGContextSetRGBFillColor(ctx, 135.0/255.0, 232.0/255.0, 84.0/255.0, 1);
    CGContextSetRGBStrokeColor(ctx, 135.0/255.0, 232.0/255.0, 84.0/255.0, 1);
    //    CGContextFillRect(ctx, CGRectMake(0, 0, 100, 100));
    //    CGContextFillEllipseInRect(ctx, CGRectMake(50, 50, 100, 100));
    CGContextMoveToPoint(ctx, 94.5, 33.5);

    //// Star Drawing
    CGContextAddLineToPoint(ctx,104.02, 47.39);
    CGContextAddLineToPoint(ctx,120.18, 52.16);
```



```
CGContextAddLineToPoint(ctx,109.91, 65.51);
CGContextAddLineToPoint(ctx,110.37, 82.34);
CGContextAddLineToPoint(ctx,94.5, 76.7);
CGContextAddLineToPoint(ctx,78.63, 82.34);
CGContextAddLineToPoint(ctx,79.09, 65.51);
CGContextAddLineToPoint(ctx,68.82, 52.16);
CGContextAddLineToPoint(ctx,84.98, 47.39);
CGContextClosePath(ctx);

CGContextDrawPath(ctx, kCGPathFillStroke);
}

@end
```

KCView.m

```
//
//  KCView.m
//  CALayer
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCView.h"
#import "KCLayer.h"

@implementation KCView

-(instancetype)initWithFrame:(CGRect)frame{
    NSLog(@"initWithFrame:");
    if (self=[super initWithFrame:frame]) {
        KCLayer *layer=[[KCLayer alloc]init];
        layer.bounds=CGRectMake(0, 0, 185, 185);
        layer.position=CGPointMake(160,284);
        layer.backgroundColor=[UIColor colorWithRed:0 green:146/255.0 blue:1.0 alpha:1.0];

        //显示图层
        [layer setNeedsDisplay];


        [self.layer addSublayer:layer];
    }
    return self;
}

-(void)drawRect:(CGRect)rect{
    NSLog(@"2-drawRect:");
}
```

```
        NSLog(@"CGContext:%@", UIGraphicsGetCurrentContext()); //得到的当前图形上下文正是drawLayer
        [super drawRect:rect];
    }

    -(void)drawLayer:(CALayer *)layer inContext:(CGContextRef)ctx{
        NSLog(@"1-drawLayer:inContext:");
        NSLog(@"CGContext:%@", ctx);
        [super drawLayer:layer inContext:ctx];
    }

@end
```



KCMainViewController.m

```
//
//  KCMainViewController.m
//  CALayer
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"
#import "KCView.h"

@interface KCMainViewController ()

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

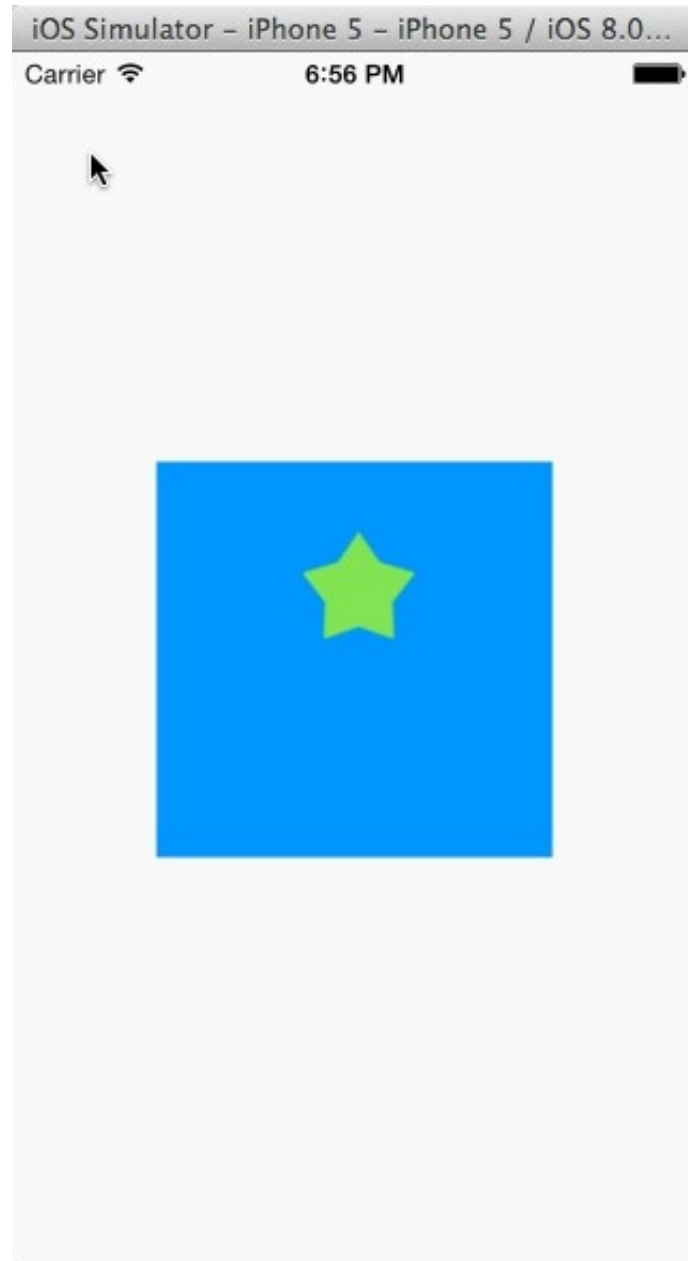
    KCView *view=[[KCView alloc] initWithFrame:[UIScreen mainScreen].bounds];
    view.backgroundColor=[UIColor colorWithRed:249.0/255.0 green:249.0/255.0 blue:249.0/

    [self.view addSubview:view];
}

@end
```



运行效果：



Core Animation

大家都知道在iOS中实现一个动画相当简单，只要调用UIView的块代码即可实现一个动画效果，这在其他系统开发中基本不可能实现。下面通过一个简单的UIView进行一个图片放大动画效果演示：

```
//  
//  KCMainViewController.m  
//  Animation  
//  
//  Created by Kenshin Cui on 14-3-22.  
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.  
//  
  
#import "KCMainViewController.h"
```

```
@interface KCMainViewController ()

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

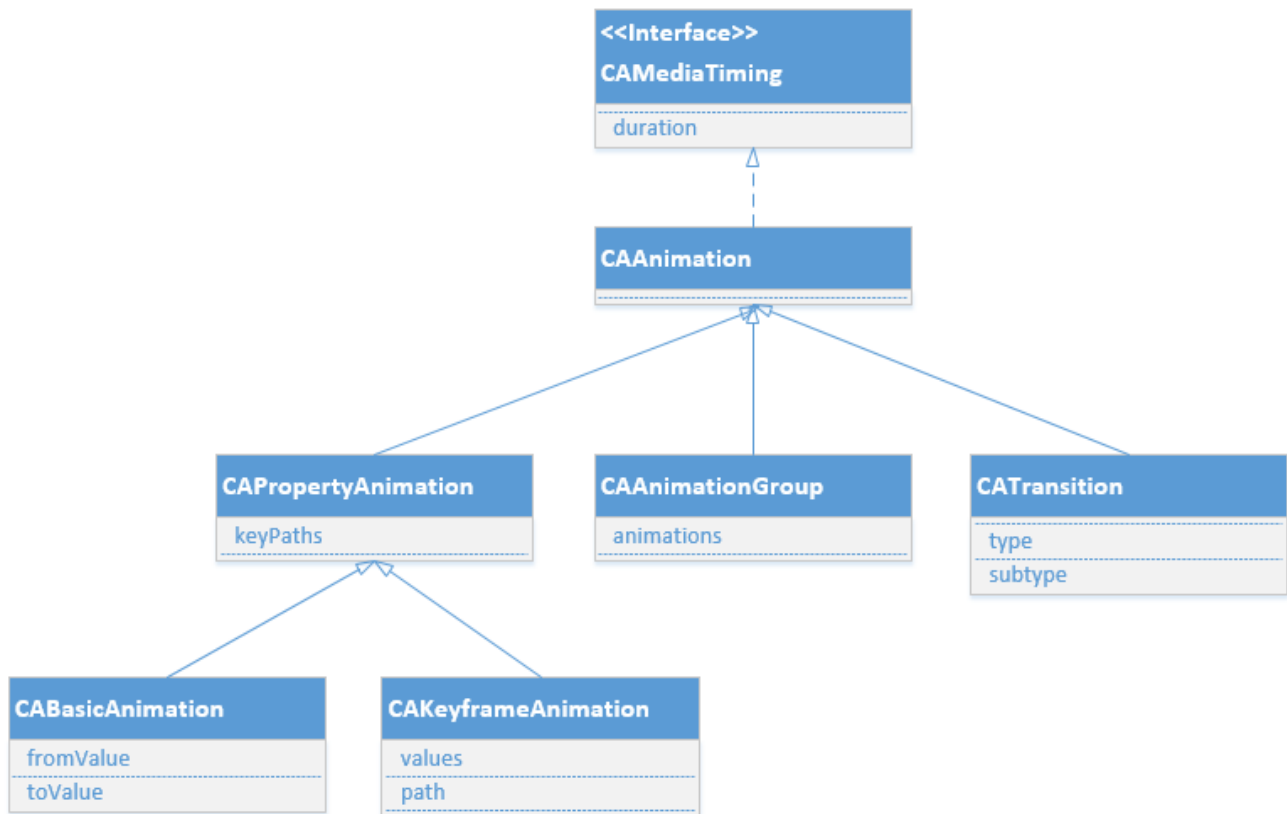
    UIImage *image=[UIImage imageNamed:@"open2.png"];
    UIImageView *imageView=[[UIImageView alloc] init];
    imageView.image=image;
    imageView.frame=CGRectMake(120, 140, 80, 80);
    [self.view addSubview:imageView];

    //两秒后开始一个持续一分钟的动画
    [UIView animateWithDuration:1 delay:2 options:UIViewAnimationOptionBeginFromCurrentState
        animations:^{
            imageView.frame=CGRectMake(80, 100, 160, 160);
        } completion:nil];
}

@end
```

使用上面UIView封装的方法进行动画设置固然十分方便，但是具体动画如何实现我们是不清楚的，而且上面的代码还有一些问题是无法解决的，例如：如何控制动画的暂停？如何进行动画的组合？。。。

这里就需要了解iOS的核心动画Core Animation（包含在Quartz Core框架中）。在iOS中核心动画分为几类：基础动画、关键帧动画、动画组、转场动画。各个类的关系大致如下：



CAAnimation：核心动画的基础类，不能直接使用，负责动画运行时间、速度的控制，本身实现了CAMediaTiming协议。

CAPropertyAnimation：属性动画的基类（通过属性进行动画设置，注意是可动画属性），不能直接使用。

CAAnimationGroup：动画组，动画组是一种组合模式设计，可以通过动画组来进行所有动画行为的统一控制，组中所有动画效果可以并发执行。

CATransition：转场动画，主要通过滤镜进行动画效果设置。

CABasicAnimation：基础动画，通过属性修改进行动画参数控制，只有初始状态和结束状态。

CAKeyframeAnimation：关键帧动画，同样是通过属性进行动画参数控制，但是同基础动画不同的是它可以有多个状态控制。

基础动画、关键帧动画都属于属性动画，就是通过修改属性值产生动画效果，开发人员只需要设置初始值和结束值，中间的过程动画（又叫“补间动画”）由系统自动计算产生。和基础动画不同的是关键帧动画可以设置多个属性值，每两个属性中间的补间动画由系统自动完成，因此从这个角度而言基础动画又可以看成是有两个关键帧的关键帧动画。

基础动画

在开发过程中很多情况下通过基础动画就可以满足开发需求，前面例子中使用的UIView代码块进行图像放大缩小的演示动画也是基础动画（在iOS7中UIView也对关键帧动画进行了封装），只是UIView装饰方法隐藏了更多的细节。如果不使用UIView封装的方法，动画创建一般分为以下几步：

- 1.初始化动画并设置动画属性
- 2.设置动画属性初始值（可以省略）、结束值以及其他动画属性
- 3.给图层添加动画

下面以一个移动动画为例进行演示，在这个例子中点击屏幕哪个位置落花将飞向哪里。

```
//
//  KCMainViewController.m
//  Animation
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"

@interface KCMainViewController () {
    CALayer *_layer;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景(注意这个图片其实在根图层)
    UIImage *backgroundImage=[UIImage imageNamed:@"background.jpg"];
    self.view.backgroundColor=[UIColor colorWithPatternImage:backgroundImage];

    //自定义一个图层
    _layer=[[CALayer alloc]init];
    _layer.bounds=CGRectMake(0, 0, 10, 20);
    _layer.position=CGPointMake(50, 150);
    _layer.contents=(id)[UIImage imageNamed:@"petal.png"].CGImage;
    [self.view.layer addSublayer:_layer];
}
```

```
#pragma mark 移动动画
-(void)translatonAnimation:(CGPoint)location{
    //1.创建动画并指定动画属性
    CABasicAnimation *basicAnimation=[CABasicAnimation animationWithKeyPath:@"position"]

    //2.设置动画属性初始值和结束值
    //    basicAnimation.fromValue=[NSNumber numberWithInt:50];//可以不设置，默认为图层初始值
    basicAnimation.toValue=[NSValue valueWithCGPoint:location];

    //设置其他动画属性
    basicAnimation.duration=5.0;//动画时间5秒
    //basicAnimation.repeatCount=HUGE_VALF;//设置重复次数,HUGE_VALF可看做无穷大，起到循环动画的作用
    //    basicAnimation.removedOnCompletion=NO;//运行一次是否移除动画

    //3.添加动画到图层，注意key相当于给动画进行命名，以后获得该动画时可以使用此名称获取
    [_layer addAnimation:basicAnimation forKey:@"KCBasicAnimation_Translation"];
}

#pragma mark 点击事件
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch *touch=touches.anyObject;
    CGPoint location= [touch locationInView:self.view];
    //创建并开始动画
    [self translatonAnimation:location];
}

@end
```



运行效果：



上面实现了一个基本动画效果，但是这个动画存在一个问题：动画结束后动画图层回到了原来的位置，当然是用UIView封装的方法是没有这个问题的。如何解决这个问题呢？

前面说过图层动画的本质就是将图层内部的内容转化为位图经硬件操作形成一种动画效果，其实图层本身并没有任何的变化。上面的动画中图层并没有因为动画效果而改变它的位置（对于缩放动画其大小也是不会改变的），所以动画完成之后图层还是在原来的显示位置没有任何变化，如果这个图层在一个UIView中你会发现在UIView移动过程中你要触发UIView的点击事件也只能点击原来的位置（即使它已经运动到了别的位置），因为它的位置从来没有变过。当然解决这个问题方法比较多，这里不妨在动画完成之后重新设置它的位置。

```
//  
// KCMainViewController.m
```



```
// Animation
//
// Created by Kenshin Cui on 14-3-22.
// Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"

@interface KCMainViewController () {
    CALayer *_layer;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景(注意这个图片其实在根图层)
    UIImage *backgroundImage=[UIImage imageNamed:@"background.jpg"];
    self.view.backgroundColor=[UIColor colorWithPatternImage:backgroundImage];

    //自定义一个图层
    _layer=[[CALayer alloc] init];
    _layer.bounds=CGRectMake(0, 0, 10, 20);
    _layer.position=CGPointMake(50, 150);
    _layer.contents=(id)[UIImage imageNamed:@"petal.png"].CGImage;
    [self.view.layer addSublayer:_layer];
}

#pragma mark 移动动画
-(void)translatonAnimation:(CGPoint)location{
    //1.创建动画并指定动画属性
    CABasicAnimation *basicAnimation=[CABasicAnimation animationWithKeyPath:@"position"]

    //2.设置动画属性初始值和结束值
    // basicAnimation.fromValue=[NSNumber numberWithInt:50];//可以不设置，默认为图层初始值
    basicAnimation.toValue=[NSValue valueWithCGPoint:location];

    //设置其他动画属性
    basicAnimation.duration=5.0;//动画时间5秒
    //basicAnimation.repeatCount=HUGE_VALF;//设置重复次数,HUGE_VALF可看做无穷大，起到循环动画的作用
    // basicAnimation.removedOnCompletion=NO;//运行一次是否移除动画
    basicAnimation.delegate=self;
    //存储当前位置在动画结束后使用
    [basicAnimation setValue:[NSValue valueWithCGPoint:location] forKey:@"KCBasicAnimation"]
}
```

```

//3.添加动画到图层，注意key相当于给动画进行命名，以后获得该动画时可以使用此名称获取
[_layer addAnimation:basicAnimation forKey:@"KCBasicAnimation_Translation"];
}

#pragma mark 点击事件
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch *touch=touches.anyObject;
    CGPoint location= [touch locationInView:self.view];
    //创建并开始动画
    [self translatonAnimation:location];
}

#pragma mark - 动画代理方法
#pragma mark 动画开始
-(void)animationDidStart:(CAAnimation *)anim{
    NSLog(@"animation(%%) start.\r_layer.frame=%%",anim,NSStringFromCGRect(_layer.frame))
    NSLog(@"%%",[_layer animationForKey:@"KCBasicAnimation_Translation"]);//通过前面的设置
}

#pragma mark 动画结束
-(void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag{
    NSLog(@"animation(%%) stop.\r_layer.frame=%%",anim,NSStringFromCGRect(_layer.frame))
    _layer.position=[[anim valueForKey:@"KCBasicAnimationLocation"] CGPointValue];
}
@end

```

上面通过给动画设置一个代理去监听动画的开始和结束事件，在动画开始前给动画添加一个自定义属性“KCBasicAnimationLocation”存储动画终点位置，然后在动画结束后设置动画的位置为终点位置。

如果运行上面的代码大家可能会发现另外一个问题，那就是动画运行完成后会重新从起始点运动到终点。这个问题产生的原因就是前面提到的，对于非根图层，设置图层的可动画属性（在动画结束后重新设置了position，而position是可动画属性）会产生动画效果。解决这个问题有两种办法：关闭图层隐式动画、设置动画图层为根图层。显然这里不能采取后者，因为根图层当前已经作为动画的背景。

要关闭隐式动画需要用到动画事务CATransaction，在事务内将隐式动画关闭，例如上面的代码可以改为：

```

#pragma mark 动画结束
-(void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag{
    NSLog(@"animation(%%) stop.\r_layer.frame=%%",anim,NSStringFromCGRect(_layer.frame))
    //开启事务
    [CATransaction begin];
    //禁用隐式动画

```

```
[CATransaction setDisableActions:YES];

_layer.position=[anim valueForKey:@"KCBasicAnimationLocation"] CGPointValue];

//提交事务
[CATransaction commit];
}
```

补充

上面通过在animationDidStop中重新设置动画的位置主要为了说明隐式动画关闭和动画事件之间传参的内容，有朋友发现这种方式有可能在动画运行完之后出现从原点瞬间回到终点的过程，最早在调试的时候没有发现这个问题，这里感谢这位朋友。其实解决这个问题并不难，首先必须设置fromValue，其次在动画开始前设置动画position为终点位置（当然也必须关闭隐式动画）。但是这里主要还是出于学习的目的，真正开发的时候做平移动画直接使用隐式动画即可，没有必要那么麻烦。

当然上面的动画还显得有些生硬，因为落花飘散的时候可能不仅仅是自由落体运动，本身由于空气阻力、外界风力还会造成落花在空中的旋转、摇摆等，这里不妨给图层添加一个旋转的动画。对于图层的旋转前面已经演示过怎么通过key path设置图层旋转的内容了，在这里需要强调一下，图层的形变都是基于锚点进行的。例如旋转，旋转的中心点就是图层的锚点。

```
//
//  KCMainViewController.m
//  Animation
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"

@interface KCMainViewController () {
    CALayer *_layer;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景(注意这个图片其实在根图层)
```

```
UIImage *backgroundImage=[UIImage imageNamed:@"background.jpg"];
self.view.backgroundColor=[UIColor colorWithPatternImage:backgroundImage];

//自定义一个图层
_layer=[[CALayer alloc] init];
_layer.bounds=CGRectMake(0, 0, 10, 20);
_layer.position=CGPointMake(50, 150);
_layer.anchorPoint=CGPointMake(0.5, 0.6); //设置锚点
_layer.contents=(id)[UIImage imageNamed:@"petal.png"].CGImage;
[self.view.layer addSublayer:_layer];
}

#pragma mark 移动动画
-(void)translationAnimation:(CGPoint)location{
    //1.创建动画并指定动画属性
    CABasicAnimation *basicAnimation=[CABasicAnimation animationWithKeyPath:@"position"]

    //2.设置动画属性初始值、结束值
    // basicAnimation.fromValue=[NSNumber numberWithInt:50]; //可以不设置，默认为图层初始值
    basicAnimation.toValue=[NSValue valueWithCGPoint:location];

    //设置其他动画属性
    basicAnimation.duration=5.0; //动画时间5秒
    //basicAnimation.repeatCount=HUGE_VALF; //设置重复次数，HUGE_VALF可看做无穷大，起到循环动画的作用
    // basicAnimation.removedOnCompletion=NO; //运行一次是否移除动画
    basicAnimation.delegate=self;
    //存储当前位置在动画结束后使用
    [basicAnimation setValue:[NSValue valueWithCGPoint:location] forKey:@"KCBasicAnimationTranslation"];

    //3.添加动画到图层，注意key相当于给动画进行命名，以后获得该图层时可以使用此名称获取
    [_layer addAnimation:basicAnimation forKey:@"KCBasicAnimation_Translation"];
}

#pragma mark 旋转动画
-(void)rotationAnimation{
    //1.创建动画并指定动画属性
    CABasicAnimation *basicAnimation=[CABasicAnimation animationWithKeyPath:@"transform.rotation"];

    //2.设置动画属性初始值、结束值
    // basicAnimation.fromValue=[NSNumber numberWithInt:M_PI_2];
    basicAnimation.toValue=[NSNumber numberWithFloat:M_PI_2*3];

    //设置其他动画属性
    basicAnimation.duration=6.0;
    basicAnimation.autoreverses=true; //旋转后再旋转到原来的位置

    //4.添加动画到图层，注意key相当于给动画进行命名，以后获得该动画时可以使用此名称获取
```

```
[_layer addAnimation:basicAnimation forKey:@"KCBasicAnimation_Rotation"];
}

#pragma mark 点击事件
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch *touch=touches.anyObject;
    CGPoint location= [touch locationInView:self.view];
    //创建并开始动画
    [self translatonAnimation:location];

    [self rotationAnimation];
}

#pragma mark - 动画代理方法
#pragma mark 动画开始
-(void)animationDidStart:(CAAnimation *)anim{
    NSLog(@"animation(%%) start.\r_layer.frame=%%",anim,NSStringFromCGRect(_layer.frame))
    NSLog(@"%%",[ _layer animationForKey:@"KCBasicAnimation_Translation"]); //通过前面的设置
}

#pragma mark 动画结束
-(void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag{
    NSLog(@"animation(%%) stop.\r_layer.frame=%%",anim,NSStringFromCGRect(_layer.frame))
    //开启事务
    [CATransaction begin];
    //禁用隐式动画
    [CATransaction setDisableActions:YES];

    _layer.position=[[anim valueForKey:@"KCBasicAnimationLocation"] CGPointValue];

    //提交事务
    [CATransaction commit];
}

@end
```

上面代码中结合两种动画操作，需要注意的是只给移动动画设置了代理，在旋转动画中并没有设置代理，否则代理方法会执行两遍。由于旋转动画会无限循环执行（上面设置了重复次数无穷大），并且两个动画的执行时间没有必然的关系，这样一来移动停止后可能还在旋转，为了让移动动画停止后旋转动画停止就需要使用到动画的暂停和恢复方法。

核心动画的运行有一个媒体时间的概念，假设将一个旋转动画设置旋转一周用时60秒的话，那么当动画旋转90度后媒体时间就是15秒。如果此时要将动画暂停只需要让媒体时间偏移量设置为15秒即可，并把动画运行速度设置为0使其停止运动。类似的，如果又过了60秒后需要恢复动画（此时媒体时间为75秒），这时只要将动画开始开始时间设置为当前媒体时间75秒减去暂停时的时间（也就是之前定格动画时的偏移量）15秒（开始时间

=75-15=60秒)，那么动画就会重新计算60秒后的状态再开始运行，与此同时将偏移量重新设置为0并且把运行速度设置1。这个过程中真正起到暂停动画和恢复动画的其实是动画速度的调整，媒体时间偏移量以及恢复时的开始时间设置主要为了让动画更加连贯。

下面的代码演示了移动动画结束后旋转动画暂停，并且当再次点击动画时旋转恢复的过程(注意在移动过程中如果再次点击屏幕可以暂停移动和旋转动画，再次点击可以恢复两种动画。但是当移动结束后触发了移动动画的完成事件如果再次点击屏幕则只能恢复旋转动画，因为此时移动动画已经结束而不是暂停，无法再恢复)。

```
//
//  KCMainViewController.m
//  Animation
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"

@interface KCMainViewController () {
    CALayer *_layer;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景(注意这个图片其实在根图层)
    UIImage *backgroundImage=[UIImage imageNamed:@"background.jpg"];
    self.view.backgroundColor=[UIColor colorWithPatternImage:backgroundImage];

    //自定义一个图层
    _layer=[[CALayer alloc] init];
    _layer.bounds=CGRectMake(0, 0, 10, 20);
    _layer.position=CGPointMake(50, 150);
    _layer.anchorPoint=CGPointMake(0.5, 0.6); //设置锚点
    _layer.contents=(id)[UIImage imageNamed:@"petal.png"].CGImage;
    [self.view.layer addSublayer:_layer];
}

#pragma mark 移动动画
-(void)translationAnimation:(CGPoint)location{
    //1.创建动画并指定动画属性
```

```

    CABasicAnimation *basicAnimation=[CABasicAnimation animationWithKeyPath:@"position"]

    //2.设置动画属性初始值、结束值
    //    basicAnimation.fromValue=[NSNumber numberWithInt:50];//可以不设置，默认为图层初始值
    basicAnimation.toValue=[NSValue valueWithCGPoint:location];

    //设置其他动画属性
    basicAnimation.duration=5.0;//动画时间5秒
    //    basicAnimation.repeatCount=HUGE_VALF;//设置重复次数,HUGE_VALF可看做无穷大，起到循环动画的作用
    basicAnimation.removedOnCompletion=NO;//运行一次是否移除动画
    basicAnimation.delegate=self;
    //存储当前位置在动画结束后使用
    [basicAnimation setValue:[NSValue valueWithCGPoint:location] forKey:@"KCBasicAnimation_Translation"]

    //3.添加动画到图层，注意key相当于给动画进行命名，以后获得该图层时可以使用此名称获取
    [_layer addAnimation:basicAnimation forKey:@"KCBasicAnimation_Translation"];
}

```

#pragma mark 旋转动画

```

-(void)rotationAnimation{
    //1.创建动画并指定动画属性
    CABasicAnimation *basicAnimation=[CABasicAnimation animationWithKeyPath:@"transform.rotation"]

    //2.设置动画属性初始值、结束值
    //    basicAnimation.fromValue=[NSNumber numberWithInt:M_PI_2];
    basicAnimation.toValue=[NSNumber numberWithFloat:M_PI_2*3];

    //设置其他动画属性
    basicAnimation.duration=6.0;
    basicAnimation.autoreverses=true;//旋转后在旋转到原来的位置
    basicAnimation.repeatCount=HUGE_VALF;//设置无限循环
    basicAnimation.removedOnCompletion=NO;
    //    basicAnimation.delegate=self;

    //4.添加动画到图层，注意key相当于给动画进行命名，以后获得该动画时可以使用此名称获取
    [_layer addAnimation:basicAnimation forKey:@"KCBasicAnimation_Rotation"];
}

```

#pragma mark 点击事件

```

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch *touch=touches.anyObject;
    CGPoint location= [touch locationInView:self.view];
    //判断是否已经常见过动画，如果已经创建则不再创建动画
    CAAnimation *animation= [_layer animationForKey:@"KCBasicAnimation_Translation"];
    if(animation){
        if (_layer.speed==0) {
            [self animationResume];
        }else{
            [self animationPause];
        }
    }
}

```

```
        }
    }else{
        //创建并开始动画
        [self translatonAnimation:location];

        [self rotationAnimation];
    }
}

#pragma mark 动画暂停
-(void)animationPause{
    //取得指定图层动画的媒体时间，后面参数用于指定子图层，这里不需要
    CFTimeInterval interval=[_layer convertTime:CACurrentMediaTime() fromLayer:nil];
    //设置时间偏移量，保证暂停时停留在旋转的位置
    [_layer setTimeOffset:interval];
    //速度设置为0，暂停动画
    _layer.speed=0;
}

#pragma mark 动画恢复
-(void)animationResume{
    //获得暂停的时间
    CFTimeInterval beginTime= CACurrentMediaTime()- _layer.timeOffset;
    //设置偏移量
    _layer.timeOffset=0;
    //设置开始时间
    _layer.beginTime=beginTime;
    //设置动画速度，开始运动
    _layer.speed=1.0;
}

#pragma mark - 动画代理方法
#pragma mark 动画开始
-(void)animationDidStart:(CAAnimation *)anim{
    NSLog(@"animation(%%) start.\r_layer.frame=%%",anim,NSStringFromCGRect(_layer.frame));
    NSLog(@"%%",[_layer animationForKey:@"KCBasicAnimation_Translation"]); //通过前面的设置
}

#pragma mark 动画结束
-(void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag{
    NSLog(@"animation(%%) stop.\r_layer.frame=%%",anim,NSStringFromCGRect(_layer.frame));

    //开启事务
    [CATransaction begin];
    //禁用隐式动画
    [CATransaction setDisableActions:YES];

    _layer.position=[[anim valueForKey:@"KCBasicAnimationLocation"] CGPointValue];

    //提交事务
}
```



```
[CATransaction commit];

//暂停动画
[self animationPause];

}

@end
```

运行效果：



注意：

- 动画暂停针对的是图层而不是图层中的某个动画。

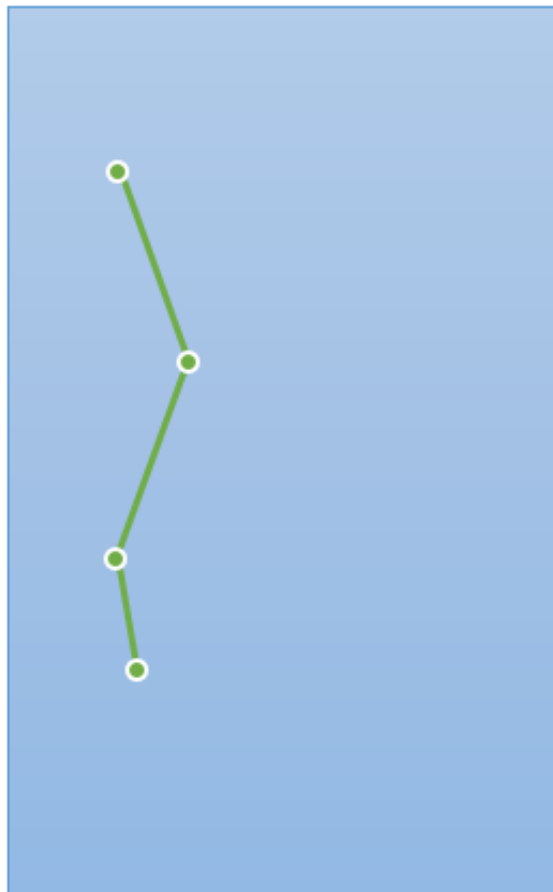
- 要做无限循环的动画，动画的removedOnCompletion属性必须设置为NO，否则运行一次动画就会销毁。

关键帧动画

熟悉flash开发的朋友对于关键帧动画应该不陌生，这种动画方式在flash开发中经常用到。关键帧动画就是在动画控制过程中开发者指定主要的动画状态，至于各个状态间动画如何进行则由系统自动运算补充（每两个关键帧之间系统形成的动画称为“补间动画”），这种动画的好处就是开发者不用逐个控制每个动画帧，而只要关心几个关键帧的状态即可。

关键帧动画开发分为两种形式：一种是通过设置不同的属性值进行关键帧控制，另一种是通过绘制路径进行关键帧控制。后者优先级高于前者，如果设置了路径则属性值就不再起作用。

对于前面的落花动画效果而言其实落花的过程并不自然，很显然实际生活中它不可能沿着直线下落，这里我们不妨通过关键帧动画的values属性控制它在下落过程中的属性。假设下落过程如图：



在这里需要设置四个关键帧（如图中四个关键点），具体代码如下（动画创建过程同基本动画基本完全一致）：

```
//
// 通过values设置关键帧动画
// Animation
//
// Created by Kenshin Cui on 14-3-22.
// Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"

@interface KCMainViewController () {
    CALayer *_layer;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景(注意这个图片其实在根图层)
    UIImage *backgroundImage=[UIImage imageNamed:@"background.jpg"];
    self.view.backgroundColor=[UIColor colorWithPatternImage:backgroundImage];

    //自定义一个图层
    _layer=[[CALayer alloc]init];
    _layer.bounds=CGRectMake(0, 0, 10, 20);
    _layer.position=CGPointMake(50, 150);
    _layer.contents=(id)[UIImage imageNamed:@"petal.png"].CGImage;
    [self.view.layer addSublayer:_layer];

    //创建动画
    [self translationAnimation];
}

#pragma mark 关键帧动画
-(void)translationAnimation{
    //1.创建关键帧动画并设置动画属性
    CAKeyframeAnimation *keyframeAnimation=[CAKeyframeAnimation animationWithKeyPath:@"t"];

    //2.设置关键帧,这里有四个关键帧
    NSValue *key1=[NSValue valueWithCGPoint:_layer.position];//对于关键帧动画初始值不能省略
    NSValue *key2=[NSValue valueWithCGPoint:CGPointMake(80, 220)];
    NSValue *key3=[NSValue valueWithCGPoint:CGPointMake(45, 300)];
    NSValue *key4=[NSValue valueWithCGPoint:CGPointMake(55, 400)];
    NSArray *values=@[key1,key2,key3,key4];
    keyframeAnimation.values=values;
    //设置其他属性
    keyframeAnimation.duration=8.0;
```

```
keyframeAnimation.beginTime=CACurrentMediaTime()+2;//设置延迟2秒执行
```

```
//3.添加动画到图层，添加动画后就会执行动画
```

```
[_layer addAnimation:keyframeAnimation forKey:@"KCKeyframeAnimation_Position"];
```

```
}
```

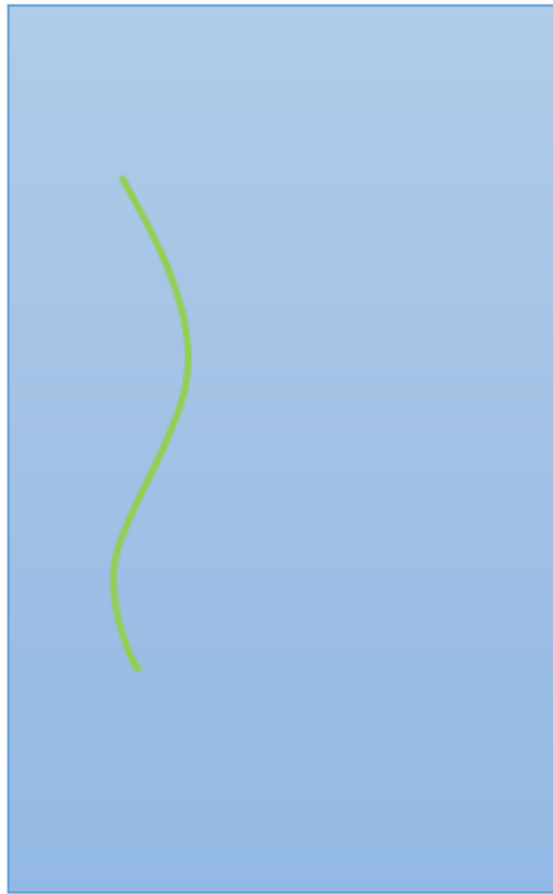
```
@end
```

运行效果(注意运行结束没有设置图层位置为动画运动结束位置):



上面的方式固然比前面使用基础动画效果要好一些，但其实还是存在问题，那就是落花飘落的路径是直线的，当然这个直线是根据程序中设置的四个关键帧自动形成的，那么如何让它沿着曲线飘落呢？这就是第二种类型的关键帧动画，通过描绘路径进行关键帧动画控

制。假设让落花沿着下面的曲线路径飘落：



当然，这是一条贝塞尔曲线，学习了前篇文章之后相信对于这类曲线应该并不陌生，下面是具体实现代码：

```
//  
// 通过path设置关键帧动画  
// Animation  
//  
// Created by Kenshin Cui on 14-3-22.  
// Copyright (c) 2014年 Kenshin Cui. All rights reserved.  
//  
  
#import "KCMainViewController.h"  
  
@interface KCMainViewController () {  
    CALayer *_layer;  
}  
  
@end  
  
@implementation KCMainViewController
```

```
- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景(注意这个图片其实在根图层)
    UIImage *backgroundImage=[UIImage imageNamed:@"background.jpg"];
    self.view.backgroundColor=[UIColor colorWithPatternImage:backgroundImage];

    //自定义一个图层
    _layer=[[CALayer alloc]init];
    _layer.bounds=CGRectMake(0, 0, 10, 20);
    _layer.position=CGPointMake(50, 150);
    _layer.contents=(id)[UIImage imageNamed:@"petal.png"].CGImage;
    [self.view.layer addSublayer:_layer];

    //创建动画
    [self translationAnimation];
}


#pragma mark 关键帧动画
-(void)translationAnimation{
    //1.创建关键帧动画并设置动画属性
    CAKeyframeAnimation *keyframeAnimation=[CAKeyframeAnimation animationWithKeyPath:@"t"];

    //2.设置路径
    //绘制贝塞尔曲线
    CGPathRef path=CGPathCreateMutable();
    CGPathMoveToPoint(path, NULL, _layer.position.x, _layer.position.y); //移动到起始点
    CGPathAddCurveToPoint(path, NULL, 160, 280, -30, 300, 55, 400); //绘制二次贝塞尔曲线

    keyframeAnimation.path=path; //设置path属性
    CGPathRelease(path); //释放路径对象
    //设置其他属性
    keyframeAnimation.duration=8.0;
    keyframeAnimation.beginTime=CACurrentMediaTime()+5; //设置延迟2秒执行

    //3.添加动画到图层, 添加动画后就会执行动画
    [_layer addAnimation:keyframeAnimation forKey:@"KCKeyframeAnimation_Position"];
}

@end
```



运行效果(注意运行结束没有设置图层位置为动画运动结束位置):



看起来动画不会那么生硬了，但是这里需要注意，对于路径类型的关键帧动画系统是从描绘路径的位置开始路径，直到路径结束。如果上面的路径不是贝塞尔曲线而是矩形路径那么它会从矩形的左上角开始运行，顺时针一周回到左上角；如果指定的路径是一个椭圆，那么动画运行的路径是从椭圆右侧开始（0度）顺时针一周回到右侧。

补充--其他属性

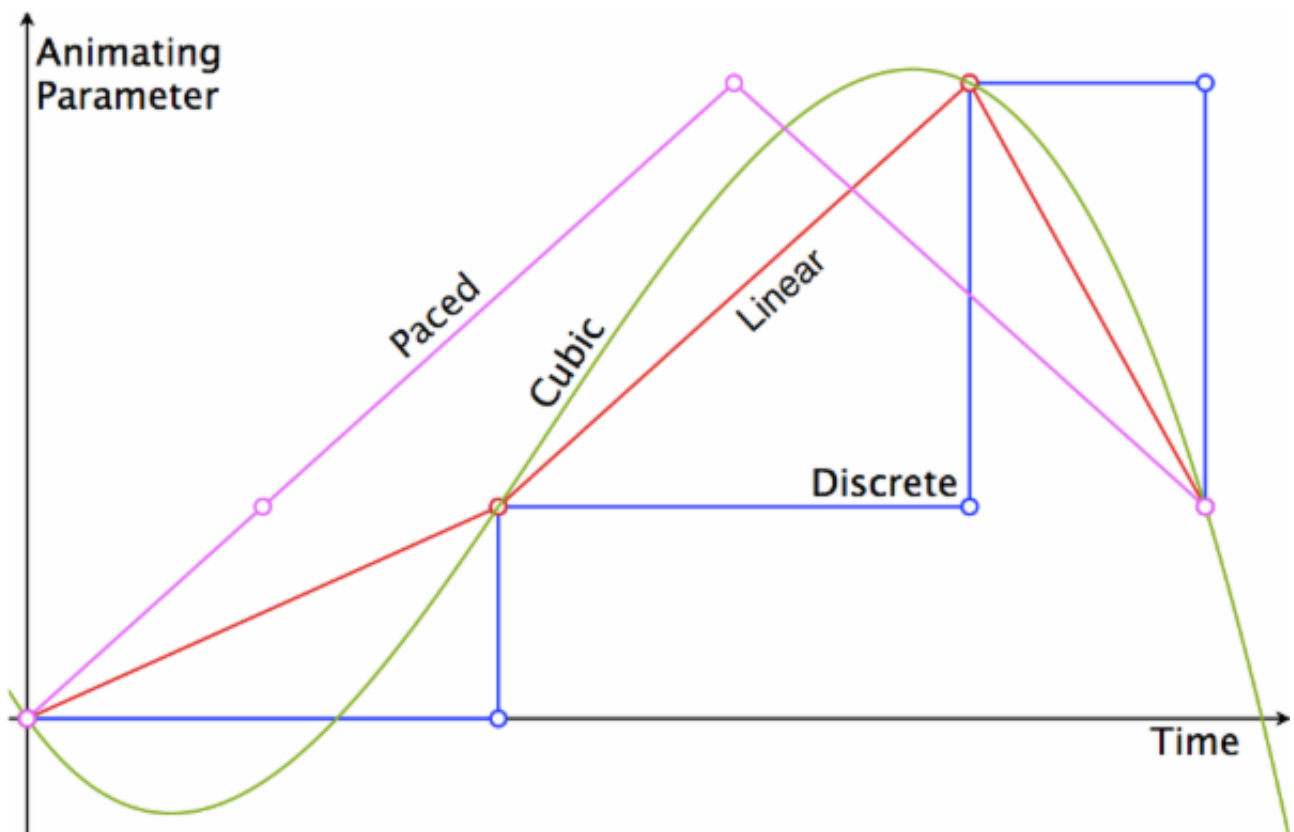
在关键帧动画中还有一些动画属性初学者往往比较容易混淆，这里专门针对这些属性做一下介绍。

keyTimes：各个关键帧的时间控制。前面使用values设置了四个关键帧，默认情况下每两帧之间的间隔为： $8/(4-1)$ 秒。如果想要控制动画从第一帧到第二帧占用时间4秒，从第二帧到第三帧时间为2秒，而从第三帧到第四帧时间2秒的话，就可以通过keyTimes进行设置。keyTimes中存储的是时间占用比例点，此时可以设置keyTimes的值为0.0, 0.5,

0.75, 1.0 (当然必须转换为NSNumber), 也就是说1到2帧运行到总时间的50%, 2到3帧运行到总时间的75%, 3到4帧运行到8秒结束。

calculationMode: 动画计算模式。还拿上面keyValues动画举例, 之所以1到2帧能形成连贯性动画而不是直接从第1帧经过8/3秒到第2帧是因为动画模式是连续的 (值为kCAAnimationLinear, 这是计算模式的默认值); 而如果指定了动画模式为kCAAnimationDiscrete离散的那么你会看到动画从第1帧经过8/3秒直接到第2帧, 中间没有任何过渡。其他动画模式还有: kCAAnimationPaced (均匀执行, 会忽略keyTimes)、kCAAnimationCubic (平滑执行, 对于位置变动关键帧动画运行轨迹更平滑)、kCAAnimationCubicPaced (平滑均匀执行)。

下图描绘出了几种动画模式的关系 (横坐标是运行时间, 纵坐标是动画属性[例如位置、透明度等]):



动画组

实际开发中一个物体的运动往往是复合运动, 单一属性的运动情况比较少, 但恰恰属性动画每次进行动画设置时一次只能设置一个属性进行动画控制(不管是基础动画还是关键帧动画都是如此), 这样一来要做一个复合运动的动画就必须创建多个属性动画进行组合。对于一两种动画的组合或许处理起来还比较容易, 但是对于更多动画的组合控制往往会变得很麻烦, 动画组的产生就是基于这样一种情况而产生的。动画组是一系列动画的组合, 凡是添加到动画组中的动画都受控于动画组, 这样一来各类动画公共的行为就可以统一进行控

制而不必单独设置，而且放到动画组中的各个动画可以并发执行，共同构建出复杂的动画效果。

动画组使用起来并不复杂，首先单独创建单个动画（可以是基础动画也可以是关键帧动画），然后将基础动画添加到动画组，最后将动画组添加到图层即可。

前面关键帧动画部分，路径动画看起来效果虽然很流畅，但是落花本身的旋转运动没有了，这里不妨将基础动画部分的旋转动画和路径关键帧动画进行组合使得整个动画看起来更加的和谐、顺畅。

```
//
// 动画组
// Animation
//
// Created by Kenshin Cui on 14-3-22.
// Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"

@interface KCMainViewController () {
    CALayer *_layer;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景(注意这个图片其实在根图层)
    UIImage *backgroundImage=[UIImage imageNamed:@"background.jpg"];
    self.view.backgroundColor=[UIColor colorWithPatternImage:backgroundImage];

    //自定义一个图层
    _layer=[[CALayer alloc] init];
    _layer.bounds=CGRectMake(0, 0, 10, 20);
    _layer.position=CGPointMake(50, 150);
    _layer.contents=(id)[UIImage imageNamed:@"petal.png"].CGImage;
    [self.view.layer addSublayer:_layer];

    //创建动画
    [self groupAnimation];
}

#pragma mark 基础旋转动画
-(CABasicAnimation *)rotationAnimation{
```

```

CABasicAnimation *basicAnimation=[CABasicAnimation animationWithKeyPath:@"transform.

CGFloat toValue=M_PI_2*3;
basicAnimation.toValue=[NSNumber numberWithFloat:M_PI_2*3];

//    basicAnimation.duration=6.0;
basicAnimation.autoreverses=true;
basicAnimation.repeatCount=HUGE_VALF;
basicAnimation.removedOnCompletion=NO;

[basicAnimation setValue:[NSNumber numberWithFloat:toValue] forKey:@"KCBasicAnimatic

return basicAnimation;
}

#pragma mark 关键帧移动动画
-(CAKeyframeAnimation *)translationAnimation{
    CAKeyframeAnimation *keyframeAnimation=[CAKeyframeAnimation animationWithKeyPath:@"t

    CGPoint endPoint= CGPointMake(55, 400);
    CGPathRef path=CGPathCreateMutable();
    CGPathMoveToPoint(path, NULL, _layer.position.x, _layer.position.y);
    CGPathAddCurveToPoint(path, NULL, 160, 280, -30, 300, endPoint.x, endPoint.y);

    keyframeAnimation.path=path;
    CGPathRelease(path);

    [keyframeAnimation setValue:[NSValue valueWithCGPoint:endPoint] forKey:@"KCKeyframeA

    return keyframeAnimation;
}

#pragma mark 创建动画组
-(void)groupAnimation{
    //1.创建动画组
    CAAAnimationGroup *animationGroup=[CAAnimationGroup animation];

    //2.设置组中的动画和其他属性
    CABasicAnimation *basicAnimation=[self rotationAnimation];
    CAKeyframeAnimation *keyframeAnimation=[self translationAnimation];
    animationGroup.animations=@[basicAnimation,keyframeAnimation];

    animationGroup.delegate=self;
    animationGroup.duration=10.0;//设置动画时间，如果动画组中动画已经设置过动画属性则不再生效
    animationGroup.beginTime=CACurrentMediaTime()+5;//延迟五秒执行

    //3.给图层添加动画
    [_layer addAnimation:animationGroup forKey:nil];
}

```

```
#pragma mark - 代理方法
#pragma mark 动画完成
-(void)animationDidStop:(CAAnimation *)anim finished:(BOOL)flag{
    CAAAnimationGroup *animationGroup=(CAAnimationGroup *)anim;
    CABasicAnimation *basicAnimation=animationGroup.animations[0];
    CAKeyframeAnimation *keyframeAnimation=animationGroup.animations[1];
    CGFloat toValue=[[basicAnimation valueForKey:@"KCBasicAnimationProperty_ToValue"] floatValue];
    CGPoint endPoint=[[keyframeAnimation valueForKey:@"KCKeyframeAnimationProperty_EndPosition"] CGPointValue];

    [CATransaction begin];
    [CATransaction setDisableActions:YES];

    //设置动画最终状态
    _layer.position=endPoint;
    _layer.transform=CATransform3DMakeRotation(toValue, 0, 0, 1);

    [CATransaction commit];
}

@end
```



运行效果：



转场动画

转场动画就是从一个场景以动画的形式过渡到另一个场景。转场动画的使用一般分为以下几个步骤：

- 1.创建转场动画
- 2.设置转场类型、子类型（可选）及其他属性
- 3.设置转场后的新视图并添加动画到图层

下表列出了常用的转场类型(注意私有API是苹果官方没有公开的动画类型，但是目前通过仍然可以使用)：

--	--	--

动画类型	说明	对应常量
公开API		
fade	淡出效果	kCATransitionFade
movein	新视图移动到旧视图上	kCATransitionMoveIn
push	新视图推出旧视图	kCATransitionPush
reveal	移开旧视图显示新视图	kCATransitionReveal
私有API		私有API只能通过字符串访问
cube	立方体翻转效果	无
ogIFlip	翻转效果	无
suckEffect	收缩效果	无
rippleEffect	水滴波纹效果	无
pageCurl	向上翻页效果	无
pageUnCurl	向下翻页效果	无
cameraIrisHollowOpen	摄像头打开效果	无
cameraIrisHollowClose	摄像头关闭效果	无

另外对于支持方向设置的动画类型还包含子类型：

动画子类型	说明
kCATransitionFromRight	从右侧转场
kCATransitionFromLeft	从左侧转场
kCATransitionFromTop	从顶部转场
kCATransitionFromBottom	从底部转场

在前面的文章“[iOS开发系列--无限循环的图片浏览器](#)”中为了使用UIScrollView做无限循环图片浏览器花费了不少时间在性能优化上面，这里使用转场动画利用一个UIImageView实现一个漂亮的无限循环图片浏览器。

```
//
//  KCMainViewController.m
//  TransitionAnimation
//
//  Created by Kenshin Cui on 14-3-12.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"
#define IMAGE_COUNT 5

@interface KCMainViewController ()
```

```
    UIImageView *_imageView;
    int _currentIndex;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //定义图片控件
    _imageView=[[UIImageView alloc]init];
    _imageView.frame=[UIScreen mainScreen].applicationFrame;
    _imageView.contentMode=UIViewContentModeScaleAspectFit;
    _imageView.image=[UIImage imageNamed:@"0.jpg"]; //默认图片
    [self.view addSubview:_imageView];
    //添加手势
    UISwipeGestureRecognizer *leftSwipeGesture=[[UISwipeGestureRecognizer alloc] initWithGestureRecognizer:self];
    leftSwipeGesture.direction=UISwipeGestureRecognizerDirectionLeft;
    [self.view addGestureRecognizer:leftSwipeGesture];

    UISwipeGestureRecognizer *rightSwipeGesture=[[UISwipeGestureRecognizer alloc] initWithGestureRecognizer:self];
    rightSwipeGesture.direction=UISwipeGestureRecognizerDirectionRight;
    [self.view addGestureRecognizer:rightSwipeGesture];
}

#pragma mark 向左滑动浏览下一张图片
-(void)leftSwipe:(UISwipeGestureRecognizer *)gesture{
    [self transitionAnimation:YES];
}

#pragma mark 向右滑动浏览上一张图片
-(void)rightSwipe:(UISwipeGestureRecognizer *)gesture{
    [self transitionAnimation:NO];
}

#pragma mark 转场动画
-(void)transitionAnimation:(BOOL)isNext{
    //1.创建转场动画对象
    CATransition *transition=[[CATransition alloc]init];

    //2.设置动画类型,注意对于苹果官方没公开的动画类型只能使用字符串,并没有对应的常量定义
    transition.type=@"cube";

    //设置子类型
    if (isNext) {
        transition.subtype=kCATransitionFromRight;
    }else{

```

```
        transition.subtype=kCATransitionFromLeft;
    }
    //设置动画时常
    transition.duration=1.0f;

    //3.设置转场后的新视图添加转场动画
    _imageView.image=[self getImage:isNext];
    [_imageView.layer addAnimation:transition forKey:@"KCTransitionAnimation"];
}

#pragma mark 取得当前图片
-(UIImage *)getImage:(BOOL)isNext{
    if (isNext) {
        _currentIndex=(_currentIndex+1)%IMAGE_COUNT;
    }else{
        _currentIndex=(_currentIndex-1+IMAGE_COUNT)%IMAGE_COUNT;
    }
    NSString *imageName=[NSString stringWithFormat:@"%i.jpg",_currentIndex];
    return [UIImage imageNamed:imageName];
}
@end
```



运行效果：



代码十分简单，但是效果和性能却很惊人。当然演示代码有限，其他动画类型大家可以自己实现，效果都很绚丽。

逐帧动画

前面介绍了核心动画中大部分动画类型，但是做过动画处理的朋友都知道，在动画制作中还有一种动画类型“逐帧动画”。说到逐帧动画相信很多朋友第一个想到的就是 `UIImageView`，通过设置 `UIImageView` 的 `animationImages` 属性，然后调用它的 `startAnimating` 方法去播放这组图片。当然这种方法在某些场景下是可以达到逐帧的动画效果，但是它也存在着很大的性能问题，并且这种方法一旦设置完图片中间的过程就无法控制了。当然，也许有朋友会想到利用iOS的定时器 `NSTimer` 定时更新图片来达到逐帧动画的效果。这种方式确实可以解决 `UIImageView` 一次性加载大量图片的问题，而且让播放过程可控，唯一的缺点就是定时器方法调用有时可能会因为当前系统执行某种比较占用时

间的任务造成动画连续性出现问题。

虽然在核心动画没有直接提供逐帧动画类型，但是却提供了用于完成逐帧动画的相关对象CADisplayLink。CADisplayLink是一个计时器，但是同NSTimer不同的是，CADisplayLink的刷新周期同屏幕完全一致。例如在iOS中屏幕刷新周期是60次/秒，CADisplayLink刷新周期同屏幕刷新一致也是60次/秒，这样一来使用它完成的逐帧动画（又称为“时钟动画”）完全感觉不到动画的停滞情况。

在iOS开篇“[iOS开发系列--iOS程序开发概览](#)”中就曾说过：iOS程序在运行后就进入一个消息循环中（这个消息循环称为“主运行循环”），整个程序相当于进入一个死循环中，始终等待用户输入。将CADisplayLink加入到主运行循环队列后，它的时钟周期就和主运行循环保持一致，而主运行循环周期就是屏幕刷新周期。在CADisplayLink加入到主运行循环队列后就会循环调用目标方法，在这个方法中更新视图内容就可以完成逐帧动画。

当然这里不得不强调的是逐帧动画性能势必较低，但是对于一些事物的运动又不得不选择使用逐帧动画，例如人的运动，这是一个高度复杂的运动，基本动画、关键帧动画是不可能解决的。所以大家一定要注意在循环方法中尽可能的降低算法复杂度，同时保证循环过程中内存峰值尽可能低。下面以一个鱼的运动为例为大家演示一下逐帧动画。

```
//
//  KCMainViewController.m
//  DisplayLink
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"
#define IMAGE_COUNT 10

@interface KCMainViewController () {
    CALayer *_layer;
    int _index;
    NSMutableArray *_images;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景
    self.view.layer.contents=(id)[UIImage imageNamed:@"bg.png"].CGImage;
```

```
//创建图像显示图层
_layer=[[CALayer alloc]init];
_layer.bounds=CGRectMake(0, 0, 87, 32);
_layer.position=CGPointMake(160, 284);
[self.view.layer addSublayer:_layer];

//由于鱼的图片在循环中会不断创建，而10张鱼的照片相对都很小
//与其在循环中不断创建UIImage不如直接将10张图片缓存起来
_images=[NSMutableArray array];
for (int i=0; i<10; ++i) {
    NSString *imageName=[NSString stringWithFormat:@"fish%i.png",i];
    UIImage *image=[UIImage imageNamed:imageName];
    [_images addObject:image];
}

//定义时钟对象
CADisplayLink *displayLink=[CADisplayLink displayLinkWithTarget:self selector:@selec
//添加时钟对象到主运行循环
[displayLink addToRunLoop:[NSRunLoop mainRunLoop] forMode:NSDefaultRunLoopMode];
}

#pragma mark 每次屏幕刷新就会执行一次此方法(每秒接近60次)
-(void)step{
    //定义一个变量记录执行次数
    static int s=0;
    //每秒执行6次
    if (++s%10==0) {
        UIImage *image=_images[_index];
        _layer.contents=(id)image.CGImage;//更新图片
        _index=(_index+1)%IMAGE_COUNT;
    }
}
@end
```

运行效果：



注意：上面仅仅演示了逐帧动画的过程，事实上结合其他动画类型可以让整条鱼游动起来，这里不再赘述。

UIView动画封装

有了前面核心动画的知识，相信大家开发出一般的动画效果应该不在话下。在核心动画开篇也给大家说过，其实UIView本身对于基本动画和关键帧动画、转场动画都有相应的封装，在对动画细节没有特殊要求的情况下使用起来也要简单的多。可以说在日常开发中90%以上的情况使用UIView的动画封装方法都可以搞定，因此在熟悉了核心动画的原理之后还是有必要给大家简单介绍一下UIView中各类动画使用方法的。由于前面核心动画内容已经进行过详细介绍，学习UIView的封装方法根本是小菜一碟，这里对于一些细节就不再赘述了。

基础动画

基础动画部分和前面的基础动画演示相对应，演示点击屏幕落叶飘落到鼠标点击位置的过程。注意根据前面介绍的隐式动画知识其实非根图层直接设置终点位置不需要使用UIView的动画方法也可以实现动画效果，因此这里落花不再放到图层中而是放到了一个UIImageView中。

下面的代码演示了通过block和静态方法实现动画控制的过程：

```
//
//  UIView实现基础动画
//  Animation
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"

@interface KCMainViewController () {
    UIImageView *_imageView;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景
    UIImage *backgroundImage=[UIImage imageNamed:@"background.jpg"];
    self.view.backgroundColor=[UIColor colorWithPatternImage:backgroundImage];

    //创建图像显示控件
    _imageView=[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"petal.png"]];
    _imageView.center=CGPointMake(50, 150);
    [self.view addSubview:_imageView];
}

#pragma mark 点击事件
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch *touch=touches.anyObject;
    CGPoint location= [touch locationInView:self.view];
    //方法1: block方式
    /*开始动画，UIView的动画方法执行完后动画会停留在重点位置，而不需要进行任何特殊处理
    duration:执行时间
    delay:延迟时间
    */
}
```

```

        options:动画设置,例如自动恢复、匀速运动等
        completion:动画完成回调方法
    */
//    [UIView animateWithDuration:5.0 delay:0 options:UIViewAnimationOptionCurveLinear &
//    _imageView.center=location;
//    } completion:^(BOOL finished) {
//        NSLog(@"Animation end.");
//    }];

//方法2: 静态方法
//开始动画
[UIView beginAnimations:@"KCBasicAnimation" context:nil];
[UIView setAnimationDuration:5.0];
//[UIView setAnimationDelay:1.0];//设置延迟
//[UIView setAnimationRepeatAutoreverses:NO];//是否回复
//[UIView setAnimationRepeatCount:10];//重复次数
//[UIView setAnimationStartDate:(NSDate *)];//设置动画开始运行的时间
//[UIView setAnimationDelegate:self];//设置代理
//[UIView setAnimationWillStartSelector:(SEL)];//设置动画开始运动的执行方法
//[UIView setAnimationDidStopSelector:(SEL)];//设置动画运行结束后的执行方法

_imageView.center=location;

//开始动画
[UIView commitAnimations];
}
@end

```

补充--弹簧动画效果

由于在iOS开发中弹性动画使用很普遍,所以在iOS7苹果官方直接提供了一个方法用于弹性动画开发,下面简单的演示一下:

```

//
//  UIView实现基础动画--弹性动画
//  Animation
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"

@interface KCMainViewController () {
    UIImageView *_imageView;
}

```

```
@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

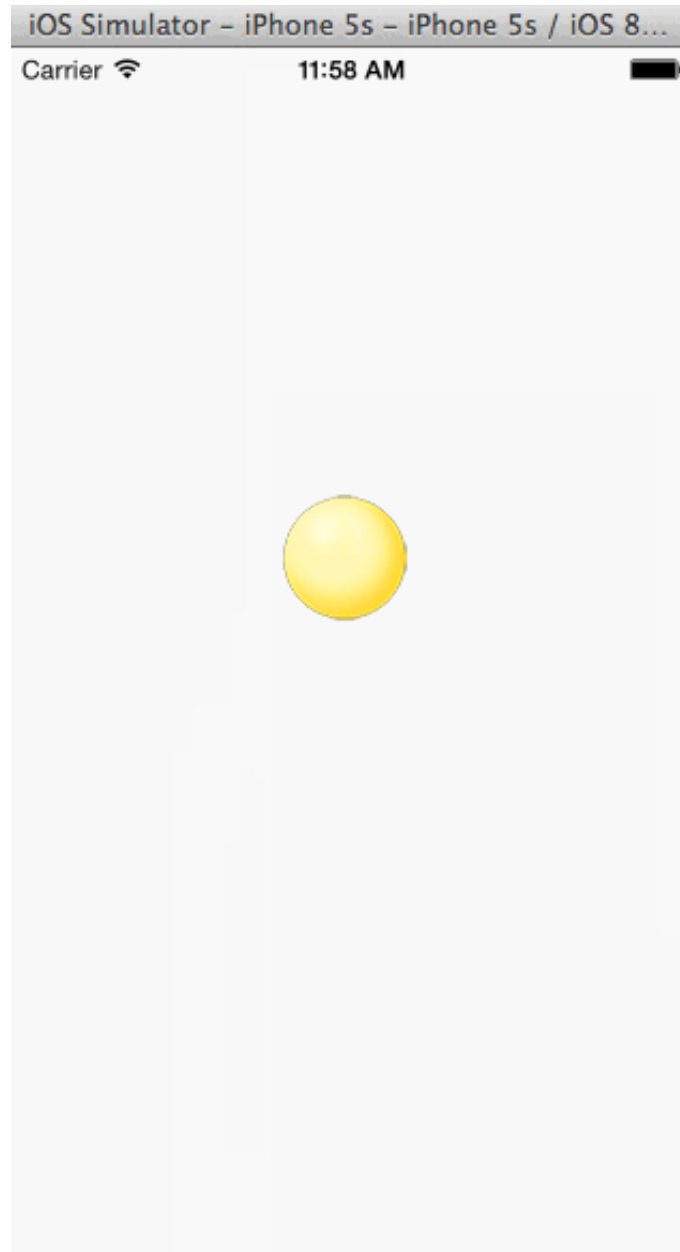
    //创建图像显示控件
    _imageView=[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"ball.png"]];
    _imageView.center=CGPointMake(160, 50);
    [self.view addSubview:_imageView];
}

#pragma mark 点击事件
-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
    UITouch *touch=touches.anyObject;
    CGPoint location= [touch locationInView:self.view];
    /*创建弹性动画
    damping:阻尼, 范围0-1, 阻尼越接近于0, 弹性效果越明显
    velocity:弹性复位的速度
    */
    [UIView animateWithDuration:5.0 delay:0 usingSpringWithDamping:0.1 initialSpringVelc
        _imageView.center=location; //CGPointMake(160, 284);
    ] completion:nil];
}

@end
```



运行效果：



补充--动画设置参数

在动画方法中有一个option参数，UIViewAnimationOptions类型，它是一个枚举类型，动画参数分为三类，可以组合使用：

1.常规动画属性设置（可以同时选择多个进行设置）

UIViewAnimationOptionLayoutSubviews：动画过程中保证子视图跟随运动。

UIViewAnimationOptionAllowUserInteraction：动画过程中允许用户交互。

UIViewAnimationOptionBeginFromCurrentState：所有视图从当前状态开始运行。

UIViewAnimationOptionRepeat：重复运行动画。

`UIViewAnimationOptionAutoreverse`：动画运行到结束后仍然以动画方式回到初始点。

`UIViewAnimationOptionOverrideInheritedDuration`：忽略嵌套动画时间设置。

`UIViewAnimationOptionOverrideInheritedCurve`：忽略嵌套动画速度设置。

`UIViewAnimationOptionAllowAnimatedContent`：动画过程中重绘视图（注意仅仅适用于转场动画）。

`UIViewAnimationOptionShowHideTransitionViews`：视图切换时直接隐藏旧视图、显示新视图，而不是将旧视图从父视图移除（仅仅适用于转场动画）

`UIViewAnimationOptionOverrideInheritedOptions`：不继承父动画设置或动画类型。

2.动画速度控制（可从其中选择一个设置）

`UIViewAnimationOptionCurveEaseInOut`：动画先缓慢，然后逐渐加速。

`UIViewAnimationOptionCurveEaseIn`：动画逐渐变慢。

`UIViewAnimationOptionCurveEaseOut`：动画逐渐加速。

`UIViewAnimationOptionCurveLinear`：动画匀速执行，默认值。

3.转场类型（仅适用于转场动画设置，可以从中选择一个进行设置，基本动画、关键帧动画不需要设置）

`UIViewAnimationOptionTransitionNone`：没有转场动画效果。

`UIViewAnimationOptionTransitionFlipFromLeft`：从左侧翻转效果。

`UIViewAnimationOptionTransitionFlipFromRight`：从右侧翻转效果。

`UIViewAnimationOptionTransitionCurlUp`：向后翻页的动画过渡效果。

`UIViewAnimationOptionTransitionCurlDown`：向前翻页的动画过渡效果。

`UIViewAnimationOptionTransitionCrossDissolve`：旧视图溶解消失显示下一个新视图的效果。

`UIViewAnimationOptionTransitionFlipFromTop`：从上方翻转效果。

`UIViewAnimationOptionTransitionFlipFromBottom`：从底部翻转效果。

关键帧动画

从iOS7开始UIView动画中封装了关键帧动画，下面就来看一下如何使用UIView封装方法进行关键帧动画控制，这里实现前面关键帧动画部分对于落花的控制。

```
//
//  UIView关键帧动画
//  UIViewAnimation
//
//  Created by Kenshin Cui on 14-3-22.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"

@interface KCMainViewController () {
    UIImageView *_imageView;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //设置背景
    UIImage *backgroundImage=[UIImage imageNamed:@"background.jpg"];
    self.view.backgroundColor=[UIColor colorWithPatternImage:backgroundImage];

    //创建图像显示控件
    _imageView=[[UIImageView alloc] initWithImage:[UIImage imageNamed:@"petal.png"]];
    _imageView.center=CGPointMake(50, 150);
    [self.view addSubview:_imageView];
}

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event{
    //UITouch *touch=touches.anyObject;
    //CGPoint location= [touch locationInView:self.view];

    /*关键帧动画
    options:
    */
    [UIView animateKeyframesWithDuration:5.0 delay:0 options: UIViewAnimationOptionCurveEaseInOut
    //第二个关键帧（准确的说第一个关键帧是开始位置）：从0秒开始持续50%的时间，也就是5.0*0.5=2.5秒
    [UIView addKeyframeWithRelativeStartTime:0.0 relativeDuration:0.5 animations:^(
        _imageView.center=CGPointMake(80.0, 220.0);
    )];
    //第三个关键帧，从0.5*5.0秒开始，持续5.0*0.25=1.25秒
```

```
[UIView addKeyframeWithRelativeStartTime:0.5 relativeDuration:0.25 animations:^(
    _imageView.center=CGPointMake(45.0, 300.0);
)];
//第四个关键帧：从0.75*5.0秒开始，持所需5.0*0.25=1.25秒
[UIView addKeyframeWithRelativeStartTime:0.75 relativeDuration:0.25 animations:^(
    _imageView.center=CGPointMake(55.0, 400.0);
)];

} completion:^(BOOL finished) {
    NSLog(@"Animation end.");
}];
}
@end
```

补充--动画设置参数

对于关键帧动画也有一些动画参数设置options，UIViewKeyframeAnimationOptions类型，和上面基本动画参数设置有些差别，关键帧动画设置参数分为两类，可以组合使用：

1.常规动画属性设置（可以同时选择多个进行设置）

UIViewAnimationOptionLayoutSubviews：动画过程中保证子视图跟随运动。

UIViewAnimationOptionAllowUserInteraction：动画过程中允许用户交互。

UIViewAnimationOptionBeginFromCurrentState：所有视图从当前状态开始运行。

UIViewAnimationOptionRepeat：重复运行动画。

UIViewAnimationOptionAutoreverse：动画运行到结束后仍然以动画方式回到初始点。

UIViewAnimationOptionOverrideInheritedDuration：忽略嵌套动画时间设置。

UIViewAnimationOptionOverrideInheritedOptions：不继承父动画设置或动画类型。

2.动画模式设置（同前面关键帧动画动画模式一一对应，可以从其中选择一个进行设置）

UIViewKeyframeAnimationOptionCalculationModeLinear：连续运算模式。

UIViewKeyframeAnimationOptionCalculationModeDiscrete：离散运算模式。

UIViewKeyframeAnimationOptionCalculationModePaced：均匀执行运算模式。

UIViewKeyframeAnimationOptionCalculationModeCubic：平滑运算模式。

UIViewKeyframeAnimationOptionCalculationModeCubicPaced: 平滑均匀运算模式。

注意: 前面说过关键帧动画有两种形式, 上面演示的是属性值关键帧动画, 路径关键帧动画目前UIView还不支持。

转场动画

从iOS4.0开始, UIView直接封装了转场动画, 使用起来同样很简单。

```
//
//  UIView转场动画
//  TransitionAnimation
//
//  Created by Kenshin Cui on 14-3-12.
//  Copyright (c) 2014年 Kenshin Cui. All rights reserved.
//

#import "KCMainViewController.h"
#define IMAGE_COUNT 5

@interface KCMainViewController () {
    UIImageView *_imageView;
    int _currentIndex;
}

@end

@implementation KCMainViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //定义图片控件
    _imageView=[[UIImageView alloc]init];
    _imageView.frame=[UIScreen mainScreen].applicationFrame;
    _imageView.contentMode=UIViewContentModeScaleAspectFit;
    _imageView.image=[UIImage imageNamed:@"0.jpg"]; //默认图片
    [self.view addSubview:_imageView];
    //添加手势
    UISwipeGestureRecognizer *leftSwipeGesture=[[UISwipeGestureRecognizer alloc] initWithTarget:self, action:@selector(leftSwipe)];
    leftSwipeGesture.direction=UISwipeGestureRecognizerDirectionLeft;
    [self.view addGestureRecognizer:leftSwipeGesture];

    UISwipeGestureRecognizer *rightSwipeGesture=[[UISwipeGestureRecognizer alloc] initWithTarget:self, action:@selector(rightSwipe)];
    rightSwipeGesture.direction=UISwipeGestureRecognizerDirectionRight;
    [self.view addGestureRecognizer:rightSwipeGesture];
}
```

```
#pragma mark 向左滑动浏览下一张图片
-(void)leftSwipe:(UISwipeGestureRecognizer *)gesture{
    [self transitionAnimation:YES];
}

#pragma mark 向右滑动浏览上一张图片
-(void)rightSwipe:(UISwipeGestureRecognizer *)gesture{
    [self transitionAnimation:NO];
}

#pragma mark 转场动画
-(void)transitionAnimation:(BOOL)isNext{
    UIViewAnimationOptions option;
    if (isNext) {
        option=UIViewAnimationOptionCurveLinear|UIViewAnimationOptionTransitionFlipFromI
    }else{
        option=UIViewAnimationOptionCurveLinear|UIViewAnimationOptionTransitionFlipFromI
    }

    [UIView transitionWithView:_imageView duration:1.0 options:option animations:^(
        _imageView.image=[self getImage:isNext];
    ) completion:nil];
}

#pragma mark 取得当前图片
-(UIImage *)getImage:(BOOL)isNext{
    if (isNext) {
        _currentIndex=(_currentIndex+1)%IMAGE_COUNT;
    }else{
        _currentIndex=(_currentIndex-1+IMAGE_COUNT)%IMAGE_COUNT;
    }
    NSString *imageName=[NSString stringWithFormat:@"%i.jpg",_currentIndex];
    return [UIImage imageNamed:imageName];
}
@end
```

上面的转场动画演示中，其实仅仅有一个视图UIImageView做转场动画，每次转场通过切换UIImageView的内容而已。如果有两个完全不同的视图，并且每个视图布局都很复杂，此时要在这两个视图之间进行转场可以使用+ **(void)transitionFromView:(UIView *)fromView toView:(UIView *)toView duration:(NSTimeInterval)duration options:(UIViewAnimationOptions)options completion:(void (^)(BOOL finished))completion** **NS_AVAILABLE_IOS(4_0)**方法进行两个视图间的转场，需要注意的是默认情况下转出的视图会从父视图移除，转入后重新添加，可以通过**UIViewAnimationOptionShowHideTransitionViews**参数设置，设置此参数后转出的视图会隐藏（不会移除）转入后再显示。

注意：转场动画设置参数完全同基本动画参数设置；同直接使用转场动画不同的是使用UIView的装饰方法进行转场动画其动画效果较少，因为这里无法直接使用私有API。

您可能感兴趣的文章

[iOS开发系列--iOS程序开发概览](#)

[iOS开发系列--无限循环的图片浏览器](#)

[iOS开发系列--UITableView全面解析](#)

[iOS开发系列--视图切换](#)

[iOS开发系列--触摸事件、手势识别、摇晃事件、耳机线控](#)

[iOS开发系列--打造自己的“美图秀秀”](#)

[源代码下载](#)