14 Dec 2016 • on iOS

详解iOS触摸事件与手势识别

本文主要想讲的是触摸事件和手势混合使用的一个问题，但作为知识储备，还是把两者再单独介绍一下。两者的基本知识点都是iOS开发文档或者参考其他博客的，算是一个总结，文章最后会标出参考链接。

iOS的事件有`Touch Events`、`Motion Events`、`Remote Events`，最常见的是触摸事件`Touch Events`。触摸事件除了是view来处理，还有高级的手势可以处理。所以，本文分别来讲讲触摸事件和手势，并结合例子讲讲两者混合使用的问题。

由于本文讲的东西有点繁杂，特在此列一个目录，方便大家有一个清晰的第一印象。

- 手势识别与事件响应混用

UITouch对象

一个手指第一次点击屏幕，就会生成一个`UITouch`对象，到手指离开时销毁。当我们有多个手指触摸屏幕时，会生成多个`UITouch`对象。`UITouch`对象可以表明触摸的位置、状态，其状态的类型有:

```
typedef NS_ENUM(NSInteger, UITouchPhase) {
    UITouchPhaseBegan,             // whenever
a finger touches the surface.
    UITouchPhaseMoved,             // whenever
a finger moves on the surface.
    UITouchPhaseStationary,        // whenever
a finger is touching the surface but hasn't
moved since the previous event.
    UITouchPhaseEnded,             // whenever
a finger leaves the surface.
    UITouchPhaseCancelled,         // whenever
a touch doesn't end but we need to stop
tracking (e.g. putting device to face)
};
```

UIEvent对象

一个`UIEvent`对象代表iOS的一个事件。一个触摸事件定义为第一个手指开始触摸屏幕到最后一个手指离开屏幕。所以，一个`UIEvent`对象实际上对应多个`UITouch`对象。

响应链

响应链可以理解为一种虚拟的链表，每一个节点是一个`UIResponder`对象。`UIResponder`是事件接收与处理的基
类，`UIApplication`、`UIViewController`和`UIView`都继承自`UIResponder`。`UIResponder`

提供了几个事件处理的方法：

```
- (void)touchesBegan:(NSSet<UITouch *>
*)touches withEvent:(nullable UIEvent *)event;
- (void)touchesMoved:(NSSet<UITouch *>
*)touches withEvent:(nullable UIEvent *)event;
- (void)touchesEnded:(NSSet<UITouch *>
*)touches withEvent:(nullable UIEvent *)event;
- (void)touchesCancelled:(NSSet<UITouch *>
*)touches withEvent:(nullable UIEvent *)event;
```

`UIResponder`对象之间的联系靠`nextResponder`指针。

```
#if UIKIT_DEFINE_AS_PROPERTIES
@property(nonatomic, readonly, nullable)
UIResponder *nextResponder;
#else
- (nullable UIResponder*)nextResponder;
#endif
```

一个触摸事件到达真正处理它的对象时经过了一个搜索路径，这就是响应链的一部分。事件沿着这个响应链一直传递，直到碰到可以处理这个事件的`UIResponder`对象或者到达响应链的末尾(UIApplication)。

响应链的构造规则如下：

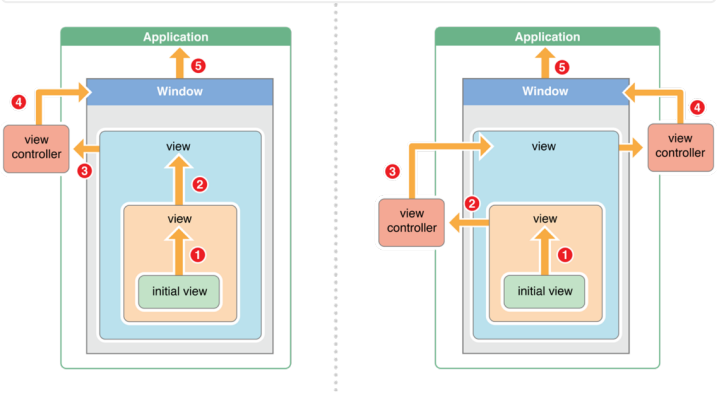
1. 程序启动时，UIApplication会生成一个单例，并会关联一个APPDelegate。APPDelegate作为整个响应链的根建立起来，UIApplication的nextResponser为APPDelegate。

2. 程序启动后，任何的UIWindow被创建时，UIWindow内部都会把nextResponser设置为UIApplication单例。

3. UIWindow初始化rootViewController，rootViewController的nextResponser会设置为UIWindow。

4. UIViewController初始化View，View的nextResponser会设置为rootViewController。

5. AddSubView时，如果subView不是ViewController的View,那么subView的nextResponser会被设置为superView。否则就是 subView -> subView.VC -> superView。



有了这个响应链，事件就可以按照这个路径逐级传递了。当前的对象不能处理这个事件，就交给nextResponser，一直到UIApplication单例。如果仍然不能处理，就丢弃。

实际应用中，一个事件都是由一个响应对象来处理，不会继续向下传递，不过有了上面的知识，我们是可以手动传递的，做我们想做的事情。

Hit-Testing

当我们触摸屏幕时，到底应该由哪个对象最先响应

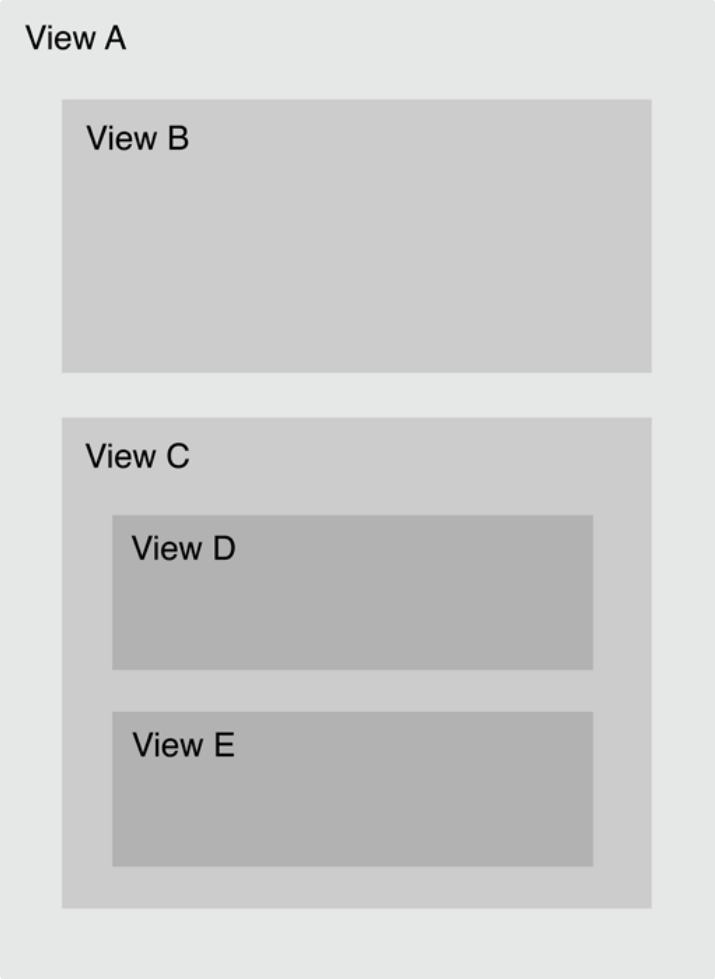
这个事件呢？这就需要去探测，这个过程称为`Hit-Testing`，最后的结果称为`hit-test view`。涉及到两个方法是：

```
//先判断点是否在View内部，然后遍历subViews
- (UIView *)hitTest:(CGPoint)point withEvent:
(UIEvent *)event;
//判断点是否在这个View内部
- (BOOL)pointInside:(CGPoint)point withEvent:
(UIEvent *)event;
```

`Hit-Testing`是一个递归的过程，每一步监测触摸位置是否在当前view中，如果是，就递归监测subviews；否则，返回nil。 递归的根节点是UIWindow，对subviews的遍历顺序按照 **后添加的先遍历** 原则。大致过程的代码如下：

```
- (nullable UIView *)hitTest:(CGPoint)point
withEvent:(nullable UIEvent *)event {
    //判断是否合格
    if (!self.hidden && self.alpha > 0.01 &&
self.isUserInteractionEnabled) {
        //判断点击位置是否在自己区域内
        if ([self pointInside: point
withEvent:event]) {
            UIView *attachedView;
            for (int i = self.subviews.count -
1; i >= 0; i--) {
                UIView *view  =
self.subviews[i];
                //对子view进行hitTest
                attachedView =  [view
hitTest:point withEvent:event];
                if (attachedView)
                    break;
            }
            if (attachedView)  {
                return attachedView;
            } else {
                return self;
            }
        }
    }
    return nil;
}
```

来看最经典的示例图：



当点击E时，探测的步骤是：

> 1 触摸点在A的范围内，所以继续探测A的subView，既view B和view C。
> 2 触摸点不在view B里，在view C里，所以继续探测C的subView D 和 E。
> 3 触摸点不在D里，在E里，E已经是最低级的View，故返回E。

综上，我们可以看出有两个几乎相反的链，一是**响应链**，一是**探测链**。 有触摸事件时，先依赖探测链来确定响应链的开始节点(UIResponder对象)，然后依赖响应链来确定最终处理事件的对象。

手势识别

手势是Apple提供的更高级的事件处理技术，可以完成更多更复杂的触摸事件，比如旋转、滑动、长按等。基类是`UIGestureRecognizer`，派生的类有：

Gesture	UIKit Class
Tapping (any number of taps)	UITapGestureRecognizer
Pinching in and out (for zooming a view)	UIPinchGestureRecognizer
Panning or dragging	UIPanGestureRecognizer
Swiping (in any direction)	UISwipeGestureRecognizer
Rotating (fingers moving in opposite directions)	UIRotationGestureRecognizer
Long press (also known as “touch and hold”)	UILongPressGestureRecognizer

手势绑定到一个View上，一个View上可以绑定多个手势。`UIGestureRecognizer`
同`UIResponder`一样也有四个方法

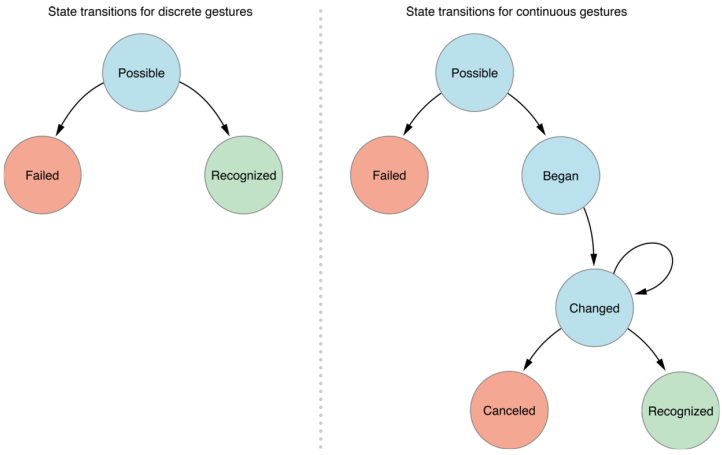
```
//UIGestureRecognizer
- (void)touchesBegan:(NSSet<UITouch *>
*)touches withEvent:(nullable UIEvent *)event;
- (void)touchesMoved:(NSSet<UITouch *>
*)touches withEvent:(nullable UIEvent *)event;
- (void)touchesEnded:(NSSet<UITouch *>
*)touches withEvent:(nullable UIEvent *)event;
- (void)touchesCancelled:(NSSet<UITouch *>
*)touches withEvent:(nullable UIEvent *)event;
```

手势会在以上四个方法中去对手势的State做更改，手势的State表明当前手势是识别还是失败等等。比如单击手势会在`touchesBegan`时记录点击位置，然后在`touchesEnded`判断点击次数、时间、是否移动过，最后得出否识别该手势。这几个方法一般在自定义手势里面使用。 手势的State有：

```
typedef NS_ENUM(NSInteger,
UIGestureRecognizerState) {
    UIGestureRecognizerStatePossible,   // the
recognizer has not yet recognized its gesture,
but may be evaluating touch events. this is the
default state
    UIGestureRecognizerStateBegan,      // the
recognizer has received touches recognized as
the gesture. the action method will be called
at the next turn of the run loop
    UIGestureRecognizerStateChanged,    // the
recognizer has received touches recognized as a
change to the gesture. the action method will
be called at the next turn of the run loop
    UIGestureRecognizerStateEnded,      // the
recognizer has received touches recognized as
the end of the gesture. the action method will
be called at the next turn of the run loop and
the recognizer will be reset to
UIGestureRecognizerStatePossible
    UIGestureRecognizerStateCancelled,  // the
recognizer has received touches resulting in
the cancellation of the gesture. the action
method will be called at the next turn of the
run loop. the recognizer will be reset to
UIGestureRecognizerStatePossible
    UIGestureRecognizerStateFailed,     // the
recognizer has received a touch sequence that
can not be recognized as the gesture. the
action method will not be called and the
recognizer will be reset to
UIGestureRecognizerStatePossible
// Discrete Gestures – gesture recognizers that
recognize a discrete event but do not report
changes (for example, a tap) do not transition
through the Began and Changed states and can
not fail or be cancelled
    UIGestureRecognizerStateRecognized =
UIGestureRecognizerStateEnded // the recognizer
has received touches recognized as the gesture.
the action method will be called at the next
```

```
turn of the run loop and the recognizer will be
reset to UIGestureRecognizerStatePossible
};
```

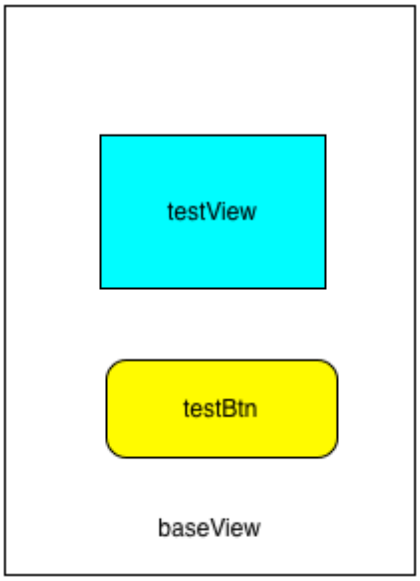
手势在这些状态之间变化，形成了一个有限状态机：



左侧是非连续手势(比如单击)的状态机，右侧是连续手势(比如滑动)的状态机。所有的手势的开始状态都是UIGestureRecognizerStatePossible。 非连续的手势要么识别成功(UIGestureRecognizerStateRecognized)，要么识别失败(UIGestureRecognizerStateFailed)。 连续的手势识别到第一个手势时，变成UIGestureRecognizerStateBegan，然后变成UIGestureRecognizerStateChanged，并且不断地在这个状态下循环，当用户最后一个手指离开view时，变成UIGestureRecognizerStateEnded，当然如果手势不再符合它的模式的时候，状态也可能变成UIGestureRecognizerStateCancelled。

手势识别与事件响应混用

重点来了！ 前面我们知道触摸事件可以通过响应链来传递与处理，也可以被绑定在view上的手势识别和处理。那么这两个一起用会出现什么问题？我们来看一个简单的例子。



图中baseView 有两个subView，分别是testView和testBtn。我们在baseView和testView都重载`touchsBegan:withEvent`、`touchsEnded:withEvent`、`touchsMoved:withEvent`、`touchsCancelled:withEve`方法，并且在baseView上添加单击手势，action名为`tapAction`，给testBtn绑定action名为`testBtnClicked`。主要代码如下：

```
//baseView
- (void)viewDidLoad {
    [super viewDidLoad];
    UITapGestureRecognizer *tap =
[[UITapGestureRecognizer alloc]
initWithTarget:self
action:@selector(tapAction)];
    [self.view addGestureRecognizer:tap];
    ...
    [_testBtn addTarget:self
action:@selector(testBtnClicked)
forControlEvents:UIControlEventTouchUpInside];
}

- (void)touchesBegan:(NSSet<UITouch *>
*)touches withEvent:(UIEvent *)event {
    NSLog(@"=========> base view touchs
Began");
}
- (void)touchesMoved:(NSSet<UITouch *>
*)touches withEvent:(UIEvent *)event {
     NSLog(@"=========> base view touchs
Moved");
```

Foolish Boy

Stay hungry,
Stay foolish!

Blog

```
}
- (void)touchesEnded:(NSSet<UITouch *>
*)touches withEvent:(UIEvent *)event {
    NSLog(@"=========> base view touchs
Ended");
}
- (void)touchesCancelled:(NSSet<UITouch *>
*)touches withEvent:(UIEvent *)event {
    NSLog(@"=========> base view touchs
Cancelled");
}
- (void)tapAction {
    NSLog(@"=========> single Tapped");
}
- (void)testBtnClicked {
    NSLog(@"=========> click testbtn");
}


//test view
- (void)touchesBegan:(NSSet<UITouch *>
*)touches withEvent:(UIEvent *)event {
    NSLog(@"=========> test view touchs
Began");
}
- (void)touchesMoved:(NSSet<UITouch *>
*)touches withEvent:(UIEvent *)event {
    NSLog(@"=========> test view touchs
Moved");
}
- (void)touchesEnded:(NSSet<UITouch *>
*)touches withEvent:(UIEvent *)event {
    NSLog(@"=========> test view touchs
Ended");
}
- (void)touchesCancelled:(NSSet<UITouch *>
*)touches withEvent:(UIEvent *)event {
    NSLog(@"=========> test view touchs
Cancelled");
}
```

情景A： 单击baseView，输出结果为：

```
=========> base view touchs Began
=========> single Tapped
=========> base view touchs Cancelled
```

情景B： 单击testView，输出结果为：

```
==========> test view touchs Began
==========> single Tapped
==========> test view touchs Cancelled
```

情景C： 单击testBtn, 输出结果为：

```
==========> click testbtn
```

情景D： 按住testView，过5秒后或更久释放，输出结果为：

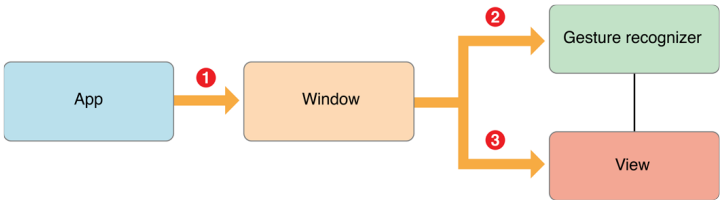
```
==========> test view touchs Began
==========> test view touchs Ended
```

何解啊?

原来，手势有更高的优先级来识别一个触摸事件。

> Gesture Recognizers Get the First Opportunity to Recognize a Touch
> A window delays the delivery of touch objects to the view so that the gesture recognizer can analyze the touch first. During the delay, if the gesture recognizer recognizes a touch gesture, then the window never delivers the touch object to the view, and also cancels any touch objects it previously sent to the view that were part of that recognized sequence.

既触摸事件首先传递到手势上，如果手势识别成功，就会取消事件的继续传递，否则，事件还是会被响应链处理。具体地，系统维持了与响应链关联的所有手势，事件首先发给这些手势，然后再发给响应链。



这就可以解释了情景A和B了，首先我们的单击事件传递到了tap手势上了，不过这个是手势识别需要一点时间，手势还在Possible状态的时候事件传递到了响应链的第一个响应对象（baseView或者testView），于是就回调用它们的`touchsBegan:withEvent:`方法，然后手势识别成功，就会去cancel之前传递到的所有响应对象，于是就会调用它们的`touchsCancelled:withEvent:`方法。

按照这个道理，情景C也应该是这样才对啊！这就涉及到一个响应级别的问题了，iOS开发文档里说到

> In iOS 6.0 and later, default control actions prevent overlapping gesture recognizer behavior. For example, the default action for a button is a single tap. If you have a single tap gesture recognizer attached to a button's parent view, and the user taps the button, then the button's action method receives the touch event instead of the gesture recognizer. This applies only to gesture recognition that overlaps the default action for a control, which includes:
> A single finger single tap on a UIButton, UISwitch, UISegmentedControl, UIStepper,and UIPageControl.
> A single finger swipe on the knob of a UISlider, in a direction parallel to the slider.
> A single finger pan gesture on the knob of a UISwitch, in a direction parallel to the switch.

所以testBtn的action会覆盖手势。

在情景D中，由于长按住testView不释放，tap手势就会识别失败，然后就可以继续正常传递给testView处理。

假如实际应用中我们在testview中有UITableView,上面的方式就不能点击UITableViewCell了，除非长按cell再释放。 要解决这个问题,需要了解UIGestureRecognizer的`cancelsTouchsInView`属性：

```
//default is YES. causes
touchesCancelled:withEvent: or
pressesCancelled:withEvent: to
//be sent to the view for all touches or
presses recognized as part of this gesture
immediately
//before the action method is called.
@property(nonatomic) BOOL cancelsTouchesInView;
```

既在手势识别成功之后，是否取消传递给响应链的触摸事件。

我们设置tap的cancelsTouchsInView为NO，那么输出结果就变成

```
//A
==========> base view touchs Began
==========> single Tapped
==========> base view touchs Ended
//B
==========> test view touchs Began
==========> single Tapped
==========> test view touchs Ended
//C
==========> single Tapped
==========> click testbtn
//D
==========> test view touchs Began
==========> test view touchs Ended
```

此时，baseView 和testView上的触摸事件就可以完整执行。

顺便再提一下`delaysTouchesBegan`属性：

```
// default is NO.  causes all touch or press events to be delivered to
//the target view only after this gesture has failed recognition. set to
//YES to prevent views from processing any touches or presses that
//may be recognized as part of this gesture
@property(nonatomic) BOOL delaysTouchesBegan;
```

既延迟传递事件给view，我们设置tap的delaysTouchesBegan为YES，那么输出结果就变成

```
//A
==========> single Tapped
//B
==========> single Tapped
//C
==========> click testbtn
//D
==========> test view touchs Began
==========> test view touchs Ended
```

所以事件先被手势识别了，就不再传递给响应链了。

参考链接

iOS开发者文档

深入浅出iOS事件机制

iOS 触摸事件处理详解