



Core Animation:

Simplified Animation Techniques for
Mac and iPhone Development

第二部分

核心动画基础

第三章

基础动画

版本 1.0

翻译时间：2012-10-22

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](http://www.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術需要支持的话，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您。

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
第 4 章 基础动画	6
4.1.1. 最简单的动画	6
4.1.2. 动画的代理对象	6
4.1.3. 在窗口、视图和层动画之间的不同	6
4.1.4. 准备一个视图来做层动画	8
4.1.5. 使用 CABasicAnimation	10
4.1.6. 有用的动画属性	12
4.1.7. 动画组	12
4.1.8. 总结	14

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

DevDiv 翻译

第 4 章 基础动画

核心动画是一个强大成熟的技术，运用它可以使你达到事半功倍的效果。为了执行动画，苹果公司提供了动画的代理对象，当它被调用时同时那些可视的组件，譬如视图的外框、透明度、或者位置改变时，就会自动的触发隐式的动画。对于基础的层动画，`CABasicAnimation` 类通过提供一个开始值和一个结束值，来执行一个动画。这章，通过应用程序，我们来看一些执行动画的基础方法。

4.1.1. 最简单的动画

由于核心动画集成到了 `cocoa` 中，因而你通过给窗口、视图、层简单的设定一些你感兴趣的属性值，从而使那些可视的组件做动画到新的值。当你使用 `CALayer` 的时候，你需要做的就是直接设置这些值。例如，如果你想改变一个层的边框大小，你简单的调用 `[layer setBounds:newFrame]`，这里要说的是，层是你已经创建的 `CALayer` 对象，并且要增加这个对象到层树中，而 `newFrame` 是一个 `CGRect` 包含了边框的尺寸和原点。当这些代码运行时，就会使用关键路径“`bounds`”的默认动画，使层具有边框大小改变时的动画。

简单的说，当你使用一个窗口(`NSWindow`)或者视图(`NSView`)，你需要做的就是，使用了动画的代理对象，来设置窗口或者视图的属性值。这意味着，例如，你可以代替使用 `[view setFrame:newFrame]` 来设置窗口的框架，通过 `[[view animator] setFrame:newFrame]` 这个函数调用，达到目的。不同之处就是，你用了个动画的代理对象，设置了这些属性，而这些都会隐式的执行框架的初始值到结束值的动画。

4.1.2. 动画的代理对象

因此什么是动画的代理对象呢？动画的代理在 `NSView` 和 `NSWindow` 上面都可用。它实现了 `NSAnimatablePropertyContainer` 的代理。这包含了使用键值编码对，当你需要做插值和在场景后面做动画时，来设定你指派的参数值。

顾名思义，动画代理就是一个媒介，来设定你给定的值，达到控制动画的属性，从开始或者目前的值，到结束的值。它设定了这些值，就像是你已经在这些属性上调用了一样。

4.1.3. 在窗口、视图和层动画之间的不同

在窗口，视图和层之间的动画是一样的，然而实现却不同。这个段落中，我们将讨论你可能会想到的，最普通的动画的实现---框架的尺寸改变。

窗口尺寸改变

自从 `mac os x` 的第一个版本，动画一个窗口框架的能力已经可用，对于开发者来说，利用

```
-(void)setFrame:(NSRect>windowFrame  
display:(BOOL)displayViews animate:(BOOL)performAnimation
```

来设定动画。第一个参数是你将要动画到的新框架。第二个参数告诉窗口在所有的子视图上面调用 `displayIfNeeded`，第三个参数是告诉窗口以动画的方式，从目前的框架转换到新的框架。如果最后一个参数设定为 `NO`，新的框架会立即改变，而不会有渐变的动画。

既然内嵌的窗口框架具有改变大小的能力，那么为什么你需要使用核心动画来改变窗口的框架那？答案是你不必。对于许多情况，当你改变尺寸的时候，你可能仅仅需要使用内嵌的功能就行了。然而，可能有这些情况，你想要在窗口动画上有更多的控制。当你要做这些的时候，记住一些事情。`NSWindow` 就像 `NSView` 一样，有一个动画代理。当你调用动画者时，它会按照你指定的参数做动画，但是那些参数都是附带物。如果你想把一个窗口，在屏幕上移动到一个不同的位置，事实上，要么你用

```
- (void)setFrame:(NSRect>windowFrame  
display:(BOOL)displayViews
```

这个方法（这个方法不要第三个参数），或者你可以增加动画到窗口本身的动画字典中。首先让我们看怎么来使用动画代理。如下：

```
[[window animator] setFrame:newFrame display:YES];
```

这个可以使你简单的动画窗口的框架。

默认的动画播放是 0.25 秒。如果你想改变这个时间，需要使用 `NSAnimationContext` 这个对象，对于 `CATransaction` 它是 `NSView/NSWindow` 的副本。如果我们利用 `NSAnimationContext` 封装调用 `setFrame`，我们可以指定动画的时间。表 3-1 演示了如何做：

```
[NSAnimationContext beginGrouping];  
[[NSAnimationContext currentContext] setDuration:5.0f];  
[[window animator] setFrame:newFrame display:YES];  
[NSAnimationContext endGrouping];
```

List 3-1

这样使框架的改变时间为 5 秒，而非默认的 0.25 秒。你下一段即将看到，这种方法也是你能使用的，可以改变 `NSView` 的动画时间。

使用核心动画的 `CABasicAnimation` 也能够窗口和视图上做动画，但是动画如何安装有点不同。作为通过窗口动画代理调用 `setFrame` 的替换者，我们可以创建一个 `CABasicAnimation` 和动画框架的属性。下面在表 3-2 中可以看到如何在窗口创建、增加和运行基础动画。

```
CABasicAnimation *animation =  
[CABasicAnimation animationWithKeyPath:@" frame" ];  
[animation setFromValue:[NSValue valueWithRect:oldFrame]];  
[animation setToValue:[NSValue valueWithRect:newFrame]];  
[animation setDuration:5.0f];  
[window setAnimations:[NSDictionary animation forKey:@" frame" ]];  
[[window animator] setFrame:newFrame display:YES];
```

表 3-2 增加一个动画到窗口动画字典

这个动画的效果和表 3-1 是相同的。

视图的改变

视图和窗口同样能被改变，但是你要使用不同的关键路径。你可以在视图上使用 `setFrame`，也可以在窗口上使用同样的代码。如图 3-3

```
[NSAnimationContext beginGrouping];  
[[NSAnimationContext currentContext] setDuration:5.0f];  
[[view animator] setFrame:newFrame display:YES];  
[NSAnimationContext endGrouping];
```

表 3-3

仅仅的不同之处是，表 3-1 中调用的对象不同，这里是用的 `view`

如果你想使用显示的动画，代替框架做动画，你可以使框架原点和框架尺寸做动画。表 3-4 展示了如何使用这些属性做动画。

```
CABasicAnimation *originAnimation = [CABasicAnimation  
animationWithKeyPath:@" frameOrigin" ];  
[originAnimation setFromValue:[NSValue  
valueWithPoint:oldImageFrame.origin]];  
[originAnimation setToValue:[NSValue valueWithPoint:newFrame.origin]];  
[originAnimation setDuration:5.0];  
CABasicAnimation *sizeAnimation = [CABasicAnimation  
animationWithKeyPath:@" frameSize" ];  
[sizeAnimation setFromValue:  
[NSValue valueWithSize:oldImageFrame.size]];
```

```
[sizeAnimation setValue:[NSValue valueWithSize:newFrame.size]];
[sizeAnimation setDuration:5.0];
[[view animator] setAnimations:[NSDictionary
dictionaryWithObjectsAndKeys:originAnimation,
@" frameOrigin" ,
sizeAnimation,
@" frameSize" ,
nil]];
[[view animator] setFrame:newFrame];
```

表 3-4 显示的使原点和尺寸做动画

层的尺寸改变

一个层的框架的动画是有点不同于窗口和视图的动画。在 CALayer 对象中没有动画代理可用，但是也并非当你想有一个显示的改变动画的属性时，动画总是被调用。事实上，如果你想禁用动画，你就得显示的关闭动画。表 3-5 演示了如何做。

```
[CATransaction begin]
[CATransaction setValue:[NSNumber numberWithInt:YES]
 forKey: kCATransactionDisableActions]
[layer setBounds:bounds];
[CATransaction commit];
```

表 3-5 显示的关闭动画

CATransaction 类和我们在表 3-2 和 3-4 对于窗口和视图中的 APPKit 中使用的 NSAnimationContext 非常的相似。就像 NSAnimationContext 一样，CATransaction 也可以给动画设置时间。表 3-6 演示了如何做：

```
[CATransaction begin]
[CATransaction setValue:[NSNumber numberWithFloat:5.0f]
 forKey: kCATransactionAnimationDuration]
[layer setBounds:bounds];
[CATransaction commit];
```

表 3-6

就像你猜想的，我们也能显示的使用层的属性做动画。表 3-7 是来完成表 3-6 同样的效果的代码。

```
CABasicAnimation *boundsAnimation = [CABasicAnimation
animationWithKeyPath:@" bounds" ];
[boundsAnimation setFromValue:[NSValue valueWithRect:oldRect]];
[boundsAnimation setValue:[NSValue valueWithRect:newRect]];
[boundsAnimation setDuration:5.0f];
[layer setBounds:NSRectToCGRect(newRect)];
[layer addAnimation:boundsAnimation forKey:@" bounds" ];
```

表 3-7

在表 3-6 中，我们也使用了 CABasicAnimation 这个类，它是基础动画的主要类。马上我们会深入的理解这个类，但是首先我们要安装一个简单的 Xcode 的工程来演示基础的层动画。

4.1.4. 准备一个视图来做层动画

当你创建一个基于核心动画的工程时，你要做的第一件事是确保视图的根层有一个后背的层。下面让我们一步一步创建基于核心动画的工程，并且在 OS X 上安装一个根层。

创建 XCode 工程

创建应用程序，按照下面的步骤：

- 1.在 Xcode 中，按 shift - command - N 和在工程模板中，选择 cocoa 应用程序。
- 2.命名工程叫 CA Basics，点击保存。
- 3.扩展 frameworks 组，control - click 链接 framework 子组，并且选择 add>existing frameworks
- 4.在结果对话框中，导航到/system/library/frameworks，然后选择 QuartzCore.framework。按照提示，点击增加 2 次。
- 5.Control - click class 组，选择 add>New File。

- 6.在新的模板对话框中，下面的 cocoa 组中选择 Objective-c 类，然后点击 next。
- 7.命名文件为 appDelegate.m，并且核对确保创建了 appDelegate.h。然后点击完成
- 8.选择 appDelegate.h 在编辑器中打开文件，并且增加下面的代码

```
@interface AppDelegate : NSObject {
IBOutlet UIWindow *window;
}
```

- 9.选择 appDelegate.m 在编辑器中打开文件，增加下面的代码

```
@implementation AppDelegate
- (void)awakeFromNib;
{
[[window contentView] setWantsLayer:YES];
}
@end
```

- 10.在工程的 Resources 组下面，双击 MainMenu.xib，在接口编辑器中打开 XIB 文件。
- 11.从 Library 画板中，拖出 NSObject 对象到 MainMenu.xib 中，重命名它为 appDelegate。
- 12.确保 AppDelegate 对象被选择，在对象编辑器中，点击 Identity 标签并且改变类的域为 AppDelegate
- 13.在 MainMenu.xib 中，control - click File's Owner 并且拖拽链接到 AppDelegate 对象中。在下拉的按钮中选择 delegate。
- 14.在 MainMenu.xib 中，control - click AppDelegate 和拖拽链接到 Window object。在下拉菜单中选择 window
- 15.保存 xib 文件和返回 XCode

工程时安装完毕。在先前的步骤中，我们创建了一个应用程序的代理，这个代理是用来控制我们的层、窗口、和视图的。

给根层增加动画层

增加我们将要动画的层，做法如下：

- 1.打开 AppDelegate.h 和增加一个 CALayer 实例变量：

```
@interface AppDelegate : NSObject
{
IBOutlet UIWindow *window;
CALayer *layer;
}
```

- 2.打开 AppDelegate.m，并且增加层的初始化代码在-awakeFromNib:

```
@implementation AppDelegate
- (void)awakeFromNib;
{
[[window contentView] setWantsLayer:YES];
layer = [CALayer layer];
[layer setBounds:CGRectMake(0.0, 0.0, 100.0, 100.0)];
// Center the animation layer
[layer setPosition:CGPointMake([[window contentView]
frame].size.width/2,
[[window contentView]
frame].size.height/2)];
CGColorRef color = CGColorCreateGenericRGB(0.4, 0.3, 0.2, 1);
[layer setBackgroundColor:color];
CFRelease(color);
[layer setOpacity:0.75];
[layer setBorderWidth:5.0f];
[[[window contentView] layer] addSublayer:layer];
}
@end
```

层内存分配的注意事项

尽管你有了一个实例变量，但是当你安装完层后，另一个你应该意识到的是，这个层它没有 `retained`，除非你显示的 `retain` 了它。在 `objective - c` 的内存管理中的规则，是在你仅仅需要的地方 `retain`。如果你不想长久的持有他，你不应该 `retain` 一个对象，而你需要持有时，你就应该 `retain` 这个对象。

这听起来简单，但是实际用起来比较复杂。在先前的代码中，你看到了我们为层开辟空间，用了一个便捷的方法 `layer = [CALayer layer]`；这会为 `CALayer` 对象分配一个 `auto-released` 的对象。当这个层对象离开了 `-awakeFromNid` 这个代码段空间时，它将自动释放，除非你 `retain` 了它。在我的例子中，我们增加了层到 `contentView` 的子层数组中，这样就给我们 `retain` 了层。然而，假如我们要等到真在 `-awakeFromNib` 中初始化时，增加的层到子层的数组中，我们需要用 `[[CALayer alloc] init]` 这个方法来自初始化。然而我们需要释放这个层，在 `dealloc` 方法中调用 `[layer release]`；

你只要一次使用了 `-removeFromSuperlayer` 这个方法，你会发现，如果你再试图增加层到子层中时，它将会使你的应用程序 `crash`。这是因为当调用 `-removeFromSuperLayer` 时，层将会被释放。假如你想从它的父层中，移除它的子层，但是还想保留它在内存中，那么你就必须 `retain` 这个层。

4.1.5. 使用 CABasicAnimation

这里你已经看到了 `CABasicAnimation` 对象的行动。然而，在这个段落里，我们会考虑一些细节，如何来使用这些类和基础动画。

使用 `CABasicAnimation` 类实现的基础动画，是通过 2 个值，一个开始的，一个结束的来做动画的。例如，为了在窗口中，移动从一个点到另一个点，我们能够使用关键路径 `position` 创建一个基础动画。我们可以给动画一个开始的值和一个结束的值，然后增加动画到层中。在下一个运行循环中，那个动画就可以立即执行。表 3-8 演示了如何来对一个层的位置做动画。

```
- (IBAction)animate:(id)sender;
{
    CABasicAnimation *animation =
    [CABasicAnimation animationWithKeyPath:@"position"];
    [animation setFromValue:[NSValue valueWithPoint:startPoint]];
    [animation setToValue:[NSValue valueWithPoint:endPoint]];
    [animation setDuration:5.0];
    [layer addAnimation:animation forKey:@"position"];
}
```

这个代码移动一个层的位置，从 `startPoint` 到 `endPoint`。这两个值都是 `NSPoint` 对象。`Position` 属性是一个层的中心点。它和它包含的层有关系。

如果你增加了下表中的代码到我们先前创建的工程中，你就可以在接口编辑器中，简单的链接一个按钮的行动。跟着下面的步骤做：

■ 打开 `AppDelegate.h`，然后增加响应声明如下：

```
@interface AppDelegate : NSObject
{
    IBOutlet NSWindow *window;
    CALayer *layer;
}
- (IBAction)animate:(id)sender;
```

■ 打开 `AppDelegate.m` 和增加 `animate` 的实现代码如表 3-8

■ 打开接口编辑器。从对象库中，拖出按钮到 `main window` 上。

■ `Control - click` 拖拽在 `main window` 上的按钮，然后链接到 `AppDelegate` 对象上。选择 `animate` 这个行动

■ 返回 `Xcode`，编译运行，看效果。

就这些了。这就是给层创建动画的全部了。你创建一个动画，设置开始值和结束值，设置动画时间（如果不设置的话，默认是 0.25 秒）。然后就增加这个动画到你想要做动画的层上。

还要继续说，因为实现的细节增加了一个微小的不同和复杂性，这里你最好不要立刻停下来。例如，当你用表 3-8 的代码，第一次运行动画时，你会注意到随着你指定的动画时间，层移动到了正确的位置，这时

动画完成，它就会跳回到原来的位置。这是个 bug 么？我们怎么解决那？下面进一步分析。

动画 vs 层的属性

当你创建一个 `CABasicAnimation` 时，你需要通过 `-setFromValue` 和 `-setToValue` 来指定一个开始值和结束值。当你增加基础动画到层中的时候，它开始运行。当用属性做动画完成时，例如用位置属性做动画，层就会立刻返回到它的初始位置。

记住当你做动画时，你至少使用了 2 个对象。这些对象都是层本身，一个层或者层继承的对象，和在先前的例子中你分配给层的 `CABasicAnimation` 对象。因为你给动画对象设定了最后的值（目的地），但是并不意味着当动画完成的时候，层的属性就改变成了最后的值。当动画完成时，你必须显示的设定层的属性，这样动画结束后，你的层才能真正的到你设定的属性值上。

你可以简单的停止动画到你结束的点上，但是这仅仅是一个视觉效果。层实际的值仍然是一样的。要真的改变内部的值，就像刚才所说的你必须显示的设定那个属性。例如，显示的设定位置的属性，你需要在层中调用 `-setPosition` 方法。但是，这会造成一点问题。

如果你通过 `-set` 这个方法显示的设定了层属性的值，那么默认的动画将被执行，而非之前你设定的动画。在表 3-9 中演示了你设置位置的方法。注意到了，我们使用 `position` 已经创建了基础动画，但是我们在层上显示的调用了 `-setPosition` 方法，就覆盖了我们设定的动画，使我们设定的基础动画完全没用了。如果你使用了这个代码，你会看到虽然我们的层结束的时候放到了正确的位置，但是它使用的是默认的 0.25 秒，而非我们在动画里显示设定的 5 秒钟。

```
CABasicAnimation *animation =
[CABasicAnimation animationWithKeyPath:@"position"];
[animation setFromValue:[NSValue valueWithPoint:startPoint]];
[animation setToValue:[NSValue valueWithPoint:endPoint]];
[animation setDuration:5.0];
[layer setPosition:endPoint];
[layer addAnimation:animation forKey:nil];
```

表 3-9 动画和升级位置属性

因此现在问题出来了，你怎么能使用我们设定的动画呢？看表 3-9 的最后一行，注意到 `forKey:` 这个参数是被设定为 `nil`。这就是为什么动画不能覆盖默认动画的原因。如果你改变最后一行为 `[layer addAnimation:animation forKey:@"position"]`，动画将会按照我们设定的时间工作。这告诉了层当需要做动画时，使用我们给关键路径指定的新动画。

隐式的层动画和默认的时间步调

像我们之前章节做的，我们可以使用 `CATransaction` 类来重载默认的时间，并且它可以方便的使用我们指定的时间做动画。如果你使用表 3-10 的代码，`position` 属性是被设置在层理，那个属性的动画就按照你期望的方式运行。

```
[CATransaction begin];
[CATransaction setValue:[NSNumber numberWithInt:5.0]
forKey:kCATransactionAnimationDuration];
[layer setPosition:endPoint];
[CATransaction commit];
```

表 3-10 隐式的重载了动画的时间

然而，当你运行这个代码时，你会看到动画的时间确实是 5 秒，但是它应用了默认的时间步调，就是 `KCAMediaTimingFunctionEaseInEaseOut`。这个会引起开始的时候变慢，然后加速，之后再慢下了从而到达目的地。如果这是你想要的时间步调再好不过了，但是如果你想要一个线性的步调（`KCAMediaTimingFunctionLinear`），例如，你需要考虑其他的方法。没有直接的方法给隐式的动画设定时间步调。

这意味着，如果你想用其他的时间步调，你不得不用显示的动画，就像表 3-9 演示的那样。

视觉粘性

我们可以使用另一个方法，通过设定我们动画对象的属性，来达到动画完成时，固定在完成位置的效果。换句话说，层会展现为目的值。这个方案仅仅是视觉效果，也就是说那个层根本的位置还是动画开始时的位置。当你不需要真正的改变层的值时，这是一个很好的方法。表 3-11 展示了如何去实现这个方法，使层固定在结

束的位置。

```
CABasicAnimation *animation = [CABasicAnimation
animationWithKeyPath:@" position" ];
[animation setValue:[NSValue valueWithPoint:endPoint]];
[animation setDuration:5.0];
[animation setFillMode:kCAFillModeForwards];
[animation setRemovedOnCompletion:NO];
[layer addAnimation:animation forKey:@" position" ];
```

表 3-11 使层的位置固定

我们需要设定 2 个动画属性。首先是填充模式。我们告诉动画来固定属性值为最后的值，通过调用 `setFillMode` 方法，给它传递一个 `kCAFillModeForwards` 属性来实现。然后我们必须告诉动画，当动画结束的时候，不要动画从层的动画数组中移除。用 `setRemoveOnCompletion` 方法，传递 `NO` 来实现。

4.1.6. 有用的动画属性

你已经发现了，所有可以在层上做动画的属性。然而，在动画对象（`CABasicAnimation`）中还有许多有用的属性，这些属性可以让你便于控制动画，提升动画的性能。

Autoreverses

当你设定这个属性为 `YES` 时，在它到达目的地之后，动画的属性会返回到开始的值，代替了直接跳转到开始的值。

Duration

`Duration` 这个参数你已经相当熟悉了。它设定开始值到结束值花费的时间。期间会被速度的属性所影响。

RemovedOnCompletion

这个属性默认为 `YES`，那意味着，在指定的时间段完成后，动画就自动的从层上移除了。这个一般不用。假如你想要再次用这个动画时，你需要设定这个属性为 `NO`。这样的话，下次你在通过 `set` 方法设定动画的属性时，它将再次使用你的动画，而非默认的动画。

Speed

默认的为 `1.0`。这意味着动画播放按照默认的速度。如果你改变这个值为 `2.0`，动画会用 2 倍的速度播放。这样的影响就是使持续时间减半。如果你指定的持续时间为 6 秒，速度为 `2.0`，动画就会播放 3 秒钟---一半的持续时间。

BeginTime

这个属性在组动画中很有用。它根据父动画组的持续时间，指定了开始播放动画的时间。默认的是 `0.0`。组动画在下个段落中讨论“`Animation Grouping`”。

TimeOffset

如果一个时间偏移量是被设定，动画不会真正的可见，直到根据父动画组中的执行时间得到的时间都流逝了。

RepeatCount

默认的是 `0`，意味着动画只会播放一次。如果指定一个无限大的重复次数，使用 `1e100f`。这个不应该和 `repeatDuration` 属性一块使用。

RepeatDuration

这个属性指定了动画应该被重复多久。动画会一直重复，直到设定的时间流逝完。它不应该和 `repeatCount` 一起使用。

4.1.7. 动画组

在先前的段落中，“有用的动画属性”，我们定义了 2 个特殊的属性，是关系到动画组的：`beginTime` 和 `timeOffset`。在我们讨论这个之前，然而，让我们考虑为什么你想要使用动画组，而非给层增加一列动画。

在表 3-12 中，你可以看到我们建立了一列基础动画，和简单的增加他们到层上面。如果你想要所有的动画开始在同样的时间，并且他们中每个动画都有同样的执行时间，这个方法是足够了。

```
- (IBAction)animate:(id)sender;
{
NSRect oldRect = NSMakeRect(0.0, 0.0, 100.0, 100.0);
NSRect newRect = NSMakeRect(0.0, 0.0, 300.0, 300.0);
CABasicAnimation *boundsAnimation =
```

```
[CABasicAnimation animationWithKeyPath:@" bounds" ];
[boundsAnimation setFromValue:[NSValue valueWithRect:oldRect]];
[boundsAnimation setToValue:[NSValue valueWithRect:newRect]];
[boundsAnimation setDuration:5.0f];
CABasicAnimation *positionAnimation =
[CABasicAnimation animationWithKeyPath:@" position" ];
[positionAnimation setFromValue:
[NSValue valueWithPoint:
NSPointFromCGPoint([layer position])]];
[positionAnimation setToValue:
[NSValue valueWithPoint:NSMakePoint(0.0, 0.0)]];
[positionAnimation setDuration:5.0f];
CABasicAnimation *borderWidthAnimation =
[CABasicAnimation animationWithKeyPath:@" borderWidth" ];
[borderWidthAnimation setFromValue:[NSNumber numberWithFloat:5.0f]];
[borderWidthAnimation setToValue:[NSNumber numberWithFloat:30.0f]];
[borderWidthAnimation setDuration:5.0f];
[layer addAnimation:boundsAnimation forKey:@" bounds" ];
[layer addAnimation:positionAnimation forKey:@" position" ];
[layer addAnimation:borderWidthAnimation forKey:@" borderWidth" ];
}
```

每个动画都有 5 秒的执行时间，并且它们在下个循环里一起播放，最后同时结束。层的位置到左下角，层的边框宽度增加 30 个像素，和层的尺寸增加从 100x100 像素到了 300x300 像素。

让我们说下我们要做的情况，并不是使所有的动画同时播放，我们想要它们按顺序播放之前定义好的顺序。我们可以完成这些，通过使用动画组合设定 `beginTime` 这个属性的区域。这里我提下，这种情况下如果我们使用关键帧可能更有意义，但是你需要读到第四章“**Keyframe Animation**”了解他怎么工作。

我们必须显示的指定我们组动画的执行时间，以便于能为每个动画分离一部分时间。例如，我们设定我们的动画时间为 15 秒钟，然后给每个动画 5 秒钟的播放时间。列表 3-13 展示了先前的例子，这里用动画组替代，来更好的控制每个动画的播放。

```
- (IBAction)animate:(id)sender;
{
NSRect oldRect = NSMakeRect(0.0, 0.0, 100.0, 100.0);
NSRect newRect = NSMakeRect(0.0, 0.0, 300.0, 300.0);
CABasicAnimation *boundsAnimation =
[CABasicAnimation animationWithKeyPath:@" bounds" ];
[boundsAnimation setFromValue:[NSValue valueWithRect:oldRect]];
[boundsAnimation setToValue:[NSValue valueWithRect:newRect]];
[boundsAnimation setDuration:15.0f];
[boundsAnimation setBeginTime:0.0f];
CABasicAnimation *positionAnimation =
[CABasicAnimation animationWithKeyPath:@" position" ];
[positionAnimation setFromValue:
[NSValue valueWithPoint:
NSPointFromCGPoint([layer position])]];
[positionAnimation setToValue:
[NSValue valueWithPoint:NSMakePoint(0.0, 0.0)]];
[positionAnimation setDuration:15.0f];
[positionAnimation setBeginTime:5.0f];
CABasicAnimation *borderWidthAnimation =
[CABasicAnimation animationWithKeyPath:@" borderWidth" ];
[borderWidthAnimation setFromValue:[NSNumber numberWithFloat:5.0f]];
[borderWidthAnimation setToValue:[NSNumber numberWithFloat:30.0f]];
[borderWidthAnimation setDuration:15.0f];
[borderWidthAnimation setBeginTime:10.0f];
CAAnimationGroup *group = [CAAnimationGroup animation];
[group setDuration:15];
```



```
[group setAnimations:
[NSArray arrayWithObjects:boundsAnimation,
positionAnimation,
borderWidthAnimation, nil]];
[layer addAnimation:group forKey:nil];
}
```

表 3-13 使用动画组

注意到我们为每个分割的动画都设定了 15 秒的执行时间，但是每个动画的开始时间分别为 0.0, 5.0, 和 10.0。

你也注意到了我们仅仅增加组动画给层。组动画对象通过调用 `-setAnimations` 这个方法增加。

你可以看到使用组给予了你多么灵活的方法。你仅仅需要按照你的需求关注于你的执行时间和开始时间。如果你想要动画同时发生，你仅仅需要改变开始时间，就是你想要播放的开始时间。你要保持同样的执行时间，否则在层中每个关键路径（keyPath）的值（就是，`bounds`，`position`，和 `borderWidth`），在预期结束时，都会突兀的返回到开始的值，看起来很古怪。保持所有的执行时间都一样，可以使他们等待直到动画结束时，才返回到原来的值。如果你不想他们返回，在动画结束时，你需要显示的设定上一段落中提到的属性值“使用 `CABasicAnimation`”

4.1.8. 总结

基础动画是非常强大的。为了完成动画的目标，你有很多便捷的方法。通常你都不需要用基础动画额外的方法。如果你需要的全部就是动画代理，使用它，保持简单！如果你需要的就是设置一个层的属性，就调用层属性的 `set` 方法让核心动画控制其余的工作。如果你需要更灵活的动画参数，使用 `CABasicAnimation` 对象，设定所有的动画属性。通常情况下，你仅仅需要基础动画就可以了。



点击这里访问: DevDiv.com 移动开发论坛