iOS动画专题·UIView二维形变动画与CAAnimation核心动画（transform动画，基础，关键帧，组动画，路径动画，贝塞尔曲线）

1. iOS动画

总的来说，从涉及类的形式来看，iOS动画有：基于UIView的仿射形变动画，基于CAAnimation及其子类的动画，基于CG的动画。这篇文章着重总结前两种动画。

2. UIView动画

设置UIView形变动画有两种常见用到的属性，**.frame**，**.transform**，所以有的人也可以分别称之为：

- ① frame动画
- ② transform动画

这两种动画只需要在动画语法中适当的位置，基于UIView和CALayer的属性设置变化值即可。这种动画，不需要 调用核心动画CAAnimation里面的专用类和API。

其中，frame动画设置方式有限，必须确切地制定形变前后的frame，平移还好，特别是 旋转 的时候，只能通过数学知识计算出新的frame。这就得不偿失了。所以，更多的时候，当涉及一些frame，bounds，center的改变或是形变的时候可以用transform来取代frame。

2.1 设置UIView动画的两种语法形式

- **begin --- commit**

```
//偏移动画
 [UIView beginAnimations:@"move" context:nil];
 [UIView setAnimationDuration:2];
 [UIView setAnimationDelegate:self];
 imageContainView.frame = CGRectMake(80, 80, 200, 200);
 [label1 setBackgroundColor:[UIColor yellowColor]];
 [label1 setTextColor:[UIColor redColor]];
 [UIView commitAnimations];
```

- **animations block**

```
//缩放动画
  view.transform = CGAffineTransformIdentity;
  [UIView animateWithDuration:1.0f animations:^{
     view.transform = CGAffineTransformMakeScale(2.0f, 2.0f);
  }];
```

2.2 设置属性形变动画的两种类型

- UIView的 CGAffineTransform 类型属性：animatedView**.transform** 一般是View的旋转，拉伸移动等属性，是二维的，通常使用都是前缀CGAffineTransform的类。

```
CGAffineTransform transform =
CGAffineTransformScale(imageContainView.transform, 1.2, 1.2);
[UIView beginAnimations: @"scale"context: nil];
[UIView setAnimationDuration: 2];
[UIView setAnimationDelegate: self];
[imageView setTransform: transform];
[UIView commitAnimations];
```

- CALayer的CATransform3D 类型属性：animaView**.layer.transform** 通过.layer.transform 可以在3D模式下面的变化，通常使用的都是前缀为CATransform3D的类。

```
imageView.layer.transform =  CATransform3DIdentity;
[UIView animateWithDuration:1.0f animations:^{
      imageView.layer.transform = CATransform3DMakeScale(2.0, 2.0,
1.0);
}];
```

2.3 与动画相关的属性

2.3.1 UIView与动画相关的属性--与CGAffineTransform对应

下面是UIView的一些属性介绍:

```
@property(nonatomic) CGRect            frame;
@property(nonatomic) CGRect            bounds;      // default
bounds is zero origin, frame size. animatable
@property(nonatomic) CGPoint           center;      // center is
center of frame. animatable
@property(nonatomic) CGAffineTransform transform;   // default is
CGAffineTransformIdentity. animatable
@property(nonatomic) CGFloat           contentScaleFactor
NS_AVAILABLE_IOS(4_0);

@property(nonatomic,getter=isMultipleTouchEnabled) BOOL
multipleTouchEnabled __TVOS_PROHIBITED;   // default is NO
@property(nonatomic,getter=isExclusiveTouch) BOOL
exclusiveTouch __TVOS_PROHIBITED;         // default is NO
```

在实际开发中，使用场景：

- (1) 当涉及一些frame, bounds, center的改变或是形变的时候可以用 transform 来取代 frame。
- (2) 一般在实际开发中都是平移，旋转，缩放组合使用。

2.3.2 CALayer与动画相关的属性--与CATransform3D对应

下面是CALayer的一些属性介绍

```
//宽度和高度
@property CGRect bounds;

//位置(默认指中点，具体由anchorPoint决定)
@property CGPoint position;

//锚点(x,y的范围都是0-1)，决定了position的含义
@property CGPoint anchorPoint;

//背景颜色(CGColorRef类型)
@property CGColorRef backgroundColor;

//形变属性
```

```
@property CATransform3D transform;

//边框颜色(CGColorRef类型)
@property CGColorRef  borderColor;

//边框宽度
@property CGFloat borderWidth;

//圆角半径
@property CGFloat cornerRadius;

//内容(比如设置为图片CGImageRef)
@property(retain) id contents;
```

2.4 管理二维形变和三维形变的封装类：CGAffineTransform与CATransform3D

2.4.1 CGAffineTransform操作API

- CGAffineTransform结构体定义

```
struct CGAffineTransform {
  CGFloat a, b, c, d;
  CGFloat tx, ty;
};
```

它其实表示的是一个矩阵：$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$

因为最后一列总是是(0,0,1)，所以有用的信息就是前面两列。对一个view进行仿射变化就相当于对view上的每个点做一个乘法，结果就是：

$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix}$$

> a表示x水平方向的缩放，tx表示x水平方向的偏移 d表示y垂直方向的缩放，ty表示y垂直方向的偏移 如果b和c不为零的话，那么视图肯定发生了旋转，旋转角度这样计算：tan(angle) = b / a

如果这样：$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ 这个就是没有变化的最初的样子。

- CGAffineTransform操作API

```
//还原
CGAffineTransformIdentity

//位移仿射  ---- 理解为平移 (CGFloat tx,CGFloat ty)
CGAffineTransformMakeTranslation
CGAffineTransformTranslate
//旋转仿射 ---- 理解为旋转 (CGFloat angle)
CGAffineTransformMakeRotation
CGAffineTransformRotate
//缩放仿射  --- 理解缩放大小 (CGFloat sx, CGFloat sy)
CGAffineTransformMakeScale
CGAffineTransformScale
```

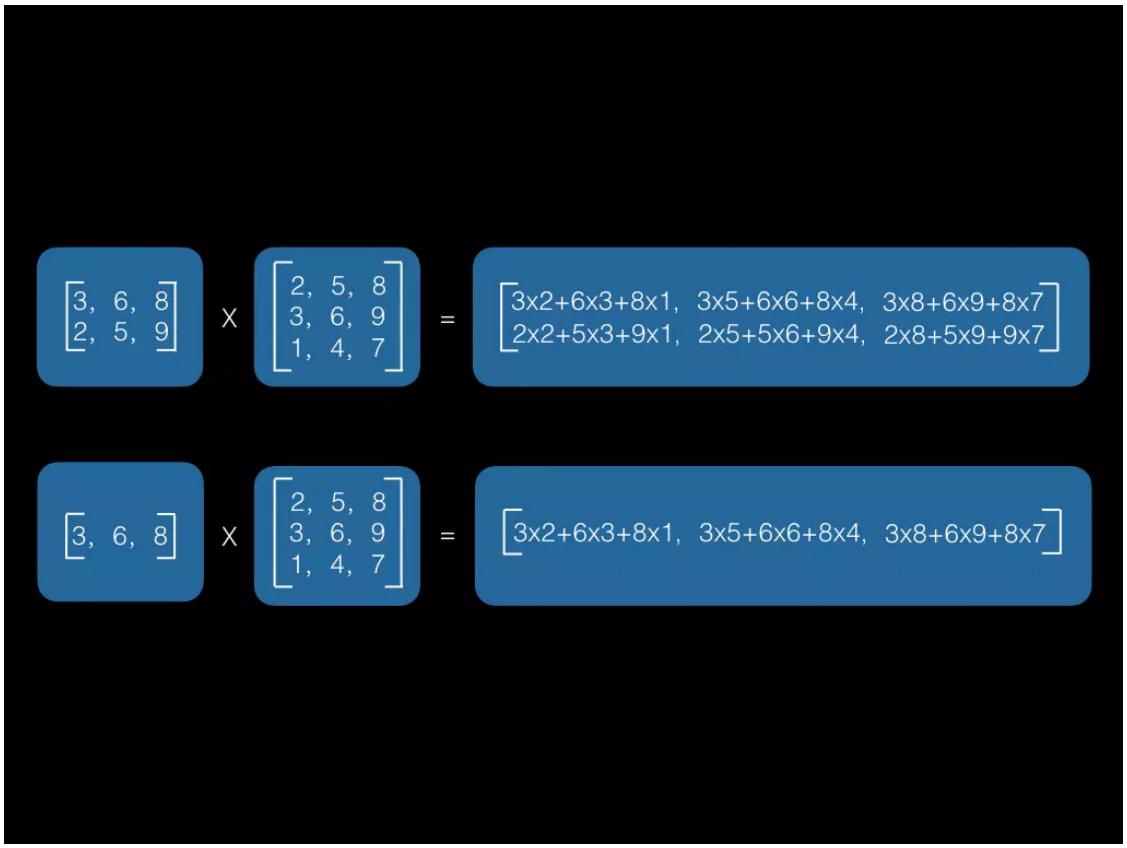
> CGAffineTransform操作的数学本质

- CGPoint转换公式

$$\begin{bmatrix} x & y & 1 \end{bmatrix} \times \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ t_x & t_y & 1 \end{bmatrix} = \begin{bmatrix} x' & y' & 1 \end{bmatrix}$$

CGPoint　　CGAffineTransform　　Transformed CGPoint

- 矩阵乘法运算原理演示



- 2.4.2 CATransform3D操作API

```
 CATransform3DIdentity

 //位移3D仿射 ==> (CGFloat tx, CGFloat ty, CGFloat tz)
CATransform3DMakeTranslation
CATransform3DTranslation
//旋转3D仿射 ==> (CGFloat angle, CGFloat x, CGFloat y, CGFloat z)
CATransform3DMakeRotation
CATransform3DRotation
//缩放3D仿射 ==>  (CGFloat angle, CGFloat x, CGFloat y, CGFloat z)
CATransform3DMakeScale
CATransform3DScale
//叠加3D仿射效果
CATransform3DConcat
//仿射基础3D方法，可以直接做效果叠加
CGAffineTransformMake (sx,shx,shy,sy,tx,ty)
//检查是否有做过仿射3D效果  == ((CATransform3D t))
CATransform3DIsIdentity(transform)
//检查2个3D仿射效果是否相同
CATransform3DEqualToTransform(transform1,transform2)
//3D仿射效果反转（反效果，比如原来扩大，就变成缩小）
```

```
CATransform3DInvert(transform)
```

- 2.4.3 CATransform3D与CGAffineTransform相互转换API

```
//将一个CGAffineTransform转化为CATransform3D
CATransform3D CATransform3DMakeAffineTransform (CGAffineTransform
m);
//判断一个CATransform3D是否可以转换为CAAffineTransformbool
CATransform3DIsAffine (CATransform3D t);
//将CATransform3D转换为CGAffineTransform
CGAffineTransform CATransform3DGetAffineTransform (CATransform3D
t);
```

2.5 “组合动画”与 CGAffineTransformConcat

2.5.1 连接设置多个属性组合成一个动画

连接设置两个以上属性的动画，可以先调用含有 formMake 的API，然后再调用只含 form 的API。例如，这样：

```
alertView.transform = CGAffineTransformMakeScale(.25, .25);
alertView.transform =
CGAffineTransformTranslate(alertView.transform, 0, 600);
```

2.5.2 利用CGAffineTransformConcat设置组合动画

另外，可以直接利用 CGAffineTransformConcat 来组合多种含有 formMake 的形变API。

```
CGAffineTransform viewTransform =
CGAffineTransformConcat(CGAffineTransformMakeScale(.25, .25),
CGAffineTransformMakeTranslation(0, 600));
alertView.transform = viewTransform;
```

关于组合3D形变也有相应的API --- CATransform3DConcat，关于3D形变下一篇会

专门介绍。

- CGAffineTransformConcat 组合多种形变动画小例子

```
- (void)viewDidAppear: (BOOL)animated {
    ///   初始化动画开始前label的位置
    CGFloat offset = label1.frame.size.height * 0.5;

    label1.transform = CGAffineTransformConcat(
      CGAffineTransformMakeScale(0, 0),
      CGAffineTransformTranslate(0, -offset)
    );
    label1.alpha = 0;
    [UIView animateWithDuration: 3. animations: ^ {
        ///   还原label1的变换状态并形变和偏移label2
        label1.transform = CGAffineTransformIdentifier;
        label1.transform = CGAffineTransformConcat(
          CGAffineTransformMakeScale(0, 0),
          CGAffineTransformTranslate(0, offset)
        );
        label1.alpha = 1;
        label2.alpha = 0;
    }];
}
```

- 组合变换的本质 CGAffineTransformConcat的数学本质是将括号内代表的若干变换的系数矩阵进行相乘。
- 另一种组合变换 基于已有的CGAffineTransform连续追加新的CGAffineTransform：

```
CGAffineTransform transform = CGAffineTransformIdentity;
transform = CGAffineTransformScale(transform, 0.5f, 0.5f);
transform = CGAffineTransformRotate(transform, 30.0f/180.0f*M_PI);
transform = CGAffineTransformTranslate(transform, 200.0f, 0.0f);
layerView.layer.affineTransform = transform;
```

**** 2.6 动画后将属性还原

当我们改变过一个view.transform属性或者view.layer.transform的时候需要恢复默认状态的话，记得先把他 们重置为：

```
view.transform = CGAffineTransformIdentity;
view.layer.transform = CATransform3DIdentity;
```

2.7 注意点: transform对frame的影响

官方文档上关于transform属性的说明：

> Changes to this property can be animated. However, if the transform property contains a non-identity transform, the value of the frame property is undefined and should not be modified. In that case, you can reposition the view using the center property and adjust the size using the bounds property instead.

如果在程序中改变了某个控件的transform，那么请不要使用这个控件的frame计算子控件 的布局，应该使用bounds+center代替。

3. CAAnimation核心动画

3.1 CAAnimation——所有动画对象的父类

- addAnimation

```
/**
 *  抖动效果
 */
-(void)shakeAnimation{
    CAKeyframeAnimation *anima = [CAKeyframeAnimation
animationWithKeyPath:@"transform.rotation"];//在这里
@"transform.rotation"==@"transform.rotation.z"
    NSValue *value1 = [NSNumber numberWithFloat:-M_PI/180*4];
    NSValue *value2 = [NSNumber numberWithFloat:M_PI/180*4];
    NSValue *value3 = [NSNumber numberWithFloat:-M_PI/180*4];
    anima.values = @[value1,value2,value3];
    anima.repeatCount = MAXFLOAT;

    [_demoView.layer addAnimation:anima forKey:@"shakeAnimation"];
}
```

3.2 CAAnimation继承结构

```
CAAnimation{
     CAPropertyAnimation{
            CABasicAnimation{
                    CASpringAnimation
            }
            CAKeyframeAnimation
     }
     CATransition
     CAAnimationGroup
}
```

是所有动画对象的父类，负责控制动画的持续时间和速度，是个抽象类，不能直接使用，应该使用它具体的子类

3.3 CAAnimation类的属性

带*号代表来自CAMediaTiming协议的属性）

- *duration：动画的持续时间
- *repeatCount：重复次数，无限循环可以设置HUGE_VALF或者MAXFLOAT
- *repeatDuration：重复时间
- removedOnCompletion：默认为YES，代表动画执行完毕后就从图层上移除，图形会恢复到动画执行前的状态。如果想让图层保持显示动画执行后的状态，那就设置为NO，不过还要设置fillMode为kCAFillModeForwards
- *fillMode：决定当前对象在非active时间段的行为。比如动画开始之前或者动画结束之后
- *beginTime：可以用来设置动画延迟执行时间，若想延迟2s，就设置为CACurrentMediaTime()+2，CACurrentMediaTime()为图层的当前时间
- timingFunction：速度控制函数，控制动画运行的节奏
- delegate：动画代理

3.4 几个重要属性值

removedOnCompletion属性值

> CAAnimation——动画填充模式

默认为YES，代表动画执行完毕后就从图层上移除，图形会恢复到动画执行前的状态。如果想让图层保持显示动画执行后的状态，那就设置为NO，不过还要设置fillMode为kCAFillModeForwards

fillMode属性值

> CAAnimation——控制恢复到动画执行前 要想fillMode有效，最好设置removedOnCompletion = NO

- 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 在动画开始前，只需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。
- kCAFillModeBoth 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

如果 fillMode = kCAFillModeForwards 同时 removedOnComletion = NO ，那么在动画执行完毕后，图层会保持显示动画执行后的状态。但在实质上，图层的属性值还是动画执行前的初始值，并没有真正被改变。

timingFunction属性值

> CAAnimation——动画速度控制函数

- kCAMediaTimingFunctionLinear（线性）：匀速，给你一个相对静态的感觉
- kCAMediaTimingFunctionEaseIn（渐进）：动画缓慢进入，然后加速离开
- kCAMediaTimingFunctionEaseOut（渐出）：动画全速进入，然后减速的到达目的地
- kCAMediaTimingFunctionEaseInEaseOut（渐进渐出）：动画缓慢的进入，中间加速，然后减速的到达目的地。这个是默认的动画行为。

4. CABasicAnimation

基本动画，是CAPropertyAnimation的子类

4.1 特别属性说明:

- keyPath: 要改变的属性名称(传字符串)

- fromValue: keyPath相应属性的初始值
- toValue: keyPath相应属性的结束值

4.2 keyPath可以是哪些值

```
CATransform3D{
    //rotation旋转
    transform.rotation.x
    transform.rotation.y
    transform.rotation.z

    //scale缩放
    transform.scale.x
    transform.scale.y
    transform.scale.z

    //translation平移
    transform.translation.x
    transform.translation.y
    transform.translation.z
}

CGPoint{
    position
    position.x
    position.y
}

CGRect{
    bounds
    bounds.size
    bounds.size.width
    bounds.size.height

    bounds.origin
    bounds.origin.x
    bounds.origin.y
}

property{
    opacity
    backgroundColor
    cornerRadius
    borderWidth
    contents
```

```
    Shadow{
        shadowColor
        shadowOffset
        shadowOpacity
        shadowRadius
    }
}
```

4.3 举例

- 缩放动画 -- transform.scale

```
//心脏缩放动画
    CABasicAnimation *scaleAnimation = [CABasicAnimation
animationWithKeyPath:@"transform.scale"]; //选中的这个keyPath就是缩放
    scaleAnimation.fromValue = [NSNumber numberWithDouble:0.5]; //一
开始时是0.5的大小
    scaleAnimation.toValue = [NSNumber numberWithDouble:1.5];  //结
束时是1.5的大小
    scaleAnimation.duration = 1; //设置时间
    scaleAnimation.repeatCount = MAXFLOAT; //重复次数
    [_heartImageView.layer addAnimation:scaleAnimation
forKey:@"CQScale"]; //添加动画
```

- 旋转动画 -- transform.rotation.z

```
    //风车旋转动画
    CABasicAnimation *rotationAnimation = [CABasicAnimation
animationWithKeyPath:@"transform.rotation.z"];
    rotationAnimation.fromValue = [NSNumber numberWithDouble:0.f];
    rotationAnimation.toValue = [NSNumber numberWithDouble:2 *
M_PI];
    rotationAnimation.duration = 2.f;
    rotationAnimation.repeatCount = MAXFLOAT;
    [_fengcheImageView.layer addAnimation:rotationAnimation
forKey:@"CQRotation"];
```

- 平移动画 -- position.x/position.y

```
    //平移动画
    CABasicAnimation *positionAnimation = [CABasicAnimation
animationWithKeyPath:@"position.x"];
    positionAnimation.fromValue = [NSNumber numberWithDouble:0.f];
    positionAnimation.toValue = [NSNumber
numberWithDouble:SCREEN_WIDTH];
    positionAnimation.duration = 2;
    positionAnimation.repeatCount = MAXFLOAT;
    [_arrowImageView.layer addAnimation:positionAnimation
forKey:@"CQPosition"];
```

5. CAKeyframeAnimation关键帧动画

5.1 参数数组形式

```
    //根据values移动的动画
    CAKeyframeAnimation *catKeyAnimation = [CAKeyframeAnimation
animationWithKeyPath:@"position"];
    CGPoint originalPoint = self.catImageView.layer.frame.origin;
    CGFloat distance =  50;
    NSValue *value1 = [NSValue
valueWithCGPoint:CGPointMake(originalPoint.x + distance,
originalPoint.y + distance)];
    NSValue *value2 = [NSValue
valueWithCGPoint:CGPointMake(originalPoint.x + 2 * distance,
originalPoint.y + distance)];
    NSValue *value3 = [NSValue
valueWithCGPoint:CGPointMake(originalPoint.x + 2 * distance,
originalPoint.y +  2 * distance)];
    NSValue *value4 = [NSValue valueWithCGPoint:originalPoint];
    catKeyAnimation.values = @[value4, value1, value2, value3,
value4];
    catKeyAnimation.duration = 2;
    catKeyAnimation.repeatCount = MAXFLOAT;
    catKeyAnimation.removedOnCompletion = NO;
    [self.catImageView.layer addAnimation:catKeyAnimation
forKey:nil];
```

5.2 path形式

```
    //指定path
    UIBezierPath *path = [UIBezierPath
bezierPathWithOvalInRect:CGRectMake(0, 200, 200, 200)];
    //指定path的动画
    UIBezierPath *path2 = [UIBezierPath bezierPath];
    [path2 moveToPoint:CGPointMake(100, 100)];
    [path2 addLineToPoint:CGPointMake(100, 200)];
    [path2 addLineToPoint:CGPointMake(200, 200)];
    [path2 addLineToPoint:CGPointMake(200, 100)];
    [path2 addLineToPoint:CGPointMake(100, 100)];
    CAKeyframeAnimation *penguinAnimation = [CAKeyframeAnimation
animationWithKeyPath:@"position"];
    penguinAnimation.path = path2.CGPath;
    penguinAnimation.duration = 2;
    penguinAnimation.repeatCount = MAXFLOAT;
    penguinAnimation.removedOnCompletion = NO;
    [self.penguinImageView.layer addAnimation:penguinAnimation
forKey:nil];
```

6. 组动画

6.1 组动画

上面单一动画的情况在实际开发中实际比较少，更多的时候是组合这些动画：创建不同类型的动画对象，设置好它们的参数，然后把这些动画对象存进数组，传进组动画对象的animations属性中去。

6.2 示例

```
//创建组动画
    CAAnimationGroup *animationGroup = [CAAnimationGroup
animation];
    animationGroup.duration = 3;
    animationGroup.repeatCount = MAXFLOAT;
    animationGroup.removedOnCompletion = NO;
    /* beginTime 可以分别设置每个动画的beginTime来控制组动画中每个动画的触发
时间，时间不能够超过动画的时间，默认都为0.f */

    //缩放动画
    CAKeyframeAnimation *animation1 = [CAKeyframeAnimation
animationWithKeyPath:@"transform.scale"];
```

```
    animation1.values = @[[NSNumber numberWithFloat:1.0],[NSNumber
numberWithFloat:0.5],[NSNumber numberWithFloat:1.5],[NSNumber
numberWithFloat:1.0]];
    animation1.beginTime = 0.f;

    //按照圆弧移动动画
    CAKeyframeAnimation *animation2 = [CAKeyframeAnimation
animationWithKeyPath:@"position"];
    UIBezierPath *bezierPath = [UIBezierPath bezierPath];
    [bezierPath moveToPoint:CGPointMake(300, 200)];
    [bezierPath addQuadCurveToPoint:CGPointMake(200, 300)
controlPoint:CGPointMake(300, 300)];
    [bezierPath addQuadCurveToPoint:CGPointMake(100, 200)
controlPoint:CGPointMake(100, 300)];
    [bezierPath addQuadCurveToPoint:CGPointMake(200, 100)
controlPoint:CGPointMake(100, 100)];
    [bezierPath addQuadCurveToPoint:CGPointMake(300, 200)
controlPoint:CGPointMake(300, 100)];
    animation2.path = bezierPath.CGPath;
    animation2.beginTime = 0.f;

    //透明度动画
    CABasicAnimation *animation3 = [CABasicAnimation
animationWithKeyPath:@"opacity"];
    animation3.fromValue = [NSNumber numberWithDouble:0.0];
    animation3.toValue = [NSNumber numberWithDouble:1.0];
    animation3.beginTime = 0.f;

    //添加组动画
    animationGroup.animations = @[animation1,
animation2,animation3];
    [_penguinImageView.layer addAnimation:animationGroup
forKey:nil];
```

7. 贝塞尔曲线

前面关键帧动画章节提到了贝塞尔曲线，这个曲线很有用，在iOS开发中有两种形式可用：CGMutablePathRef和UIBezierPath，均可以通过制定控制点数组的形式唯一确定曲线，也可以通过矩形内切椭圆唯一确定曲线。下面是两者的例子：

7.1 CGMutablePathRef

- 通过 关键点曲线连接 唯一确定

```
    // 贝塞尔曲线关键帧
    // 设置路径，绘制贝塞尔曲线
    CGMutablePathRef path = CGPathCreateMutable();
    CGPathMoveToPoint(path, NULL, 200, 200); // 起始点
    CGPathAddCurveToPoint(path, NULL, 100, 300, 300, 500, 200,
600);
    // CGPathAddCurveToPoint(path, NULL, 控制点1.x, 控制点1.y, 控制点
2.x, 控制点2.y, 终点.x, 终点.y);
    // 设置path属性
    keyframeAnimation.path = path;
    CGPathRelease(path);

    // 设置其他属性
    keyframeAnimation.duration = 4;
    keyframeAnimation.beginTime = CACurrentMediaTime() + 1; // 设置
延迟2秒执行，不设置这个属性，默认直接执行
    // 3. 添加动画到图层，会自动执行
    [_layer addAnimation:keyframeAnimation
forKey:@"GGKeyframeAnimation"];
```

7.2 UIBezierPath

- 通过 矩形内切椭圆 唯一确定

```
    CAKeyframeAnimation *anima = [CAKeyframeAnimation
animationWithKeyPath:@"position"];
    UIBezierPath *path = [UIBezierPath
bezierPathWithOvalInRect:CGRectMake(SCREEN_WIDTH/2-100,
SCREEN_HEIGHT/2-100, 200, 200)];
    anima.path = path.CGPath;
    anima.duration = 2.0f;
    [_demoView.layer addAnimation:anima forKey:@"pathAnimation"];
```

- 通过 关键点直线连接 唯一确定

```
-(void)pathAnimation2{

    //创建path
    UIBezierPath *path = [UIBezierPath bezierPath];
    //设置线宽
```

```
    path.lineWidth = 3;
    //线条拐角
    path.lineCapStyle = kCGLineCapRound;
    //终点处理
    path.lineJoinStyle = kCGLineJoinRound;
    //多条直线
    [path moveToPoint:(CGPoint){0, SCREEN_HEIGHT/2-50}];
    [path addLineToPoint:(CGPoint){SCREEN_WIDTH/3, SCREEN_HEIGHT/2-
50}];
    [path addLineToPoint:(CGPoint){SCREEN_WIDTH/3,
SCREEN_HEIGHT/2+50}];
    [path addLineToPoint:(CGPoint){SCREEN_WIDTH*2/3,
SCREEN_HEIGHT/2+50}];
    [path addLineToPoint:(CGPoint){SCREEN_WIDTH*2/3,
SCREEN_HEIGHT/2-50}];
    [path addLineToPoint:(CGPoint){SCREEN_WIDTH, SCREEN_HEIGHT/2-
50}];
//    [path closePath];

    CAKeyframeAnimation *anima = [CAKeyframeAnimation
animationWithKeyPath:@"position"];
    anima.path = path.CGPath;
    anima.duration = 2.0f;
    anima.fillMode = kCAFillModeForwards;
    anima.removedOnCompletion = NO;
    anima.timingFunction = [CAMediaTimingFunction
functionWithName:kCAMediaTimingFunctionEaseInEaseOut];
    [_demoView.layer addAnimation:anima forKey:@"pathAnimation"];
}
```