

# 从UIEvent到HitTest到RunLoop

## 1. 事件是如何被app感知的

这一节先总结整体流程，第一节内容摘录于一位博主的博客

### 1.1 物理层面

点击屏幕时，人体的电场让手指和触摸屏之间形成一个耦合电容，电容是直接导体，手指从接触点吸走一个个很小的电流，控制器会根据这个电流产生的位置进行计算，最后得出触摸点的位置

### 1.2 操作系统分发事件

- 电容传感器产生的Touch Event，将交给IOKit.framework处理封装成IOHIDEvent对象
- SpringBoard.app将会接收封装好的IOHIDEvent对象，经过逻辑判断后分发，它会判断前台有没有程序运行，有就会把封装好的事件用mach port机制传递给该进程的主线程

### 1.3 IOHIDEvent到UIEvent

- 应用程序主线程的RunLoop申请了一个mach port监听IOHIDEvent的source1事件，回调方法是\_\_IOHIDEventSystemClientQueueCallback()，内部又进一步分发Source0事件，Source0事件都是自定义的，非基于port端口，包括触摸，滚动，selector选择器事件，它的回调是\_\_UIApplicationHandleEventQueue()，将接收到的IOHIDEvent事件对象封装成UIEvent
- UIEvent封装好后，会调用UIApplication的实例分发sendEvent，将UIEvent传递给UIWindow做一些逻辑工作，比如触摸事件产生在哪个视图上

### 1.4 HitTest寻找响应者

- Source0回调中将封装好的触摸事件UIEvent（里面有多多个UITouch对象），传递给视图UIWindow，开始寻找最佳响应者，也就是HitTest
1. 事件自UIApplication -> UIWindow -> View -> ... -> subview传递
  2. 越靠后添加的视图响应程度越高
  3. 如果视图不想响应，则传递给它上层的视图，如果能响应，继续传递，某个视图能响应，又没有子视图，就是最佳响应者
  4. 在这个过程中，UIEvent中的UITouch会不断打上标签，比如HitTestView是谁，关联了什么Gesture

代码块

```
1 - (UIView *)hitTest:(CGPoint)point withEvent:(UIEvent *)event{
```

```

2    // 1. 前置条件要满足
3    if (self.userInteractionEnabled == NO ||
4        self.hidden == YES ||
5        self.alpha <= 0.01) return nil;
6
7    // 2. 判断点是否在视图内部 这是最起码的 note point 是在当前视图坐标系的点位置
8    if ([self pointInside:point withEvent:event] == NO) return nil;
9
10   // 3. 现在起码能确定当前视图能够是响应者 接下去询问子视图
11   int count = (int)self.subviews.count;
12   for (int i = count - 1; i >= 0; i--)
13   {
14       // 子视图
15       UIView *childView = self.subviews[i];
16
17       // 点需要先转换坐标系
18       CGPoint childP = [self convertPoint:point toView:childView];
19       // 子视图开始询问
20       UIView *fitView = [childView hitTest:childP withEvent:event];
21       if (fitView) {
22           return fitView;
23       }
24   }
25
26   return self;
27 }

```

这是一个HitTest的经典伪代码实现，其实就做了几件事

1. 不符合条件的不响应
  - a. Alpha < 0.1,.userInteractionEnabled == NO, hidden == YES
2. pointInside返回NO不响应
3. 触摸点在这个视图里，是一个可能的响应者
  - a. 遍历所有的子类
  - b. 转换坐标，对每个子类HitTest

## 2. 疑问和实验

### 2.1 疑问

1. HitTest返回了结果，就确定是这个target来响应事件吗？
2. 如果这是一个Gesture，CancelsTouchesInView赋值不同，有什么影响？
3. 手势和Touches方法谁的响应优先级更高？

#### 4. TouchesBegan一定会调用吗？

## 2.2 HitTest的行为

因为有以上疑问，所以有了下面的实验

代码块

```
1  UIWindow
2  |—— UIViewA
3  |   |—— UIViewA1
4  |   |   |—— UIViewA1a
5  |   |   |—— UIViewA2
6  |   |       |—— UIViewA2a
7  |   |       |—— UIViewA2b
8  |—— UIViewB
9  |   |—— UIViewB1
10 |   |   |—— UIViewB1a
11 |   |   |—— UIViewB1b
12 |—— UIViewC
13 |   |—— UIViewC1
14 |   |—— UIViewC2
15 |       |—— UIViewC2a
16
```

假设有以下继承层级，且ViewA2b将会命中HitTest，递归记录如下

### hitTest 调用顺序：

#### 1. UIWindow hitTest

- 遍历 subviews: A, B, C（假定排列顺序：A, B, C）
- 坐标转换，判断 point 是否在每个 subview 区域
- point 在 UIViewA，递归调用 A

#### 2. UIViewA hitTest

- subviews: A1, A2
- point 在 A2，递归 A2

#### 3. UIViewA2 hitTest

- subviews: A2a, A2b
- point 在 A2b，递归 A2b

#### 4. UIViewA2b hitTest

- 无 subview

- pointInside 为 YES，返回 self (A2b)

回溯：

- UIViewA2b 返回自身
- UIViewA2 收到 UIViewA2b，直接返回
- UIViewA 收到 UIViewA2，返回
- UIWindow 收到 UIViewA，返回

## 2.3 必要的准备

使用一些必备的工具来hook几个关键方法，来看看UIKit的行为

代码块

```
1  + (void)load {
2      // 确保只 swizzle 一次
3      static dispatch_once_t onceToken;
4      dispatch_once(&onceToken, ^{
5          [self swizzleHitTest];
6      });
7  }
8
9  + (void)swizzleHitTest {
10     Class class = [self class];
11
12     SEL originalSelector = @selector(hitTest:withEvent:);
13     SEL swizzledSelector = @selector(xm_swizzled_hitTest:withEvent:);
14
15     Method originalMethod = class_getInstanceMethod(class, originalSelector);
16     Method swizzledMethod = class_getInstanceMethod(class, swizzledSelector);
17
18     BOOL didAddMethod =
19     class_addMethod(class,
20                     originalSelector,
21                     method_getImplementation(swizzledMethod),
22                     method_getTypeEncoding(swizzledMethod));
23
24     if (didAddMethod) {
25         class_replaceMethod(class,
26                             swizzledSelector,
27                             method_getImplementation(originalMethod),
28                             method_getTypeEncoding(originalMethod));
29     } else {
30         method_exchangeImplementations(originalMethod, swizzledMethod);
31     }
```

```

32 }
33
34 - (UIView *)xm_swizzled_hitTest:(CGPoint)point withEvent:(UIEvent *)event {
35     NSLog(@"HitTest: %@", self.class);
36     NSLog(@"NextResponder: %@", self.nextResponder.class);
37     return [self xm_swizzled_hitTest:point withEvent:event];
38 }

```

## 对UIResponder中的方法进行hook

代码块

```

1  // 实现四个 swizzled 方法
2  - (void)xm_swizzled_touchesBegan:(NSSet<UITouch *> *)touches withEvent:(UIEvent
    *)event {
3      NSLog(@"Touches Began: %@", self);
4      [self xm_swizzled_touchesBegan:touches withEvent:event];
5  }
6  - (void)xm_swizzled_touchesEnded:(NSSet<UITouch *> *)touches withEvent:(UIEvent
    *)event {
7      NSLog(@"Touches Ended: %@", self);
8      [self xm_swizzled_touchesEnded:touches withEvent:event];
9  }
10 - (void)xm_swizzled_touchesMoved:(NSSet<UITouch *> *)touches withEvent:(UIEvent
    *)event {
11     NSLog(@"Touches Moved: %@", self);
12     [self xm_swizzled_touchesMoved:touches withEvent:event];
13 }
14 - (void)xm_swizzled_touchesCancelled:(NSSet<UITouch *> *)touches withEvent:
    (UIEvent *)event {
15     NSLog(@"Touches Cancelled: %@", self);
16     [self xm_swizzled_touchesCancelled:touches withEvent:event];
17 }

```

## 2.4 实验

如果A2b不能响应，怎么办？能响应又如何响应？

代码块

```

1  {
2      UIView *viewa = [[UIView alloc] init];
3      viewa.backgroundColor = XMBlackColor;
4      viewa.size = CGSizeMake(150, 150);
5      viewa.center = self.view.center;
6      [self.view addSubview:viewa];

```

```

7
8     UIView *viewb = [[UIView alloc] init];
9     viewb.backgroundColor = UIColor.blueColor;
10    viewb.size = CGSizeMake(100, 100);
11    viewb.center = self.view.center;
12    [self.view addSubview:viewb];
13 }

```

这是一个viewController的简单场景，self.view, viewa, viewb都没有添加事件会怎么样？

代码块

```

1  HitTest: UIWindow
2  NextResponder: UIWindowScene
3  HitTest: UITransitionView
4  NextResponder: UIWindow
5  HitTest: UIDropShadowView
6  NextResponder: UITransitionView
7  HitTest: UILayoutContainerView
8  NextResponder: UINavigationController
9  HitTest: UINavigationControllerTransitionView
10 NextResponder: UILayoutContainerView
11 HitTest: UIViewControllerWrapperView
12 NextResponder: UINavigationControllerTransitionView
13 HitTest: UIView
14 NextResponder: XMDashboardViewController
15 HitTest: UIView
16 NextResponder: UIView
17 HitTest: UIWindow
18 NextResponder: UIWindowScene
19 HitTest: UITransitionView
20 NextResponder: UIWindow
21 HitTest: UIDropShadowView
22 NextResponder: UITransitionView
23 HitTest: UILayoutContainerView
24 NextResponder: UINavigationController
25 HitTest: UINavigationControllerTransitionView
26 NextResponder: UILayoutContainerView
27 HitTest: UIViewControllerWrapperView
28 NextResponder: UINavigationControllerTransitionView
29 HitTest: UIView
30 NextResponder: XMDashboardViewController
31 HitTest: UIView
32 NextResponder: UIView
33 Touches Began: <UIView: 0x103714d80; frame = (165 416; 100 100);
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:

```

```
0x10358eb80>>
34  Touches Ended: <UIView: 0x103714d80; frame = (165 416; 100 100);
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
    0x10358eb80>>
```

在这里可以清晰的看到HitTest的递归调用栈，还有两个结论

- 不管有没有控件能处理事件，HitTest都会正常进行，一直到返回一个最合适的target为止，然后会调用这个target的touches方法
- UIResponder的响应链，除了UIViewController.view的nextResponder是viewController外，其它都是自己的superView，一直到UIWindow
- HitTest多次打印，第二次才会正式调用targetView相关的touches方法

1. B添加手势，A不添加，view不添加会怎么样，这种情况就不用说了，肯定是viewB响应事件
2. 但是如果A添加手势，而b不添加，会怎么样？

代码块

```
1  HitTest: UIWindow
2  NextResponder: UIWindowScene
3  HitTest: UITransitionView
4  NextResponder: UIWindow
5  HitTest: UIDropShadowView
6  NextResponder: UITransitionView
7  HitTest: UILayoutContainerView
8  NextResponder: UINavigationController
9  HitTest: UINavigationControllerTransitionView
10 NextResponder: UILayoutContainerView
11 HitTest: UIViewControllerWrapperView
12 NextResponder: UINavigationControllerTransitionView
13 HitTest: UIView
14 NextResponder: XMDashboardViewController
15 HitTest: UIView
16 NextResponder: UIView
17 HitTest: UIWindow
18 NextResponder: UIWindowScene
19 HitTest: UITransitionView
20 NextResponder: UIWindow
21 HitTest: UIDropShadowView
22 NextResponder: UITransitionView
23 HitTest: UILayoutContainerView
24 NextResponder: UINavigationController
25 HitTest: UINavigationControllerTransitionView
26 NextResponder: UILayoutContainerView
```

```

27 HitTest: UIViewWrapperView
28 NextResponder: UINavigationController
29 HitTest: UIView
30 NextResponder: UINavigationController
31 HitTest: UIView
32 NextResponder: UIView
33 Touches Began: <UIView: 0x104f1cd80; frame = (165 416; 100 100);
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
    0x104ef10c0>>
34 Touches Ended: <UIView: 0x104f1cd80; frame = (165 416; 100 100);
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
    0x104ef10c0>>

```

- 可以看到，HitTest最终命中了ViewB，证明了逆序遍历的正确性
- 但是ViewA并没有响应这次事件

这个时候，给view添加事件，而ab都没有事件，会怎么样？

代码块

```

1 Touches Began: <UIView: 0x13832cd80; frame = (165 416; 100 100);
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
    0x1381fee40>>
2 <UITapGestureRecognizer: 0x1383843c0; state = Ended; view = <UIView:
    0x13832ca80>; target= <(<action=_handleGesture:, target=
    <XMDashboardViewController 0x105de2fc0>>>>
3 Touches Cancelled: <UIView: 0x13832cd80; frame = (165 416; 100 100);
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
    0x1381fee40>>

```

- 这一次view正常响应了方法
- 这说明事件响应和view之间的遮盖无关，HitTest命中后，UIKit会尝试调用touchesBegan:，如果没有实现，将会实现默认的方法，也就是找到它的nextResponder，看它能不能响应
- view本身的touches方法没有触发，只是响应了手势，touches方法只会在命中目标上响应
- 当tapGesture被事件流捕获后，touchesCanceled调用，而不是调用touchesEnded，说明事件没有正常结束

为了证明和遮盖无关，并且UIKit发现HitTest找到的targetView无法响应事件后的行为，有以下实验，新增一个view是controller.view的子视图，是ab的父视图，还是controller.view响应事件，最后结果如下



代码块

```
1  HitTest: UINavigationController
2  NextResponder: UINavigationController
3  HitTest: UINavigationController
4  NextResponder: UINavigationController
5  HitTest: UINavigationController
6  NextResponder: UINavigationController
7  HitTest: UINavigationController
8  NextResponder: UINavigationController
9  HitTest: UINavigationController
10 NextResponder: UINavigationController
11 Touches Began: <UINavigationController: 0x156329080; frame = (165 416; 100 100);
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
    0x156309260>>
12 <UITapGestureRecognizer: 0x156388640; state = Ended; view = <UINavigationController:
    0x156328c00>; target= <(_handleGesture:, target=
    <UINavigationController 0x104cab8b0>)>>
13 Touches Cancelled: <UINavigationController: 0x156329080; frame = (165 416; 100 100);
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
    0x156309260>>
```

- touches还是正常调用，targetView正常
- 即使再添加一层视图层级，响应事件的还是controller.view
- 这证明了touches发现targetView没有手势，无法响应事件后，会沿着nextResponder链像底层传递，一直到能响应事件或找不到响应者为止

## 2.5 事件穿透

还是以上层级，如何让viewA命中，而不是同层级的viewB，而且viewB要正常展示？

其实很简单

1. viewB.userInteractionEnabled = NO;
2. 把viewB改成自定义子类，重写HitTest方法

代码块

```
1  - (UIView *)hitTest:(CGPoint)point withEvent:(UIEvent *)event {
2      if (/* some condition: 透明、空白区域、不需响应 */) {
3          return nil; // 事件递归传到下层
4      }
5      return [super hitTest:point withEvent:event];
6  }
```

## 2.6 CancelsTouchesInView的意义?

还是以上层级，view添加手势，且cancelsTouchesInView == NO，ab不添加手势

代码块

```
1  Touches Began: <UIView: 0x106f3cd80; frame = (165 416; 100 100);  
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:  
    0x106d92f00>>  
2  <UITapGestureRecognizer: 0x106f94500; state = Ended; cancelsTouchesInView = NO;  
    view = <UIView: 0x106f3ca80>; target= <(action=_handleGesture:, target=  
    <XMDashboardViewController 0x1063b6070>)>>  
3  Touches Ended: <UIView: 0x106f3cd80; frame = (165 416; 100 100);  
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:  
    0x106d92f00>>
```

最后可以看到，touches ended被调了，而不是canceled

给view添加一个longPressGesture，并且设置cancelsTouchesInView == NO，实验结果如下：

代码块

```
1  Touches Began: <UIView: 0x152320d80; frame = (165 416; 100 100);  
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:  
    0x1522f8be0>>  
2  <UILongPressGestureRecognizer: 0x152398000; state = Began; cancelsTouchesInView  
    = NO; view = <UIView: 0x152320a80>; target= <(action=_handleLong:, target=  
    <XMDashboardViewController 0x103e71a70>)>>; numberOfTapsRequired = 0;  
    minimumPressDuration = 0.5>  
3  <UILongPressGestureRecognizer: 0x152398000; state = Changed;  
    cancelsTouchesInView = NO; view = <UIView: 0x152320a80>; target=  
    <(action=_handleLong:, target=<XMDashboardViewController 0x103e71a70>)>>;  
    numberOfTapsRequired = 0; minimumPressDuration = 0.5>  
4  <UILongPressGestureRecognizer: 0x152398000; state = Changed;  
    cancelsTouchesInView = NO; view = <UIView: 0x152320a80>; target=  
    <(action=_handleLong:, target=<XMDashboardViewController 0x103e71a70>)>>;  
    numberOfTapsRequired = 0; minimumPressDuration = 0.5>  
5  Touches Moved: <UIView: 0x152320d80; frame = (165 416; 100 100);  
    backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:  
    0x1522f8be0>>  
6  <UILongPressGestureRecognizer: 0x152398000; state = Changed;  
    cancelsTouchesInView = NO; view = <UIView: 0x152320a80>; target=  
    <(action=_handleLong:, target=<XMDashboardViewController 0x103e71a70>)>>;  
    numberOfTapsRequired = 0; minimumPressDuration = 0.5>
```

```

7  Touches Moved: <UIView: 0x152320d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1522f8be0>>
8  <UILongPressGestureRecognizer: 0x152398000; state = Changed;
   cancelsTouchesInView = NO; view = <UIView: 0x152320a80>; target=
   <(action=_handleLong:, target=<XMDashboardViewController 0x103e71a70>)>;
   numberOfTapsRequired = 0; minimumPressDuration = 0.5>
9  Touches Moved: <UIView: 0x152320d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1522f8be0>>
10 <UILongPressGestureRecognizer: 0x152398000; state = Ended; cancelsTouchesInView
    = NO; view = <UIView: 0x152320a80>; target= <(action=_handleLong:, target=
    <XMDashboardViewController 0x103e71a70>)>; numberOfTapsRequired = 0;
    minimumPressDuration = 0.5>
11 Touches Ended: <UIView: 0x152320d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1522f8be0>>

```

设置cancelsTouchesInView == YES，实验结果如下

#### 代码块

```

1  Touches Began: <UIView: 0x1032f0d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1032d0d40>>
2  Touches Moved: <UIView: 0x1032f0d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1032d0d40>>
3  Touches Moved: <UIView: 0x1032f0d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1032d0d40>>
4  Touches Moved: <UIView: 0x1032f0d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1032d0d40>>
5  Touches Moved: <UIView: 0x1032f0d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1032d0d40>>
6  Touches Moved: <UIView: 0x1032f0d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1032d0d40>>
7  Touches Moved: <UIView: 0x1032f0d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1032d0d40>>
8  Touches Moved: <UIView: 0x1032f0d80; frame = (165 416; 100 100);
   backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:
   0x1032d0d40>>

```

```
9 Touches Moved: <UIView: 0x1032f0d80; frame = (165 416; 100 100);  
  backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:  
  0x1032d0d40>>  
10 <UILongPressGestureRecognizer: 0x103350000; state = Began; view = <UIView:  
  0x1032f0a80>; target= <(action=_handleLong:, target=<XMDashboardViewController  
  0x102692870>>>; numberOfTapsRequired = 0; minimumPressDuration = 0.5>  
11 Touches Cancelled: <UIView: 0x1032f0d80; frame = (165 416; 100 100);  
  backgroundColor = UIExtendedSRGBColorSpace 0 0 1 1; layer = <CALayer:  
  0x1032d0d40>>  
12 <UILongPressGestureRecognizer: 0x103350000; state = Changed; view = <UIView:  
  0x1032f0a80>; target= <(action=_handleLong:, target=<XMDashboardViewController  
  0x102692870>>>; numberOfTapsRequired = 0; minimumPressDuration = 0.5>
```

touchesMove在LongPressGesture响应前正常调用，一旦找到响应的gesture，宿主view，也就是HitTest找到的targetView的手势会立即被取消

结论：

- **cancelsTouchesInView = YES（默认）：**
  - 当手势被识别成功（即 gesture recognizer 的 state 从 possible 变为 recognized/ended）时，UIKit 立即给宿主 view 发送 touchesCancelled:withEvent:。
  - 此后，该 touch 流不会再进入 touchesMoved、touchesEnded，直接终止。
  - 所以 move/ended 都不会再被调用，只有 cancel。
- **cancelsTouchesInView = NO：**
  - 手势识别成功后，touches 流不会被取消。
  - 宿主 view 依然可以收到完整的 touchesMoved、touchesEnded，与手势 action 并存。

