iPhone与iPad应用开发课程
精通iOS开发

第十三讲 触摸事件和手势

主讲人：关东升

eorient@sina.com

主要知识点

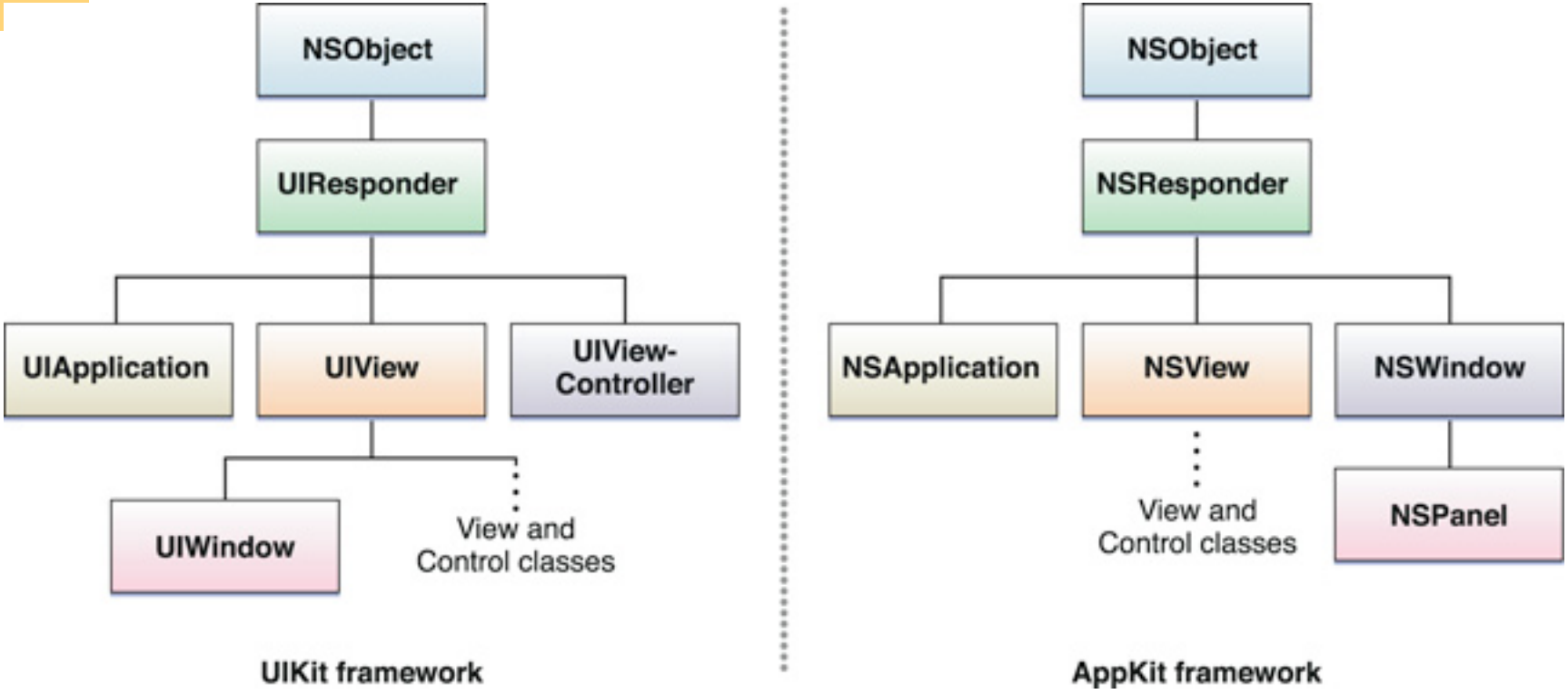
- 事件概述
- 触摸事件
- 手势

事件概述

- 事件是当用户手指触击屏幕及在屏幕上移动时，系统不断发送给应用程序的对象。
- 系统将事件按照特定的路径传递给可以对其进行处理的对象。
- 在iOS中，一个UITouch对象表示一个触摸，一个UIEvent对象表示一个事件。事件对象中包含与当前多点触摸序列相对应的所有触摸对象，还可以提供与特定视图或窗口相关联的触摸对象。

响应者对象

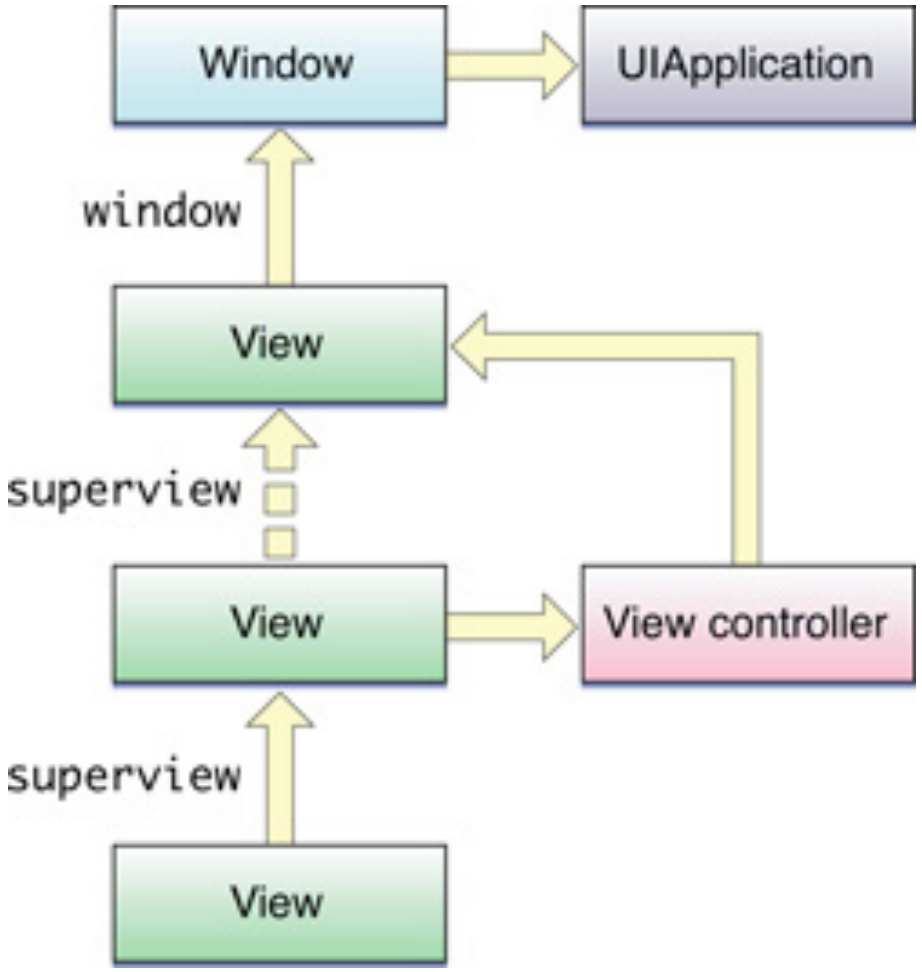
- 响应者对象是可以响应事件并对其进行处理的对象。UIResponder是所有响应者对象的基类，它不仅为事件处理，而且也为常见的响应者行为定义编程接口。UIApplication、UIView、和所有从UIView派生出来的UIKit类（包括UIWindow）都直接或间接地继承自UIResponder类。
- 第一响应者是应用程序中当前负责接收触摸事件的响应者对象（通常是一个UIView对象）。UIWindow对象以消息的形式将事件发送给第一响应者，使其有机会首先处理事件。如果第一响应者没有进行处理，系统就将事件（通过消息）传递给响应者链中的下一个响应者，看看它是否可以进行处理。


NSObject
UIResponder
UIApplication
UIView
UIView-Controller
UIWindow
View and Control classes
UIKit framework
NSObject
NSResponder
NSApplication
NSView
NSWindow
View and Control classes
NSPanel
AppKit framework

响应者链

- 响应链是一个响应者对象的连接序列，事件或动作消息（或菜单编辑消息）依次传递。它允许响应者对象把事件处理的职责转交给其它更高层的对象。应用程序通过向上传递一个事件来查找合适的处理对象。因为点击检测视图也是一个响应者对象，应用程序在处理触摸事件时也可以利用响应链。响应链由一系列的下一个响应者组成。

响应者链


Window
UIApplication
window
View
superview
View
View controller
superview
View

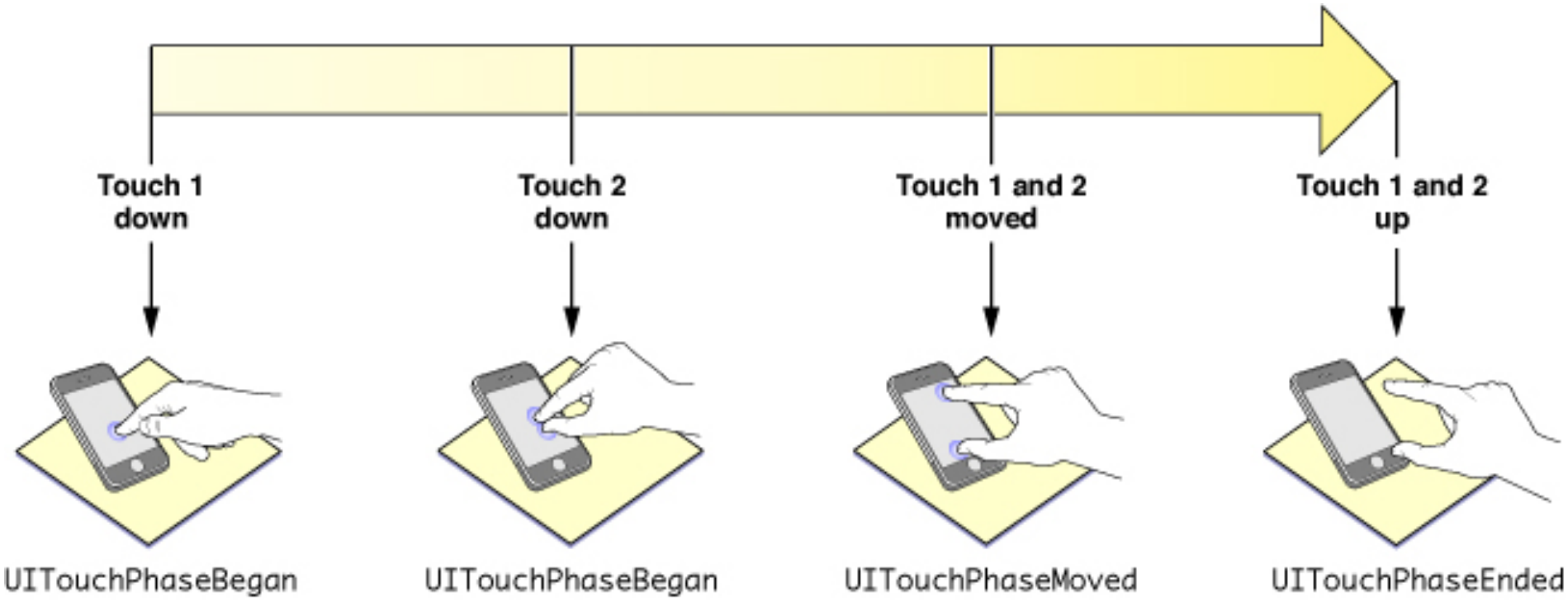
响应者链处理原则

- 1. 点击检测视图或者第一响应者传递事件或动作消息给它的视图控制器(如果它有的话)；如果没有一个视图控制器，就传递给它的父视图。
- 2. 如果一个视图或者它的视图控制器不能处理这个事件或动作消息，它将传递给该视图的父视图。
- 3. 在这个视图层次中的每个后续的父视图遵循上述的模式，如果它不能处理这个事件或动作消息的话。
- 4. 最顶层的视图如果不能处理这个事件或动作消息，就传递给UIWindow对象来处理。
- 5. 如果UIWindow 对象不能处理，就传给单件应用程序对象UIApplication。
- 如果应用程序对象也不能处理这个事件或动作消息，将抛弃它。

触摸事件

- 触摸信息有时间和空间两方面，时间方面的信息称为阶段（phrase），表示触摸是否刚刚开始、是否正在移动或处于静止状态，以及何时结束—也就是手指何时从屏幕抬起。
- 触摸信息还包括当前在视图或窗口中的位置信息，以及之前的位置信息（如果有的话）。当一个手指接触屏幕时，触摸就和某个窗口或视图关联在一起，这个关联在事件的整个生命周期都会得到维护。

触摸事件的阶段


Touch 1 down
Touch 2 down
Touch 1 and 2 moved
Touch 1 and 2 up
UITouchPhaseBegan
UITouchPhaseBegan
UITouchPhaseMoved
UITouchPhaseEnded

事件处理方法

- 在给定的触摸阶段中，如果发生新的触摸动作或已有的触摸动作发生变化，应用程序就会发送这些消息：
  - 当一个或多个手指触碰屏幕时，发送touchesBegan:withEvent:消息。
  - 当一个或多个手指在屏幕上移动时，发送touchesMoved:withEvent:消息。
  - 当一个或多个手指离开屏幕时，发送touchesEnded:withEvent:消息。
  - 当触摸序列被诸如电话呼入这样的系统事件所取消时，发送touchesCancelled:withEvent:消息。

触摸事件实例

```
#import <UIKit/UIKit.h>


@interface TouchView : UIView {

}

@end
```

EventInfo

```
@implementation TouchView
- (void)logTouchInfo:(UITouch *)touch {
    CGPoint locInSelf = [touch locationInView:self];
    CGPoint locInWin = [touch locationInView:nil];
    NSLog(@"    touch.locationInView = {%2.3f, %2.3f}", locInSelf.x, locInSelf.y);
    NSLog(@"    touch.locationInWin = {%2.3f, %2.3f}", locInWin.x, locInWin.y);
    NSLog(@"    touch.phase = %d", touch.phase);
    NSLog(@"    touch.tapCount = %d", touch.tapCount);
    NSLog(@"    touch.phase = %f", touch.timestamp);
}
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    NSLog(@"touchesBegan - touch count = %d", [touches count]);
    for(UITouch *touch in event.allTouches) {
        [self logTouchInfo:touch];
    }
}
```

说明

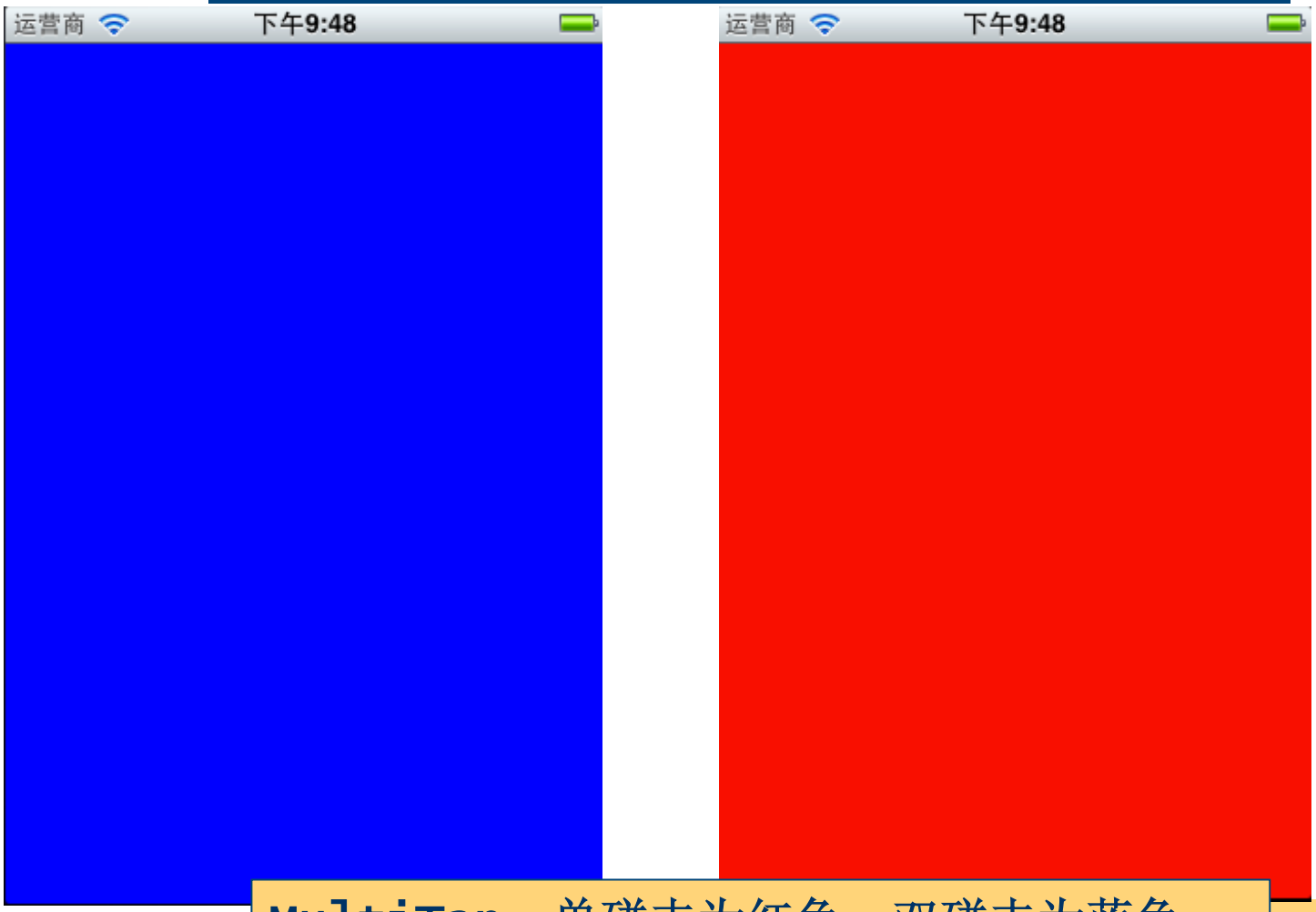
- touch.phase，触摸事件的阶段。
- touch.tapCount，触摸事件的轻碰次数，可以判断双击事件。
- UIEvent 的allTouches方法，可以获得触摸点的集合，可以判断多点触摸事件。

```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
    NSLog(@"touchesMoved - touch count = %d", [touches count]);
    for(UITouch *touch in event.allTouches) {
        [self logTouchInfo:touch];
    }
}
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    NSLog(@"touchesEnded - touch count = %d", [touches count]);
    for(UITouch *touch in event.allTouches) {
        [self logTouchInfo:touch];
    }
}
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event {
    NSLog(@"touchesCancelled - touch count = %d", [touches count]);
    for(UITouch *touch in event.allTouches) {
        [self logTouchInfo:touch];
    }
}
```

手势

- 手势在iPhone中很重要，手势就是手触摸屏幕的方式。
  - 单碰击
  - 双碰击
  - 多点触摸（合拢和展开）
  - 轻抚
  - ……

单碰击和双碰击



MultiTap、单碰击为红色，双碰击为蓝色

h文件

```
#import <UIKit/UIKit.h>

@interface MultiTapView : UIView {

}

@end
```

```
@implementation MultiTapView
- (void)turnBlue {
  self.backgroundColor = [UIColor blueColor];
}
- (void)turnRed {
  self.backgroundColor = [UIColor redColor];
}
}
```

```
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
  UITouch *touch = [touches anyObject];
  if(touch.tapCount == 2) {
    [[self class] cancelPreviousPerformRequestsWithTarget:self
                                                 selector:@selector(turnRed)
                                                   object:nil];
  }
}
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
  UITouch *touch = [touches anyObject];
  if(touch.tapCount == 1) {
    [self performSelector:@selector(turnRed) withObject:nil afterDelay:0.10f];
  }
  if(touch.tapCount == 2) {
    [self turnBlue];
  }
}
```

说明

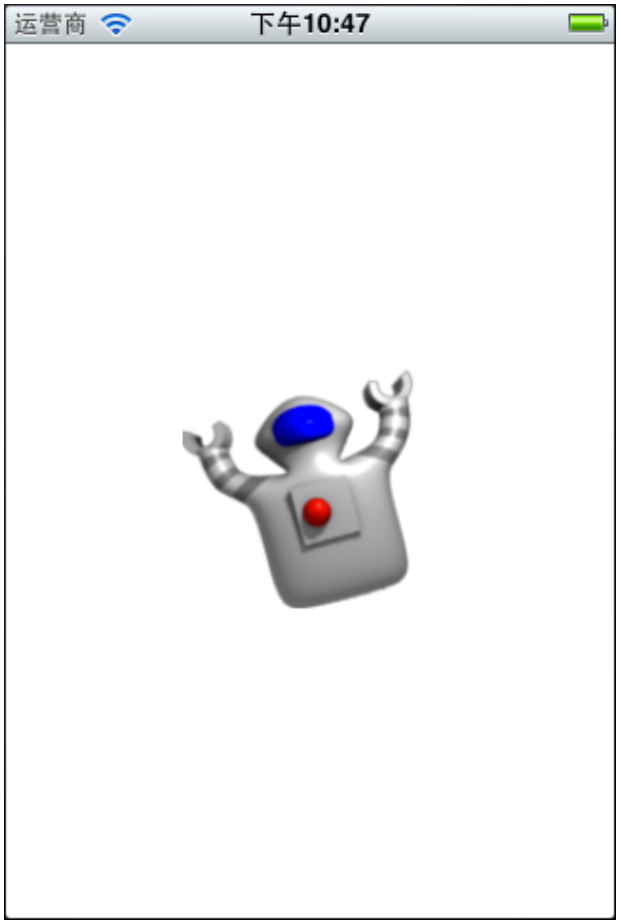
- [self performSelector:@selector(turnRed) withObject:nil afterDelay:0.10f];

是在0.1秒后调用turnRed方法。

- [[self class] cancelPreviousPerformRequestsWithTarget:self selector:@selector(turnRed) object:nil];

是取消调用方法turnRed。

多点触摸（合拢和展开）



PinchZoom

PinchZoomView .h文件

```
#import <UIKit/UIKit.h>
#import <QuartzCore/QuartzCore.h>

@interface PinchZoomView : UIView {
    CALayer *robotLayer;
    CGFloat previousDistance;
    CGFloat zoomFactor;
    BOOL pinchZoom;
}

@property(nonatomic, retain) CALayer *robotLayer;

@end
```

m文件

```
- (void)awakeFromNib {
  self.robotLayer = [CALayer layer];
  UIImage *image = [UIImage imageNamed:@"Robot.png"];
  self.robotLayer.contents = (id)[image CGImage];
  self.robotLayer.bounds = CGRectMake(0.0f, 0.0f, image.size.width, image.size.height);
  self.robotLayer.position = CGPointMake(CGRectGetMidX(self.bounds), CGRectGetMidY
(self.bounds));
  [self.layer addSublayer:self.robotLayer];
  pinchZoom = NO;
  previousDistance = 0.0f;
  zoomFactor = 1.0f;
}
```

说明

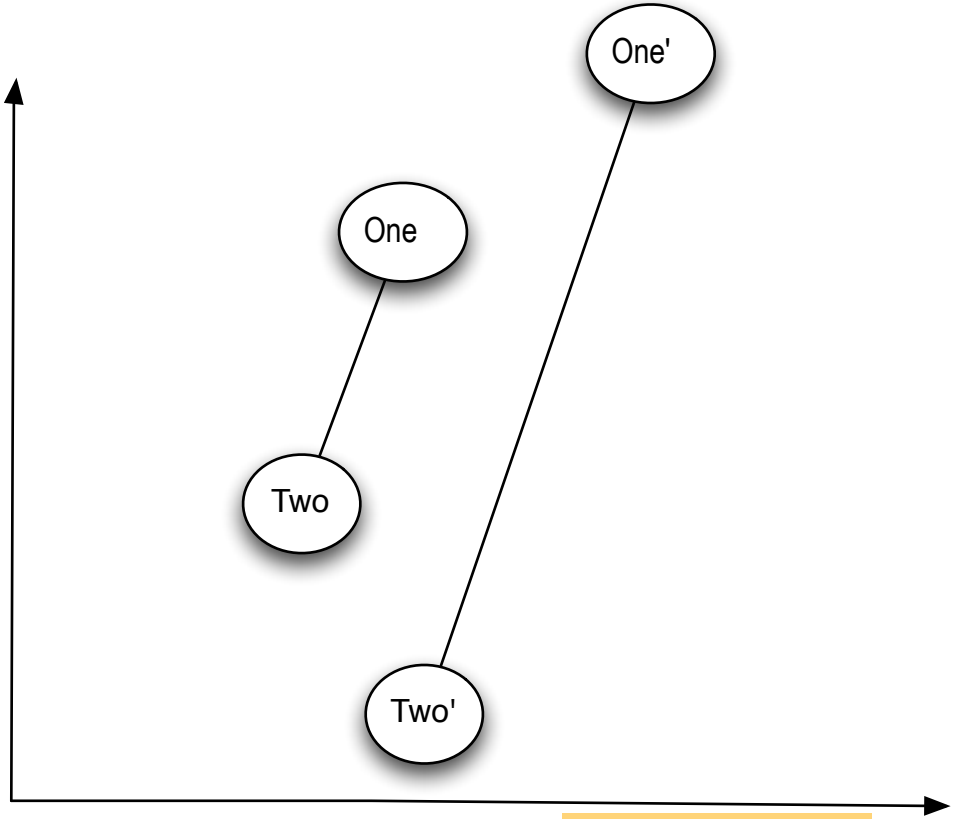
- awakeFromNib当nib文件被加载的时候，加载器会发送一个awakeFromNib的消息到nib文件中的每个对象，每个对象都可以定义自己的awakeFromNib方法来响应这个消息，执行一些必要的操作。也就是说通过nib文件创建view对象是执行awakeFromNib 。
- robotLayer是 CALayer 对象，本例子中我们把图片对象添加到robotLayer对象中。使用 CALayer 需要引入 <QuartzCore/QuartzCore.h>头文件和添加QuartzCore.framework框架。

m文件

```
//START:code.PinchZoomView.touchesBegan
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
  if(event.allTouches.count == 2) {
    pinchZoom = YES;
    NSArray *touches = [event.allTouches allObjects];
    CGPoint pointOne = [[touches objectAtIndex:0] locationInView:self];
    CGPoint pointTwo = [[touches objectAtIndex:1] locationInView:self];
    previousDistance = sqrt(pow(pointOne.x - pointTwo.x, 2.0f) +
                            pow(pointOne.y - pointTwo.y, 2.0f));
} else {
    pinchZoom = NO;
  }
}
```

说明

- previousDistance 是获得两个点的距离。
- pow是平方函数。
- sqrt是开平方根函数。



m文件

```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
  if(YES == pinchZoom && event.allTouches.count == 2) {
    NSArray *touches = [event.allTouches allObjects];
    CGPoint pointOne = [[touches objectAtIndex:0] locationInView:self];
    CGPoint pointTwo = [[touches objectAtIndex:1] locationInView:self];
    CGFloat distance = sqrt(pow(pointOne.x - pointTwo.x, 2.0f) +
                            pow(pointOne.y - pointTwo.y, 2.0f));
          zoomFactor += (distance - previousDistance) / previousDistance;
          zoomFactor = fabs(zoomFactor);
          previousDistance = distance;
          self.robotLayer.transform =
              CATransform3DMakeScale(zoomFactor, zoomFactor, 1.0f);
  }
}
```

```
//START:code.PinchZoomView.touchesEnded
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
        if(event.allTouches.count != 2) {
                pinchZoom = NO;
                previousDistance = 0.0f;
        }
}
//END:code.PinchZoomView.touchesEnded

- (void)dealloc {
        self.robotLayer = nil;
        [robotLayer release];
        [super dealloc];
}
```