Core Animation:

Simplified Animation Techniques for Mac and iPhone Development

第三部分

核心动画的层

第十一章

用户交互

版本 1.0

翻译时间：2012-12-17

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

本书翻译贴各章汇总

第 11 章 用户交互

如果用户不能和图形界面进行交互，它存在的意义有何在那？然而，核心动画的 API 显示，没有直接的方法可以接收用户的交互。

这一章我们焦距于怎么给应用程序增加交互点，尤其是核心动画。下面我们就看鼠标和键盘的输入的交互。

11.1. 鼠标点击

在你的应用程序中，最普通的交互就是具有响应鼠标点击的事件，当用户点击界面上的一些元素时，可以执行默写功能，例如点击保存按钮。通常在 Cocoa 应用程序中，这类事件是通过 NSResponder 来控制的。然而，因为核心动画是被设计的尽量轻量级，所以 CALayer 就没有继承自 NSResponder，并且层也不能接收鼠标点击事件。然而，你需要通过 NSView 来传递这些事件。

当工作在层后面的视图上时，你的应用程序就能在一个 NSView 上捕获鼠标事件，并且处理他们。然而，我们对整个栈中仅仅有这一个的 NSView 不是非常感兴趣。因为 NSView 仅仅是一个接收事件的对象，它必须指出那些层是被点击了，然后该做什么样的动作。

11.2. 点击测试 CALayer 对象

当应用程序有仅仅一个 NSView 对象时，所有的用户交互都会在 NSView 上发生。它接收所有的鼠标和键盘输入，然后需要决定怎么去处理他们。在分离接收的事件之前，我们需要创建一个自定义的 NSView 来接收事件，并且给它分配一个代理对象，如清单 11-1.

```
#import <Cocoa/Cocoa.h>
@interface LZContentView : NSView {
IBOutlet id delegate; }
@end
```

清单 11-1 接收鼠标事件的 LZContentView 的头文件

继承于 NSView 的子类增加了一个成员变量 delegate。因为这个对象是被分配在 Interface builder(一个可视化的图形编辑工具)，它被标记为 IBOutlet。只要你想在 Interface Builder 中绑定一个对象，就不能定义为 id 类型，我们需要定义为 IBOutlet，以便于让 Interface Builder 知道它。

在 NSView 的子类中，我们仅仅想要捕获-mouseDown:和-mouseUp:事件，就像清单 11-2 所示。当被捕获到时，这些事件就被发送到 delegate 中，那里控制一些其他的交互。

```
#import “LZContentView.h” @implementation LZContentView
- (void)awakeFromNib {
}
- (void)mouseDown:(NSEvent*)theEvent {
[delegate mouseDown:theEvent]; }
- (void)mouseUp:(NSEvent*)theEvent {
[delegate mouseUp:theEvent]; }
@end
```

清单 11-2 LZContentView 实现接收鼠标事件的文件

11.3. 点击测试

当用户点击一个应用程序时，2 个 NSEvent 对象是被生产。一个事件是当鼠标点击下去

的时候是被生成，然后立刻鼠标是被释放。为了跟随这个列子，应用程序也会区别这 mouseDown 和 mouseUp 这个两个事件，从而做不同的交互。

当鼠标事件行动时，我们需要做的第一件事就是决定那个层是被点击了。因为在 NSView 的层级中，不知道有多少个层在里面，所以我们通过位置不能判断那个层是被点击了。幸运的是，CALayer 有-hitTest 方法，这个方法的设计就是用来解决这个问题。当 CGPoint 是被传递到根部的 CALayer 上时，它会返回那个点击点所落在的最深层的 CALayer 上。这就能使你快速决定那个 CALayer 是被点击，从而做出响应的反应。

11.3.1. 实例应用程序：颜色的改变

为了演示 hit test 如何工作，我们创建了一个简单的应用程序，有三个按钮：红，绿和蓝并且伴随着一个颜色的条来显示颜色的改变，如图 11-1.

这些按钮和颜色条都是用 CALayer 对象创建的。在这个应用程序的第一个版本，我们决定那个按钮是被按下去，并且响应。

11.3.2. LZBButtonLayer

创建这个应用程序的第一步是创建按钮。按钮是有 2 个 CALayer 对象组成的：

主要的层(LZButtonLayer 本身)，控制着边框和圆角率(如清单 11-3 所示)

CATextLayer 对象，用来展示文本的(如清单 11-4)

头文件如清单 11-3，获得 CATextLayer 子层的引用，可以让你根据需要调整文本的内容。同样我们也需要一个颜色对象的引用。头文件包含了一对 get 和 set 方法-string、-setString，这个是用来给 CATextLayer 设定字符串的。最后要说的是，-setSelected 方法通知层被点下去了。

```
#import <Cocoa/Cocoa.h>
@interface LZButtonLayer : CALayer {
__weak CATextLayer *textLayer;
CGColorRef myColor; }
@property (assign) CGColorRef myColor;
- (NSString*)string;
- (void)setString:(NSString*)string; - (void)setSelected:(BOOL)selected;
@end
```

清单 11-3

只要[CALayer layer]调用了，init 的方法就成为了默认的初始化方法，如清单 11-4 所示。按钮的层重载了默认的初始化方法，并且配置了自己的按钮。当[super init]完成它的初始化工作时，按钮的背景层通过设定 cornerRadius,bounds,borderWidth 和 borderColor 这些配置了。

下一步，textLayer 是被初始化。既然 textLayer 是个自动释放的对象(因为我们没有调用

alloc 或者 copy 的方法)，我们就需要引用它。因为我们定义为一个弱引用，我们不能获得它，相反让层的继承树来控制这个引用。通过 CATextLayer 的初始化，下一步就是来设定层的默认属性，并且给它们在背景层上的正确位置。

```
#import “LZButtonLayer.h” @implementation LZButtonLayer @synthesize myColor;
- (id)init {
if (![super init]) return nil;
[self setCornerRadius:10.0];
[self setBounds:CGRectMake(0, 0, 100, 24)]; [self setBorderWidth:1.0];
[self setBorderColor:kWhiteColor];
textLayer = [CATextLayer layer];
[textLayer setForegroundColor:kWhiteColor]; [textLayer setFontSize:20.0f];
[textLayer setAlignmentMode:kCAAlignmentCenter]; [textLayer setString:@”blah”];
CGRect textRect;
textRect.size = [textLayer preferredFrameSize]; [textLayer setBounds:textRect];
[textLayer setPosition:CGPointMake(50, 12)];
[self addSublayer:textLayer]; return self;
}
```

清单 11-4 LZButtonLayer 的初始化

-string 和-setString 方法（如清单 11-5 所示)获取和传递字符串的值到 CATextLayer 下面。者提供了一个设置 CATextLayer 属性的方便方法。

```
- (NSString*)string; {
return [textLayer string]; }
- (void)setString:(NSString*)string; {
[textLayer setString:string];
CGRect textRect;
textRect.size = [textLayer preferredFrameSize]; [textLayer setBounds:textRect];
}
```

清单 11-5

-setSelected：方法(清单 11-6 所示)给用户提供了一个可视的反馈，以便于他们能看到在应用程序上看到点击的效果。为了展示这个效果，我们通过一个布尔变量的控制，在按钮层上增加和移除 Core Image 的滤镜(CIBoom)。

```
- (void)setSelected:(BOOL)selected {
if (!selected) {
[self setFilters:nil]; return;
}
CIFilter *effect = [CIFilter filterWithName:@”CIBloom”];
[effect setDefaults];
[effect setValue: [NSNumber numberWithFloat: 10.0f] forKey: @”inputRadius”]; [effect setName: @”bloom”];
[self setFilters: [NSArray arrayWithObject:effect]];
}
```

清单 11-6 setSelected 的实现

11.3.3. 接口组建(Interface Builder)

随着 LZButton 层已被设计完后，我们需要创建的下一个就是 AppDelegate。AppDelegate 包含了所有的层；增加他们到窗口的 contentView(内容视图)上，并且接收一个代理回调。

我们需要在 Interface Builder 上做的事情就是改变窗口的 contentView 到一个 LZContentView 的实例上。在类型已经被改变后，就绑定 contentView 的代理给 AppDelegate。这样就能够使 AppDelegate 就收来自 contentView 的鼠标点击事件了。

```
- (void)awakeFromNib {
NSView *contentView = [window contentView]; [contentView setWantsLayer:YES];
```

```
CALayer *contentLayer = [contentView layer]; [contentLayer setBackgroundColor:kBlackColor];
redButton = [LZButtonLayer layer]; [redButton setString:@"Red"];
[redButton setPosition:CGPointMake(60, 22)]; [redButton setMyColor:kRedColor];
[contentLayer addSublayer:redButton];
greenButton = [LZButtonLayer layer]; [greenButton setString:@"Green"];
[greenButton setPosition:CGPointMake(200, 22)]; [greenButton setMyColor:kGreenColor];
[contentLayer addSublayer:greenButton];
blueButton = [LZButtonLayer layer]; [blueButton setString:@"Blue"];
[blueButton setPosition:CGPointMake(340, 22)]; [blueButton setMyColor:kBlueColor];
[contentLayer addSublayer:blueButton];
colorBar = [CALayer layer];
[colorBar setBounds:CGRectMake(0, 0, 380, 20)]; [colorBar setPosition:CGPointMake(200, 100)];
[colorBar setBackgroundColor:kBlackColor]; [colorBar setBorderColor:kWhiteColor]; [colorBar
setBorderWidth:1.0];
[colorBar setCornerRadius:4.0f];
[contentLayer addSublayer:colorBar]; }
```

清单 11-7

清单 11-7，在-awakeFromNib 方法中获取了窗口的 contentView 的一个引用，并且获得了它背后的层。然后我们就获得了一个 contentView 层的引用，通过使用它作为剩下 UI 的 root layer(根层)。

当我们有了 rootLayer 后，下一步就是初始化我们先前创建的 LZButtonLayer，并且分配他们颜色，和设定在 rootLayer 中的位置。当每个按钮是被初始化时，我们就增加他们作为 rootlayer 的子层。

最后，创建一个普通的 CALayer，命名为 colorBar，并且增加到 rootlayer 的子层上。因为 colorBar 是一个 CALayer，它也需要被定义在这里。

图 11-1 展示了接口布局的样子。接下来，就需要给 AppDelegate 增加交互代码了，告诉那个层是要响应事件。开始做这件事，我们需要把 hit test 抽象出来，因为很多地方需要用到。这能够是你重用代码，避免项目中代码的重复。

```
- (LZButtonLayer*)buttonLayerHit {
NSPoint mouseLocation = [NSEvent mouseLocation];
NSPoint translated = [window convertScreenToBase:mouseLocation]; CGPoint point =
NSPointToCGPoint(translated);
CALayer *rootLayer = [[window contentView] layer];
id hitLayer = [rootLayer hitTest:point];
if (![hitLayer isKindOfClass:[LZButtonLayer class]]) {
hitLayer = [hitLayer superlayer];
if (![hitLayer isKindOfClass:[LZButtonLayer class]]) {
return nil; }
}
return hitLayer; }
```

清单 11-8 AppDelegate 中-buttonLayerHit 的实现

当进入 mouseLocation 时，它会返回给我们本地屏幕坐标系的 x 和 y 值。这个坐标系不是我们应用程序使用的坐标系，因此我们需要转换他们到我们应用程序使用的坐标系，就要要用到了 NSWindow 和 NSView 类中的一个方法。因为 CALayer 对象处理的是 CGPoints 而不是 NSPoints，所以我们需要改变窗口坐标系返回的 NSRect，让它成为 CGRect。

现在我们有了正确的鼠标坐标，我们需要发现那个层是在鼠标上。通过在 rootlayer 上调用-hitTest:方法会返回鼠标所在的最深层。

Deepest layer(最深层)被定义为，在点到的位置，它没有了任何子层。例如，如果你点击 LZButtonLayer 上的 text，通过在它上面调用-hitTest:方法 CATextLayer 就会被返回，因为 CATextLayer 在包含 CGpoint 的位置上没有了子层。但是，如果按钮的边缘是被点击，那么 LZButtonLayer 自己会被返回。最后，如果背景跟层是被点击，那么 rootlayer 就会被返回。

如图 11-2.

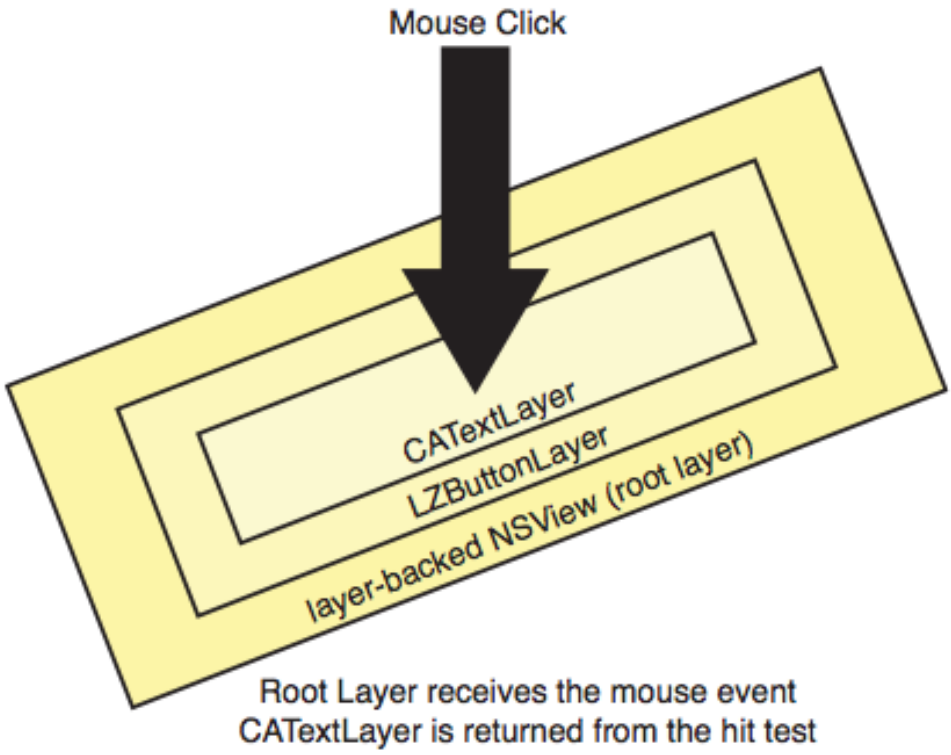

图 11-2 hit test

然而我们关心的是，是否用户在 LZButtonLayer 上点击了。用户会想，我点击到了 LZButtonLayer 上，但是事实上点击到了它的子层上。因而，一个额外的匹配需要增加。如果点击层不是一个 LZButtonLayer 类，那么我们就检查层的父层看它是否是 LZButtonLayer。如果点击的层是，那么的他的父层就返回。如果点击层和它的父层都不是那么就返回 nil。

随着 hit test 的方法被定义，下面该来控制 mouseup 和 mouseDown 的事件响应方法了，如清单 11-9.

```
- (void)mouseDown:(NSEvent*)theEvent; {
[[self buttonLayerHit] setSelected:YES]; }
```

清单 11-9

```
- (void)mouseUp:(NSEvent*)theEvent; {
LZButtonLayer *hitLayer = [self buttonLayerHit]; [hitLayer setSelected:NO];
[colorBar setBackgroundColor:[hitLayer myColor]];
}
```

清单 11-10

当 mouseDown 事件是被接收时，我们需要告诉选择按钮，它是被选中了，这样就改变它的呈现。因为-buttonLayerHit 方法返回 LZButtonLayer 或则 nil，我们就可以通过调用-setSelected:方法，如清单 11-10.

11.3.4. 监控鼠标

上面的代码离开就暴露出一个问题。如果鼠标按钮是在 LZButtonLayer 上被按下，然后松开在其他地方，那么前面的按钮仍然被选择。糟糕的是，如果一个按钮被按下，然后松开在另一个按钮上，这样第一个按钮会被选择，另一个也会被选择。

为了解决这个问题，我们需要重新定义 mouseUp 和 mouseDown 方法，如清单 11-11.

```
- (void)mouseDown:(NSEvent*)theEvent; {
LZButtonLayer *layer = [self buttonLayerHit]; if (!layer) return;
selectedButton = layer;
```

```
[selectedButton setSelected:YES];
NSRect buttonRect = NSRectFromCGRect([selectedButton frame]);
buttonDownTrackingArea = [[NSTrackingArea alloc] initWithRect:buttonRect
options:(NSTrackingMouseEnteredAndExited | NSTrackingActiveInActiveApp |
NSTrackingEnabledDuringMouseDrag | NSTrackingAssumeInside) owner:self userInfo:nil];
[[window contentView] addTrackingArea:buttonDownTrackingArea]; }
```

清单 11-11 更新了 AppDelegate mouseDown 的实现

关键的问题是我们需要决定什么时候鼠标离开了按钮。然而，在整个窗口上，不停的跟踪鼠标的位置是一个繁琐的事件，因此你需要这样做。我们尽可能的少的跟踪鼠标。为了完成这个功能，增加一个 NSTrackingArea 到 contentView 上，当鼠标 mouseDown 事件是被接收时，然后我们就限制 NSTrackingArea 到被按下按钮的矩形区域上。

AppDelegate 是被设定在 NSTrackingArea 这个区域上，无论何时只要鼠标退出或者进入这个区域都会通知 AppDelegate，前提是应用程序要处于活动状态。我们也会告知 NSTrackingArea 去假设我们一开始就在矩形区域中，以便于第一时间可以接收到鼠标退出的事件。

另外增加一个 NSTrackingArea，我们也会保留一个指向按下按钮的指针。这个指针是被用来控制选择状态的开关，从而达到控制 mouseup 的事件。

新增的-mouseDown 方法会引起另外两个方法调用-mouseExited:和-mouseEntered:.定义如清单 11-12.

```
- (void)mouseExited:(NSEvent*)theEvent {
[selectedButton setSelected:NO]; }
- (void)mouseEntered:(NSEvent*)theEvent {
[selectedButton setSelected:YES]; }
```

清单 11 -12

在增加了这些方法后，按钮会根据鼠标的进入到它的区域，来判断是选择还是不被选择。这给用户的视觉反馈就是，鼠标被按下去的时候，还有机会取消掉。

```
- (void)mouseUp:(NSEvent*)theEvent; {
if (!selectedButton) return;
[[window contentView] removeTrackingArea:buttonDownTrackingArea]; [selectedButton setSelected:NO];
LZButtonLayer *hitLayer = [self buttonLayerHit];
if (hitLayer != selectedButton) {
selectedButton = nil;
return; }
CGColorRef newBackgroundColor;
if (CGColorEqualToColor([colorBar backgroundColor], [selectedButton myColor])) {
newBackgroundColor = kBlackColor; } else {
newBackgroundColor = [selectedButton myColor]; }
CABasicAnimation *animation = [CABasicAnimation animationWithKeyPath:@"backgroundColor"];
[animation setFromValue:(id)[colorBar backgroundColor]]; [animation setToValue:(id)newBackgroundColor];
[animation setRemovedOnCompletion:NO];
//[animation setDelegate:self];
[animation setAutoreverses:NO];
[colorBar addAnimation:animation forKey:@"colorChange"]; [colorBar
setBackgroundColor:newBackgroundColor];
}
```

清单 11-13

这些改变就要求 mouseup 的时候，做一些复杂的处理了。首先，如果我们之前没有选择任何按钮，我们就立刻忽略这个事件。这就防止了这个偶发事件，在按钮是被按下的时候，鼠标移动到了按钮上。

下面，移除各宗区域事件，停止了 mouseEntered 和 mouseExisted 事件。现在鼠标已经放下去了，没必要再跟踪这些事件了。

下一步就是发现是否鼠标是按在了同一个 LZButtonLayer 上。我们做这些，通过查询在

鼠标下面的 LZButtonLayer 按钮。如果它不是我们开始的按钮，我们通过设定 selectedButton 为 nil 从而忽略它。这就防止了用户按下一个按钮但是在另外一个按钮松开的错误。

在所有的逻辑检查完成后，就设定颜色条的背景颜色。当做这些时，首先检查背景颜色是否已经是按钮的颜色了。如果是，背景颜色就设定成黑色。否则设定成按钮的颜色。

现在，当应用程序运行时，不仅仅颜色条会随着按钮的按下改变，而且我们还能取消按钮被按下的状态，通过移动鼠标出去，并且释放鼠标。

11.3.5. 键盘事件

键盘和鼠标事件都是相似的。就像鼠标事件一样，仅仅 NSResponder 对象可以接收键盘事件。然而，不像鼠标事件，键盘事件没有点的信息，并且通过被传递到目前的第一响应者。在颜色例子程序中，窗口是第一响应者。因为我们想要接收和处理键盘事件，我们首先要使 LZContentView 可以接收第一响应的状态。我们通过重载-acceptsFirstResponder 方法实现，如清单 11-14.

```
- (BOOL)acceptsFirstResponder {
return YES; }
```

清单 11-14

就像清单 11-14 中 mouseUp 和 mouseDown 事件一样，我们想要控制键盘事件在代理上，代替直接在视图上。因而，-keyUp:方法传递事件到代理上，如清单 11-15.

```
- (void)keyUp:(NSEvent *)theEvent {
[delegate keyUp:theEvent]; }
```

清单 11-15

返回到 AppDelegate，我们需要在开始时，给 contentView 第一响应状态，以便于一开始就可以接收鼠标事件。为了做这些，需要在 -awakeFromNib 中调用 [window setFirstResponder:contentView]方法。

现在事件是被传递到我们想要的地方了，该如何处理他们那。当-keyUp:事件是被触发时，我们想要基于被按下按键设定背景颜色。如清单 11-16.

```
- (void)keyUp:(NSEvent *)theEvent {
CGColorRef newColor;
if ([[theEvent charactersIgnoringModifiers] isEqualToString:@”r”]) {
newColor = kRedColor;
} else if ([[theEvent charactersIgnoringModifiers] isEqualToString:@”g”]) {
newColor = kGreenColor;
} else if ([[theEvent charactersIgnoringModifiers] isEqualToString:@”b”]) {
newColor = kBlueColor;
} else {
[super keyUp:theEvent];
return; }
if (CGColorEqualToColor([colorBar backgroundColor], newColor)) { newColor = kBlackColor;
}
[colorBar setBackgroundColor:newColor]; }
```

清单 11-16

我们可以测试下这个三个按键 r,g 和 b。如果将要来的事件不能匹配这 3 个按键，我们就忽略这个事件，传给上一层响应链，然后就返回此方法。如果它匹配了，我们就看目前的颜色是不是要设定的颜色，不是就设定，是的就去掉这个颜色。

11.3.6. 层后面的视图

到目前为止，我们讨论的整个用户接口使用的是核心动画，通过一个简单的 root 的视图支持它。另外一个情况是工作在背后的视图是不同于单独的一个层。

不像单一的 NSView 设计，后面的视图都是 NSResponder 的子类。因而，它们可能在更

底层的等级上就可以接收鼠标和键盘事件。然而，你需要考虑这些事情，当增加用户交互在层上的时候，并且背后是视图时。

键盘的输入

前面提到过，因为键盘的输入没有点的概念，应用程序需要保持跟踪那个 NSResponder 是键盘事件的接受者。这些通过响应链可以做。当我们开发自定义的 NSView 的对象时，我们需要意识到响应链并且正确的控制它。如果我们接收了一个事件，但是我们不需要控制它，我们需要传递给下一个响应链，以便于潜在的父类链可以控制它。如果你不传递这些事件，我们就可能中断了某些事件像键盘快捷键，等等。

鼠标的坐标系

鼠标事件比键盘事件更容易控制。当一个自定义的 NSView 接收鼠标事件时，它要保证属于 NSView 或者它的子类。然而，坐标系是否需要转换需要关心。就像前面清单 11-8 讨论过的，[NSEvent mouseLocation]返回的是屏幕的坐标系。这首先需要转变坐标系到窗口的坐标系上，然后再转换到接收事件的视图上。因为每个 NSResponder 都有它内部的格子，在响应点击之前我们需要我们工作在正确的坐标系上。

11.3.7. 总结

这一章介绍了在核心动画环境中捕捉用户输入的概念。使用本章提到的概念，你可以建立一个复杂的用户体验。

尽管利用层后面的视图，更容易开发交互接口。你也可以创建整个接口在单独的层上，或者建立一个自定义的层，通过传递给它鼠标和按键事件，这样就允许他们用轻量级的方法控制需要展现的东西了。

点击这里访问：DevDiv.com 移动开发论坛