
# Core Animation:

## Simplified Animation Techniques for Mac and iPhone Development

## 第二部分

## 核心动画基础

### 第三章

### 关键帧动画

版本 1.0

翻译时间：2012-11-05

DevDiv 翻译：animeng
DevDiv 校对：symbian_love BeyondVincent(破船)
DevDiv 编辑：BeyondVincent(破船)

# 写在前面

目前，移动开发被广大的开发者们看好，并大量的加入移动领域的开发。

鉴于以下原因：

- 国内的相关中文资料缺乏
- 许多开发者对 E 文很是感冒
- 电子版的文档利于技术传播和交流

DevDiv.com 移动开发论坛特此成立了翻译组，翻译组成员具有丰富的移动开发经验和英语翻译水平。组员们利用业余时间，把一些好的相关英文资料翻译成中文，为广大移动开发者尽一点绵薄之力，希望能对读者有些许作用，在此也感谢组员们的辛勤付出。

## 关于 DevDiv

DevDiv 已成长为国内最具人气的综合性移动开发社区

更多相关信息请访问 DevDiv 移动开发论坛。

## 技术支持

首先 DevDiv 翻译组对您能够阅读本文以及关注 DevDiv 表示由衷的感谢。

在您学习和开发过程中，或多或少会遇到一些问题。DevDiv 论坛集结了一流的移动专家，我们很乐意与您一起探讨移动开发。如果您有什么问题和技术需要支持的话，请访问网站 www.devdiv.com 或者发送邮件到 BeyondVincent@DevDiv.com，我们将尽力所能及的帮助你。

## 关于本文的翻译

感谢 animeng 对本文的翻译，同时非常感谢 symbian_love 和 BeyondVincent(破船)在百忙中抽出时间对翻译初稿的认真校验。才使本文与读者尽快见面。由于书稿内容多，我们的知识有限，尽管我们进行了细心的检查，但是还是会存在错误，这里恳请广大读者批评指正，并发送邮件至 BeyondVincent@devdiv.com，在此我们表示衷心的感谢。

# 推荐资源

iOS

iOS 5 Programming Cookbook 中文翻译各章节汇总

iOS6 新特征：参考资料和示例汇总

Android

DEVDIV 原创 ANDROID 学习系列教程实例

Windows Phone

Windows Phone 8 新特征讲义与示例汇总

Windows 8

Building Windows 8 apps with XAML and C#中文翻译全部汇总

Building Windows 8 apps with HTML5 and JavaScript 中文翻译汇总

Windows 8 Metro 开发书籍汇总

Windows 8 Metro App 开发 Step by Step

其它

DevDiv 出版作品汇总

# 目录

写在前面 2
关于 DevDiv 2
技术支持 2
关于本文的翻译 2
推荐资源 3
目录 4
本书翻译贴各章汇总 5
Core Animation 中文翻译各章节汇总 5
第一部分 核心动画开篇 5
1.1.1. 第一章 什么是核心动画 5
1.1.2. 第二章 我们可以和应该做哪些动画 5
1.1.3. 第三章 Core Animation 中文翻译_第三章_基础动画 5
第 4 章 关键帧动画 6
4.1.1. 随着时间改变值 6
4.1.2. 指定关键帧目的地的两个方法 6
4.1.3. 从基础动画到关键帧动画 8
4.1.4. 关键帧动画的时间 9
4.1.5. 关键帧动画的 UI 提示 11
4.1.6. 总结 16

# 本书翻译贴各章汇总

Core Animation 中文翻译各章节汇总

## 第一部分 核心动画开篇

1.1.1. 第一章 什么是核心动画

1.1.2. 第二章 我们可以和应该做哪些动画

1.1.3. 第三章 Core Animation 中文翻译_第三章_基础动画

# 第 4 章　关键帧动画

核心动画为你提供了一个完全控制动画的方法。这种控制是以关键帧动画（keyframe animation）形式存在的。关键帧动画是你能够，给动画的每个主要步骤分配你想要的值，剩下的就帮你填充。

描述关键帧动画的这个术语是来自于电影工业。随着电影中的动画发展到计算机里，在关键帧之间填充成为创建平滑动画必不可少的部分，上述过程的这个概念就叫补间动画（tweening)。核心动画同样没有什么不同的。你在动画中指定了关键帧，那么核心动画就帮你处理补间动画的过程。它可以基于你在动画中指定的属性，计算出在每个关键帧之间还有什么需要做，然后就插入需要的值。它相当的方便，可以使动画的代码任务变得非常的简单。

这章中，我们将讨论如何从基础动画过度到关键帧动画中。它如此的简单，因此过度应该也是很顺利的。我们也介绍怎么给你层做旋转，和如何通过画一些路径来给层增加内容。

## 4.1.1.　随着时间改变值

核心动画提供了一系列你可以做动画的属性。关键帧动画是一个可以随时改变这些属性的过程。你可以想象下，关键帧动画就是一组动画捆扎在一起，然后按顺序运行。

对于核心动画来说，插值（和电影中的补间动画相似）利用每个关键帧之间的属性值，来计算出附近关键帧的开始值和结束值。换个角度考虑，你可以想象成一个棒球内场中的垒。每一个垒就是一个关键帧，跑手知道，当击球时，如果他想得到一分，他必须跑到第一个垒，然后第二个，第三个，最后到主垒（这个过程中，他不能被赶出去）。他不必担心每两个垒之间的步骤，但是他必须跑到每一个垒。简单的说，关键帧中的动画就是按照顺序跑到每一个目的点（关键帧），直到所有的目的地都到达。

当然了，在棒球内场这种情况下，你的最后一帧的值应该和第一帧一样。

## 4.1.2.　指定关键帧目的地的两个方法

当你想要使一个属性做动画时，要么你指定一序列的属性值，让动画按顺序到达这些序列值，或者你指定一个核心图形的路径：

一个系列值数组很容易被使用一些东西指定，例如背景颜色的改变值，边框或者边框的宽度，还有框架的大小，甚至 CGImage(内容的图片)都能被作为关键路径。你通过指定一列 CATransform3D 对象给数组，来给旋转（transform）这个属性做动画。

使用核心图形路径，可以在屏幕上移动物体从一点到另一点。无论什么时候，只要 CGPoint 属于属性的类型，一个核心图像路径很可能就是正确的选择。你可以使用 anchorPoint 和 position 这个两个属性来使路径做动画。

按照优先级，一个路径会覆盖一组值。这就意味着如果你通过-setPath 指定了一个路径，通过-setValues 调用的任何值都将被忽略。列表 4-1 和 4-2 分别演示了怎么来使用核心图像路径和一组值。清单看起来相似，但是结果却完全不同。在清单 4-1 中的动画，会引起层按照抛物线的路径跳动。而清单 4-2 会引起层按照直角的锯齿跳动。

```
- (CAAnimation*)pathAnimation;
{
CGMutablePathRef path = CGPathCreateMutable();
CGPathMoveToPoint(path,NULL,50.0,120.0);
CGPathAddCurveToPoint(path,NULL,50.0,275.0,150.0,275.0,150.0,120.0);
CGPathAddCurveToPoint(path,NULL,150.0,275.0,250.0,275.0,250.0,120.0);
CGPathAddCurveToPoint(path,NULL,250.0,275.0,350.0,275.0,350.0,120.0);
CGPathAddCurveToPoint(path,NULL,350.0,275.0,450.0,275.0,450.0,120.0);
CAKeyframeAnimation *
animation = [CAKeyframeAnimation
animationWithKeyPath:@”position”];
[animation setPath:path];
```

```
[animation setDuration:3.0];
[animation setAutoreverses:YES];
CFRelease(path);
return animation;
}
```

清单 4-1 路径动画

```
- (CAAnimation*)valuesAnimation;
{
NSPoint pt0 = NSMakePoint(50.0, 120.0);
NSPoint pt1 = NSMakePoint(50.0, 275.0);
NSPoint pt2 = NSMakePoint(150.0, 275.0);
NSPoint pt3 = NSMakePoint(150.0, 120.0);
NSPoint pt4 = NSMakePoint(150.0, 275.0);
NSPoint pt5 = NSMakePoint(250.0, 275.0);
NSPoint pt6 = NSMakePoint(250.0, 120.0);
NSPoint pt7 = NSMakePoint(250.0, 275.0);
NSPoint pt8 = NSMakePoint(350.0, 275.0);
NSPoint pt9 = NSMakePoint(350.0, 120.0);
NSPoint pt10 = NSMakePoint(350.0, 275.0);
NSPoint pt11 = NSMakePoint(450.0, 275.0);
NSPoint pt12 = NSMakePoint(450.0, 120.0);
NSArray *values = [NSArray arrayWithObjects:
[NSValue valueWithPoint:pt0],
[NSValue valueWithPoint:pt1],
[NSValue valueWithPoint:pt2],
[NSValue valueWithPoint:pt3],
[NSValue valueWithPoint:pt4],
[NSValue valueWithPoint:pt5],
[NSValue valueWithPoint:pt6],
[NSValue valueWithPoint:pt7],
[NSValue valueWithPoint:pt8],
[NSValue valueWithPoint:pt9],
[NSValue valueWithPoint:pt10],
[NSValue valueWithPoint:pt11],
[NSValue valueWithPoint:pt12],
nil];
CAKeyframeAnimation
*animation = [CAKeyframeAnimation
animationWithKeyPath:@”position”];
[animation setValues:values];
[animation setDuration:3.0];
[animation setAutoreverses:YES];
return animation;
}
```

清单 4-2 值动画

在图 4-1 中你会看到两种模式。你可以通过给数组增加更多的点来接近的模式,也可以用路径获得的模式，但是是用路径是更加的方便。

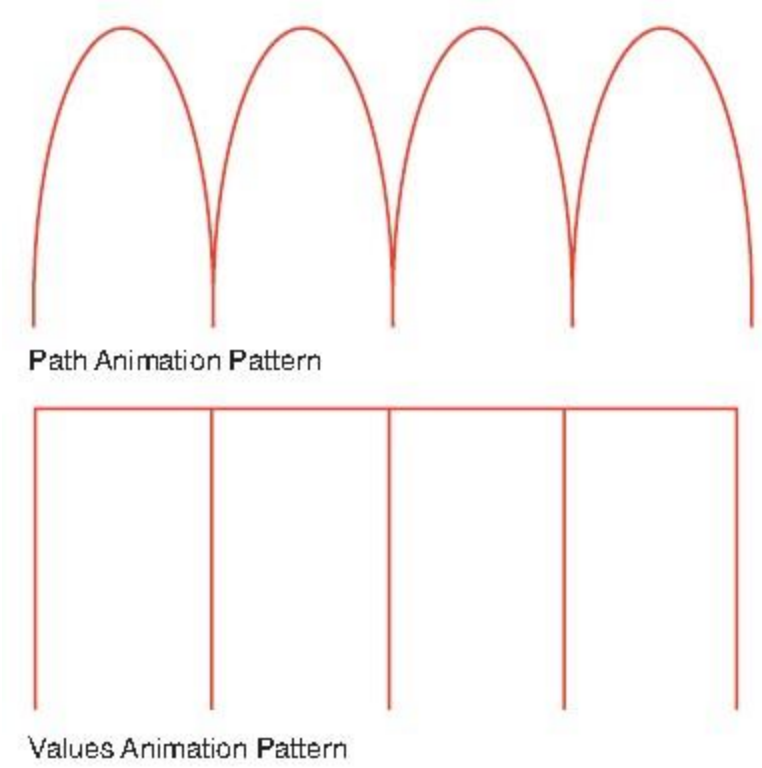



图 4-1 动画模式，路径 vs 值

图 4-2 展示了 2 各层以前以后的运动：绿色的是弧线运动，路径是被清单 4-1 指定，而直角锯齿模式是红色的层按照清单 4-2 指定的每个值运动。

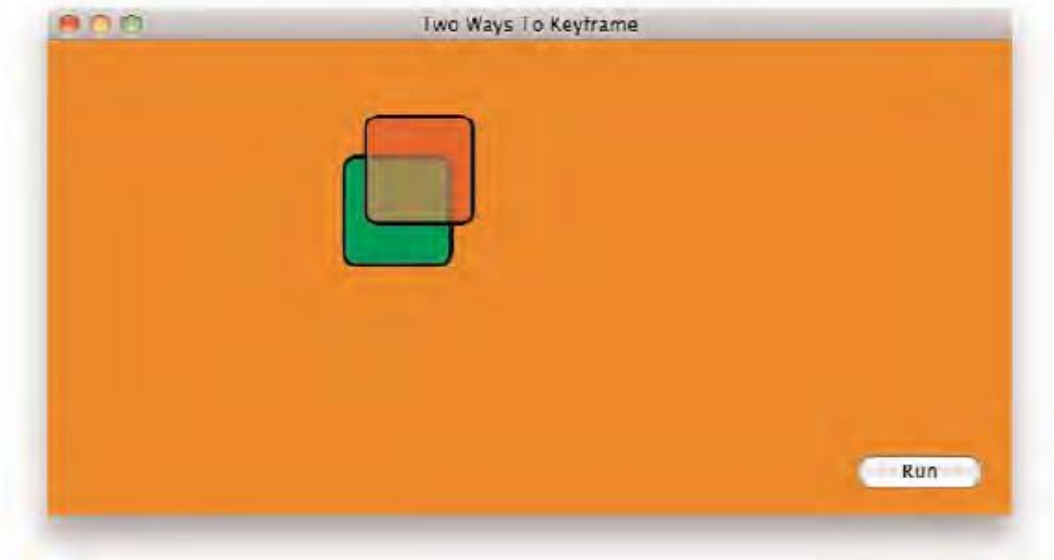



图 4-2 两个关键帧的方式的 demo

### 4.1.3. 从基础动画到关键帧动画

一个基础动画仅仅需要你指定一个开始的值和一个结束的值，或者目前的值，关键帧动画可以使你指定：
每个关键帧的一个数组值，例如，你可以给一个 CGColorRef 对象的一个数组，用来改变背景的颜色。
在 0.0 和 1.0 之间的一个数组值，用来给每个帧指定总时间中所占的百分比的时间。
一个时间功能对象的数组（CAMediaTimingFunction），用来指定每帧动画的时间步调。这个区域会使用一个预设的值来指定，例如 kCAMediaTimingFunctionEaseIn，动画出来时慢，然后加速。
关键帧动画真的相当的简单。首先，你需要考虑的是，成功的使用关键帧动画，你需要做的基本步骤：

1.决定你想要做动画的属性（例如，框架，背景，锚点，位置，边框，等等）
2.在动画对象值的区域中，指定开始，结束，和中间的值。这些都是你的关键帧（看清单 4-2）
3.使用 duration 这个字段指定动画的时间
4.通常来讲，通过使用 times 这个字段，来给每帧动画指定一个时间。如果你没有指定这些，核心动画就会通过你在 values 这个字段指定的值分割出时间段。
5.通常，指定时间功能来控制步调。
这些都是你需要做的。你创建你的动画和增加他们到层中。调用-addAnimation 就开始了动画。

### 4.1.4. 关键帧动画的时间

核心动画提供了一个高标准的间隔尺寸，来指定你的动画如何播放。默认情况下，一帧动画的播放，分割的时间是动画的总时间除以帧数减去一。你可以通过下面的公式决定每帧动画的时间：总时间/(总帧数-1)。例如，如果你指定了一个 5 帧，10 秒的动画，那么每帧的时间就是 2.5 秒钟：10/(5-1)=2.5。你可以做更多的控制通过使用 keyTimes 关键字，你可以给每帧动画指定总时间之内的某个时间点。5 帧动画，每帧动画占总时间的 25%。如果你使用 keyTimes 关键字设置动画的时间，本质上和默认的动画一样的，代码如清单 4-3.

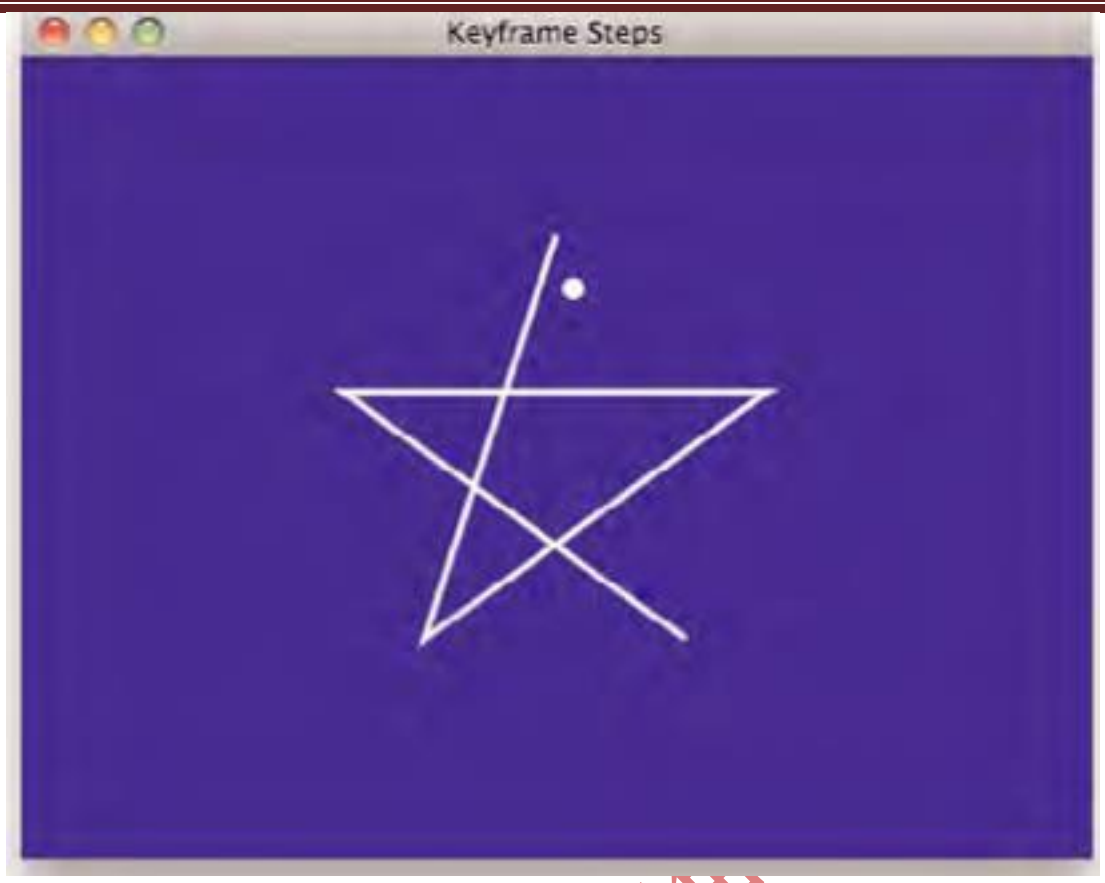
```
[animation setCalculationMode:kCAAnimationLinear];
[animation setKeyTimes:
[NSArray arrayWithObjects:
[NSNumber numberWithFloat:0.0],
[NSNumber numberWithFloat:0.25],
 [NSNumber numberWithFloat:0.50],
[NSNumber numberWithFloat:0.75],
[NSNumber numberWithFloat:1.0], nil]];
```

清单 4-3 显示地设置关键时间模拟默认动画

在清单 4-3 中，你可能注意到了我们指定了 5 个值，可能不匹配公式中的值。然而，如果你明白值(0.0)是简单的开始值，那么就会明白我们仅仅指定了 4 个目的时间，这样就匹配前面所说的公式了。
就是在清单 4-3 中，你不指定关键时间，你看到的动画也是一致的。这里仅仅是为了展示如何来指定关键时间。你可以增加或者减少每个关键帧的时间。需要记得序列中的每个值，后一个要比前一个大，并且最后一个不能超过 1.0。帧的总数和时间给核心动画提供了足够的信息，来插入每个值，简单的通过总时间除以总帧数减去 1.
根据 calculationMode 关键字段，关键时间是如清单 4-3 中被使用。当你指定步调(kCAAnimationPaced)，关键时间就会被忽略。这就像你没指定关键时间一样。当你指定了一个线性(kCAAnimationLinear)，被插入的每个关键帧上的时间，那意味着你会在帧之间看到每步动画。这就是我们上一章讨论的补间动画。如果你指定作为离散的(kCAAnimationDiscrete)，仅仅展示关键帧，在每帧之间没有任何插入。如果你移动一个层到窗口上随机的一点，事实上，你不会看到层移动到每个点。相反，你会看到层跳到每个点，当关键帧的时间到达时。

监视关键帧的目的

为了进一步理解怎么来应用关键时间的公式，进入合作网站 informit.com/coreframeworks 并且打开叫做关键帧步调的示例程序。你会注意到我们增加了定时器，在动画的每个阶段开启用来展示在动画中到达的点。在关键帧步调的例子代码中，我们使一个点在一个五角星的边上做动画，如图表 4-3 展示。随着动画到达每个点，我们都会从先前的点到下一个点画上一段路径；这些都是有定时器做的。首先，这里要决定我们需要多少关键帧并且决定我们需要多少时间。




图表 4-3 关键帧步调演示程序

在示例代码中，我们指定了动画的执行时间为 10 秒钟。为了决定每两个点之间做动画要花费的时间，我们要应用基于关键帧数量和时间的公式。包含动画的开始和结束点，总共有 6 个关键帧。我们应用下面的动画时间公式：时间段/(关键帧数-1) = 每帧的时间 10/(6-1) = 2.0.

在清单 4-4 中，注意到定时器在动画期间每 2 秒钟使用一次。这就造成了，当每个目的地到达时，路径就是会被更新和绘制来展示走的路径。

```
- (IBAction)startAnimation:(id)sender; {
if( starPath ) {
// Reset our starPath CGMutablePathRef if // it has already run before CFRelease(starPath);
[self createPath];
[[[window contentView] layer] setNeedsDisplay]; }
// Define a path for the star shape.
CGMutablePathRef path = CGPathCreateMutable(); CGPathMoveToPoint(path,NULL,240.0, 280.0);
CGPathAddLineToPoint(path, NULL, 181.0, 99.0); CGPathAddLineToPoint(path, NULL, 335.0, 210.0);
CGPathAddLineToPoint(path, NULL, 144.0, 210.0); CGPathAddLineToPoint(path, NULL, 298.0, 99.0);
CGPathCloseSubpath(path);
CAKeyframeAnimation *animation = [CAKeyframeAnimation
animationWithKeyPath:@"position"]; [animation setDuration:10.0f]; [animation setDelegate:self];
// Set the animation's path
[animation setPath:path]; // Release the path CFRelease(path);
[dotLayer addAnimation:animation forKey:@"position"];
[NSTimer scheduledTimerWithTimeInterval:2.0 target:self
selector:@selector(legOne:) userInfo:nil
repeats:NO];
[NSTimer scheduledTimerWithTimeInterval:4.0
target:self selector:@selector(legTwo:) userInfo:nil
repeats:NO];
[NSTimer scheduledTimerWithTimeInterval:6.0 target:self
selector:@selector(legThree:) userInfo:nil
```

```
repeats:NO];
[NSTimer scheduledTimerWithTimeInterval:8.0 target:self
selector:@selector(legFour:) userInfo:nil
repeats:NO];
[NSTimer scheduledTimerWithTimeInterval:10.0 target:self
selector:@selector(legFive:) userInfo:nil
repeats:NO];
// Tell the root layer to call drawLayer
[[[window contentView] layer] setNeedsDisplay]; }
```

清单 4-4 安装动画和创建定时器监视动画的过程

当每个时间到达时，另一个点就会被增加到 starpath 路径中，然后根层就会更新显示，如清单 4-5 所示。最后调用 legFive 方法，关闭路径。

```
- (void)legOne:(id)sender {
CGPathAddLineToPoint(starPath, NULL, 181.0, 99.0);
[[[window contentView] layer] setNeedsDisplay]; }
- (void)legTwo:(id)sender {
CGPathAddLineToPoint(starPath, NULL, 335.0, 210.0);
[[[window contentView] layer] setNeedsDisplay]; }
- (void)legThree:(id)sender {
CGPathAddLineToPoint(starPath, NULL, 144.0, 210.0);
[[[window contentView] layer] setNeedsDisplay]; }
- (void)legFour:(id)sender {
CGPathAddLineToPoint(starPath, NULL, 298.0, 99.0);
[[[window contentView] layer] setNeedsDisplay]; }
- (void)legFive:(id)sender {
CGPathCloseSubpath(starPath);
[[[window contentView] layer] setNeedsDisplay]; }
```

清单 4-5 每次定时器完成时就增加新的点到路径

当我们在根层上调用-setNeedDisplay 时，-drawLayer 被调用和根据目前状态的路径画到层上，如清单 4-6 所示。

```
- (void)drawLayer:(CALayer *)layer inContext:(CGContextRef)context
{
if( layer == [[window contentView] layer] ) {
CGColorRef white =
CGColorCreateGenericRGB(1.0, 1.0, 1.0, 1.0); CGContextSetStrokeColorWithColor(context, white); CFRelease(white);
CGContextBeginPath(context); CGContextAddPath(context, starPath);
CGContextSetLineWidth(context, 3.0); CGContextStrokePath(context);
} }
```

清单 4-6 每次定时器开启时绘制路径

清单 4-6 每次调用的绘制都是一样的，但是路径在定时器每次调用时，都已经被扩展，从一点到下一个点，从而能成功的绘制。

### 4.1.5. 关键帧动画的 UI 提示

核心动画本来是为 iphone 平台写的，但是在 mac OS X 平台也是可用的。如果你曾经看到过 iphone 和 itouch 中的行为，你就会目击到核心动画，因为在这个平台下几乎所有的 UI 行为都有核心动画。你是否使用过下面的带转孔的按钮（可以使试图从一边滑出或者滑入），或者你关闭你目前的应用程序，然后返回到主屏幕时，图标开始做从两边同时做的动画，你就可以看到核心动画的效果。

当你长按一个应用程序图标时，一个有趣的动画会发生。这会引起所有的应用程序图标扭动，就像我们视图要拿走他们一样。在这种编辑状态下，你可以移动应用程序的图标，以便于组织成你需要的情况(看图 4-4)。

另一个你需要注意的是在应用程序图标左上角的删除按钮，你可以通过它从 iphone 或者 touch 中删除应用程序。如果你按下了 home 键，所有的图标都将停止抖动，然后返回到正常状态。


图 4-4

这个用户体验可以帮助用户知道 iphone 和 ipod 是在不同的状态，然后你就可以改变一些东西。这不能立即明显的表明那东西是什么。然而，一旦你开始点击并且拖拽，它就很明显出现你想要做的事了。像这种提高用户体验的提示可以使应用程序更加的直观。这也是核心动画最有用的特征，因此让我们看看怎么做。

利用核心动画实现图标的抖动

我们用 cocoa，在 OS x 上实现这个抖动的动画，然而在 iphone 是一样的。我们创建这个动画，播放时间长一点，以便于我们可以看到每个动画参数是怎么影响动画实际运行的效果的。然后我们再按照你实际在 iphone 和 ipod 上看到的效果，设定运行时间。

你可能想要打开例子程序，看下效果。我们提供了 OS X 和 iphone 平台上的例子工程。他们叫做 Icon Dance for OS X 和 Icon Dance for Iphone。

Icon Dance 应用程序的某些特征可能不同于你在 iphone 主屏幕上看到的情况。因为 Cocoa 的触摸功能在桌面应用程序中目前不可用，你不能触摸屏幕来开始或者停止图标的动画，或者用你的手指拖动图标。然而，这些方法和关联的代码演示了我们怎么实现这个图标的抖动，并且工作在不管任一平台上，就像你马上看到的一样。

图 4-5 和 4-6 展示了当在 OS X 下运行时，我们的应用程序看起来会是怎样。图 4-5 展示了当我们增加关闭盒子之前时，它又看起来怎么样，4-6 展示了在我们增加关闭盒子之后，看起来怎么样。


图 4-5 图标跳动示例程序


图 4-6 带关闭盒子的图标跳动

使用以下的步骤实现一个关键帧动画（看先前的章节，从基础动画到关键帧动画），让我们用图表列出来实现的每一步。如表格 4-1

| 实现步骤 | 有用的值 |
|---|---|
| 1. 指定动画的属性 | transform.rotation.x |
| 2. 指定每一个帧使用的值 | -2,2,-2 |
| 3. 指定动画的总执行时间 | 2 秒 |
| 4. 指定时间段 | 没使用，我们运行核心动画自动的分割每帧动画的时间 |
| 5. 指定时间步调 | 所有的帧都使用 kCAMediaTimingFunctionLinear |

表 4-1 图标震动动画的实现步骤

我们将仅仅指定三个关键帧的值来使例子足够的简单。（另外，这也是我们需要的全部动画。）我简单的设定了一个最大的重复数，它可以不停的循环。

看下清单 4-7 中，动画创建的代码。3 个关键帧被指定用 NSNumber 对象，这个对象是通过调用一个有用的函数 DegreesToNumber。你可以在清单 4-8 中看到这个函数的声明。DegreesToNumber 使用一个 CGFloat 值作为参数，然后返回一个 NSNumber 把度数转化成弧度，通过调用函数 DegreesToRadians。

```
- (CAAnimation*)shakeAnimation; {
CAKeyframeAnimation * animation; animation = [CAKeyframeAnimation
animationWithKeyPath:@"transform.rotation.z"]; [animation setDuration:0.5];
[animation setRepeatCount:10000];
// Try to get the animation to begin to start with a small offset // that makes it shake out of sync with other layers. srand([[NSDate
date] timeIntervalSince1970]);
float rand = (float)random();
[animation setBeginTime: CACurrentMediaTime() + rand * .0000000001];
NSMutableArray *values = [NSMutableArray array]; // Turn right
[values addObject:DegreesToNumber(-2)]; // Turn left
[values addObject:DegreesToNumber(2)]; // Turn right
[values addObject:DegreesToNumber(-2)]; // Set the values for the animation
[animation setValues:values]; return animation;
}
```

表 4-7

```
NSNumber* DegreesToNumber(CGFloat degrees) {
return [NSNumber numberWithFloat: DegreesToRadians(degrees)];
}
CGFloat DegreesToRadians(CGFloat degrees) {
return degrees * M_PI / 180; }
```

表 4-8

在表 4-7 中第一帧转 2 个度数到右边，第二帧转 2 个度数到左边，然后第三帧再转两个度数到右边。当动画完成时，它再次启动然后通过调用 setRepeatCount:10000 来重复 10000 次。视觉上就是在中轴上不停的抖动。要停止动画，可以通过调用-removeAnimationForKey:@"rotate"这个函数从层上面移除动画。

**旋转轴和层的几何特性**

在这个例子中，层是以中心点为轴旋转的。这个是层默认的锚点(0.5,0.5)。如果你想以不同的轴旋转，例如左下角，你需要给层指定一个新的锚点，通过调用-setAnchorPoint:CGPointMake(0.0,0.0)。对于这个例子，然而默认的就是基础的。层的几何特性在第六章“层的筛选器”中会详细讲解。

**完美的效果**

通过一些微小的改变 duration 参数和每个帧的值，我们可以得到正确的效果。如果你用清单 4-7 中的代码运行，你会注意到抖动的效果看起来有点慢。为了更快，简单的改变 duration 到更合适的值。我们目前是告诉动画在一秒钟的一半 0.5 完成。如果你改变成一秒钟的 1/4（改变 setDuration:0.5 到 setDuration:0.15），这个动画就是你渴望的效果了。

**增加关闭的盒子**

工程的这个部分和关键帧动画没有关系，但是它将展示给你怎么完成这个效果，然后介绍给你如何在层中绘画。

当动画正在运行时，你会注意到圆形框中带着一个 X 在里面。这个表明一个应用程序可以被删除在 iphone/iPod touch 上。在图标跳动应用程序中，关闭盒子什么也不做，仅仅用来展示效果的。

例如我们将按照下面做：

1. 设置关闭盒子的边框为 3 像素
2. 设置整个框架为 30x30 像素
3. 设置圆角率为 15，使它看起来想圆形
4. 设置背景颜色为黑色
5. 设定阴影颜色为黑色
6. 设定阴影的圆角率为 5 像素
7. 设定阴影的透明度为 1.0

在清单 4-9 中的代码实现了每一步。注意到我们设定了层的代理为 self。这使我们可以调用-(void)drawLayer:(CALayer*)theLayer inContext:(CGContextRef) theContext(看清单 4-10 的开始)，这让我们可以增加代码来给关闭盒子的层上绘制 X。这个绘制 X 的代码（清单 4-10）使用了 Core Graphics 路径和设定了点然后画第一个线，然后设定第二个点画第二条线。

```
-(CALayer*)closeBoxLayer; {
CGColorRef white = CGColorCreateGenericRGB(1.0, 1.0, 1.0, 1.0); CGColorRef black = CGColorCreateGenericRGB(0.0, 0.0, 0.0, 1.0);
CALayer *layer = [CALayer layer]; [layer setFrame:CGRectMake(0.0,
kCompositeIconHeight - 30.0, 30.0, 30.0)];
[layer setBackgroundColor:black]; [layer setShadowColor:black]; [layer setShadowOpacity:1.0]; [layer setShadowRadius:5.0];
[layer setBorderColor:white]; [layer setBorderWidth:3];
[layer setCornerRadius:15]; [layer setDelegate:self]; // Release the color refs
CFRelease(white); CFRelease(black);
return layer; }
```

LISTING 4-9 关键帧动画

```
- (void)drawLayer:(CALayer *)layer inContext:(CGContextRef)context
{

      // Make sure the call is applied to the close // box layer
if( layer == closeLayer )
{
// Create the path ref
CGMutablePathRef path = CGPathCreateMutable();
// Set the first point and add a line
CGPathMoveToPoint(path,NULL,10.0f,10.f); CGPathAddLineToPoint(path, NULL, 20.0, 20.0);
// Set the second point and add a line
CGPathMoveToPoint(path,NULL,10.0f,20.f); CGPathAddLineToPoint(path, NULL, 20.0, 10.0);
// Set the stroke color to white
CGColorRef white =
CGColorCreateGenericRGB(1.0, 1.0, 1.0, 1.0);
CGContextSetStrokeColorWithColor(context, white); CGColorRelease(white);
// Start drawing the path
CGContextBeginPath(context); CGContextAddPath(context, path);
// Set the line width to 3 pixels
CGContextSetLineWidth(context, 3.0); // Draw the path
CGContextStrokePath(context);
// Release the path
CGPathRelease(path); }
}
```

清单 4-10 在关闭盒子中画 X

**开始和停止动画**

核心动画没有提供直接的开始和停止方法。然而，当你增加动画到层上时，动画就开始，当你从层上移除动画时，动画就结束。为了判断一个动画是否正在运行，需要查询动画字典来决定是否动画仍然存在。如果在，动画是仍然在运行。如果不在，动画就停止了。

清单 4-11 演示了如何来开始和停止动画的。当动画开始时，首先增加关闭盒子层然后使用关键字 rotate 增加动画。这个关键字是重要的，因为它是被用来引用这个动画的和当你想停止它和观看它是否仍然在运行。

```
- (void)toggleShake; {
if( [self isRunning] ) {
[self stopShake]; }
else
{
[self startShake];
} }
(BOOL)isRunning; {
return ([self animationForKey:@”rotate”] != nil);
}
- (void)startShake; {
[self addSublayer:closeLayer];
// Tell the closeLayer to draw its contents which is
// an ‘X’ to indicate a close box.
[closeLayer setNeedsDisplay];
[self addAnimation:[self shakeAnimation] forKey:@”rotate”];
}
- (void)stopShake; {
[closeLayer removeFromSuperlayer];
[self removeAnimationForKey:@”rotate”]; }
```

清单 4-11

当-stopShake 是被调用，关闭盒子层从父层移除时，我们也会从层中移除动画。来判断是否动画仍在运行，-isRunning 可以检查是否动画是在动画字典中。如果在，动画在运行，否则，不在运行。

### 4.1.6. 总结

基础动画是被作为单一的关键帧动画。这种定义帮助你解释关键帧动画提供了什么。在动画中的关键帧就像 CABasicAnimation 涉及到的字段 toValue：目的值的一个列表。提供给关键帧动画的列表值要么使用 values 要么使用 path 这个字段。这个值的每一个都是目的地，在动画期间到达每个点。

关键帧动画提供了一个强大的方法，使用一个简单的列表来做动画，通过自动的插入值到动画中，这使开发者更加容易了。这样就使创建动画的用户体验更加直接方便去实现了。