深入iOS事件处理层次及原理分析、响应链

找不到工作的iOS 关注
拥有4钻

0.396 2017.10.12 16:46:24 字数 1324 阅读 914

1. iOS事件有哪一些

运动事件

- 传感器、计数器、陀螺仪

远程控制事件

- 线控耳机

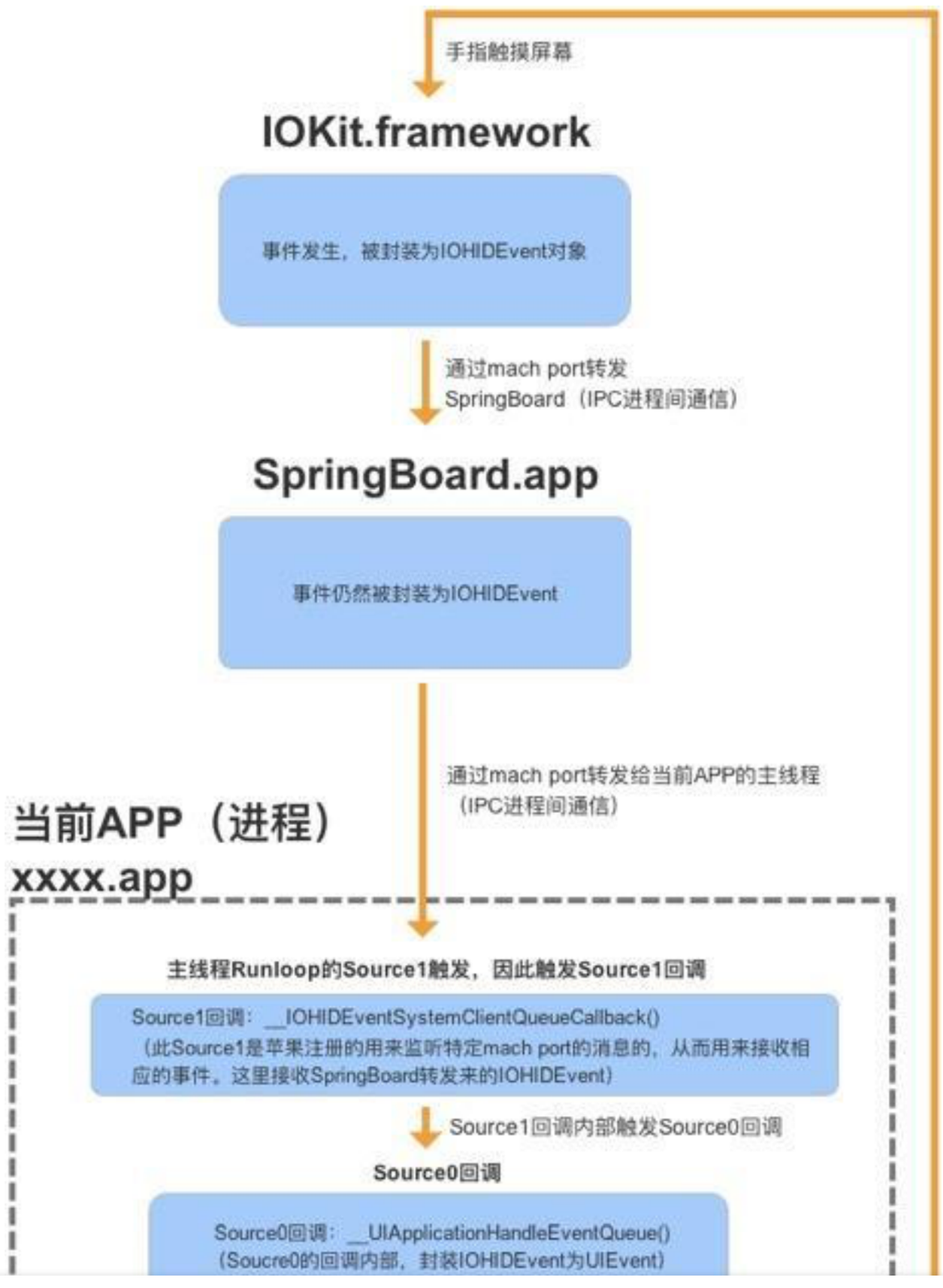
触摸事件

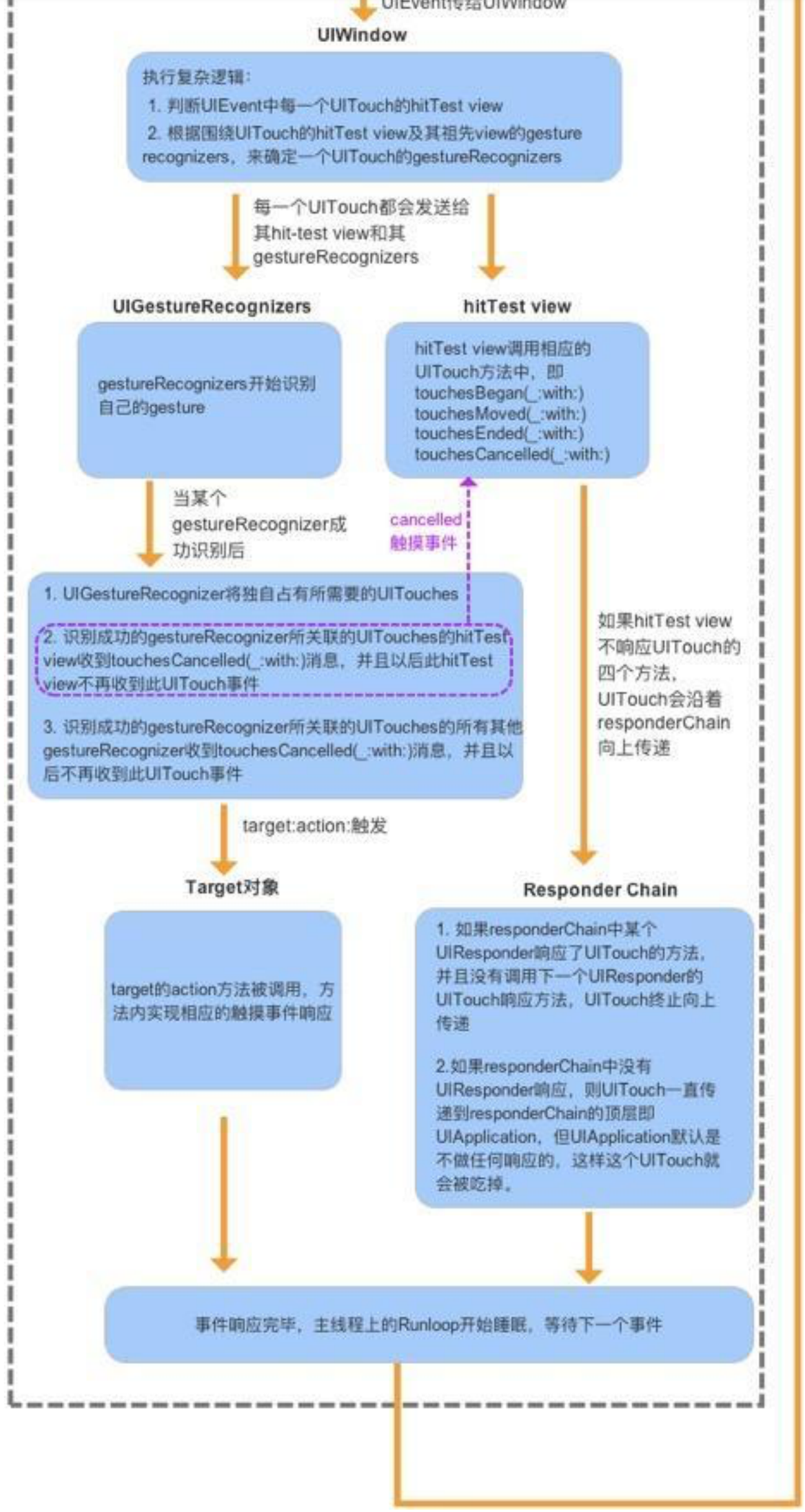
- 本文核心分析

2. 事件传递和响应

2.1 原理分析

- 当我们手指触碰到屏幕的时候，事件传递和响应的流程是怎么样的呢
- 事件的流程图





流程.png

- IOKit.framework 为系统内核的库
- SpringBoard.app 相当于手机的桌面
- Source1 主要接收系统的消息
- Source0 - UIApplication - UIWindow
- **从UIWindow 开始步骤，见下图**

window流程.png

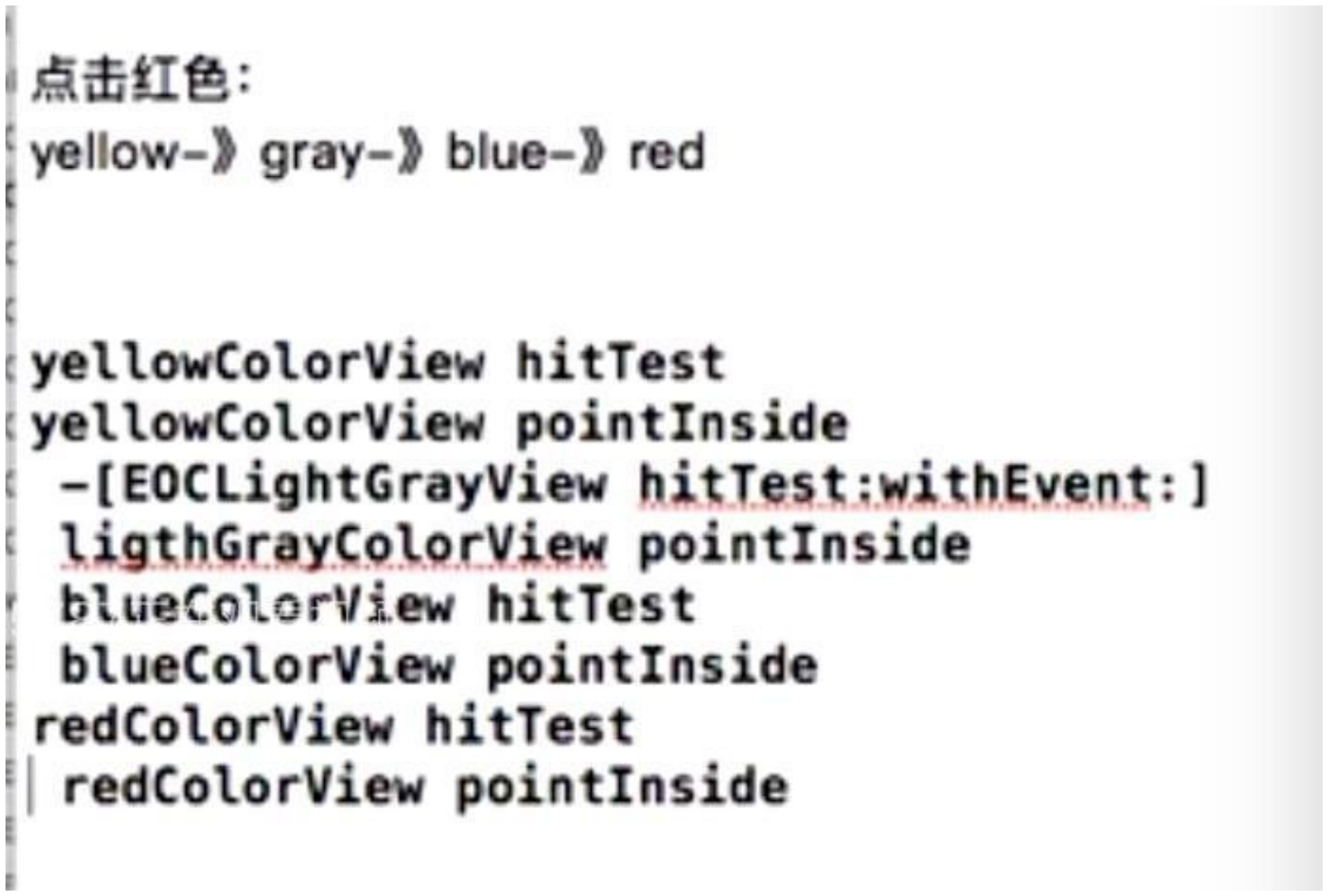
- 比如我们在self.view 上依次添加view1、view2、view3（3个view是同级关系），那么系统用 `hitTest` 以及 `pointInside` 时会先从view3开始便利，如果 `pointInside` 返回 `YES` 就继续遍历view3的subviews（如果view3没有子视图，那么会返回view3），如果 `pointInside` 返回 `NO` 就开始便利view2。
- 反序遍历，最后一个添加的subview开始。也算是一种算法优化

2.2 HitTest 、pointInside

- 上一段层级关系的简单示例代码

```
EOCLightGrayView *grayView = [[EOCLightGrayView alloc] initWithFrame:CGRectMake(50.f, 10

    redView = [[EOCRedView alloc] initWithFrame:CGRectMake(0.f, 0.f, 120.f, 100.f)];

    EOCBlueView *blueView = [[EOCBlueView alloc] initWithFrame:CGRectMake(140.f, 100.f, 1

    EOCYellowView *yellowView = [[EOCYellowView alloc] initWithFrame:CGRectMake(50.f, 360

    [self.view addSubview:grayView];
    [grayView addSubview:redView];
    [grayView addSubview:blueView];
    [self.view addSubview:yellowView];
```

布局.png



打印.png

- 点击`red`，由于`yellow`与`grey`同级，`yellow`比`grey`后添加，所以先打印`yellow`，由于触摸点不在`yellow`内，打印`grey`，然后遍历`grey`，打印他的两个`subviews`
- 通过在`HitTest`返回`nil`，`pointInside`并没有执行，我们可以得知，`pointInside`调用顺序你在`HitTest`之后的。
- `pointInside`的 参数 `:(CGPoint)poinit`的值是以自身为坐标系的，判断点是否view内的范围是以view自身的`bounds`为范围，而非`frame`
- 如果在`grey`的`hitTest`返回`[super hitTest:point event:event]`，则会执行`gery.subviews`的遍历（`subviews 的 hitTest 与 pointInside)`，`grey`的`pointInside`是判断触摸点是否在`grey`的`bounds`内（不准确），`grey`的`hitTest`是判断是否需要遍历他的`subviews`.
- `pointInside`只是在执行`hitTest`时，会在`hitTest`内部调用的一个方法
- `pointInside`只是辅助`hitTest`的关系
- `hitTest`是一个递归函数

2.3 hitTest 内部实现代码还原

```objc
- (UIView *)hitTest:(CGPoint)point withEvent:(UIEvent *)event
{
    ///hitTest：判断pointInside，是不是在view里？是的话，遍历，不是的话返回nil;假设我就是点击灰色的，
    NSLog(@"%s",__func__);

    NSArray *subViews = [[self.subviews reverseObjectEnumerator] allObjects];
    UIView *tmpView = nil;
    for (UIView *view in subViews) {

        CGPoint convertedPoint = [self convertPoint:point toView:view];
        if ([view pointInside:convertedPoint withEvent:event]) {

            tmpView = view;
            break;

        }

```

```
    }

    if (tmpView) {
        return tmpView;
    } else if([self pointInside:point withEvent:event]) {

        return self;
    } else {

        return nil;

    }


    return [self hitTest:point event:event];  //redView

    ///这里是hitTest的逻辑

    ///alpha(<=0.01)、userInterActionEnabled(NO)、hidden(YES)  pointInside返回的为NO

}
```

2.4 实战之扩大button点击区域

bigbutton.png

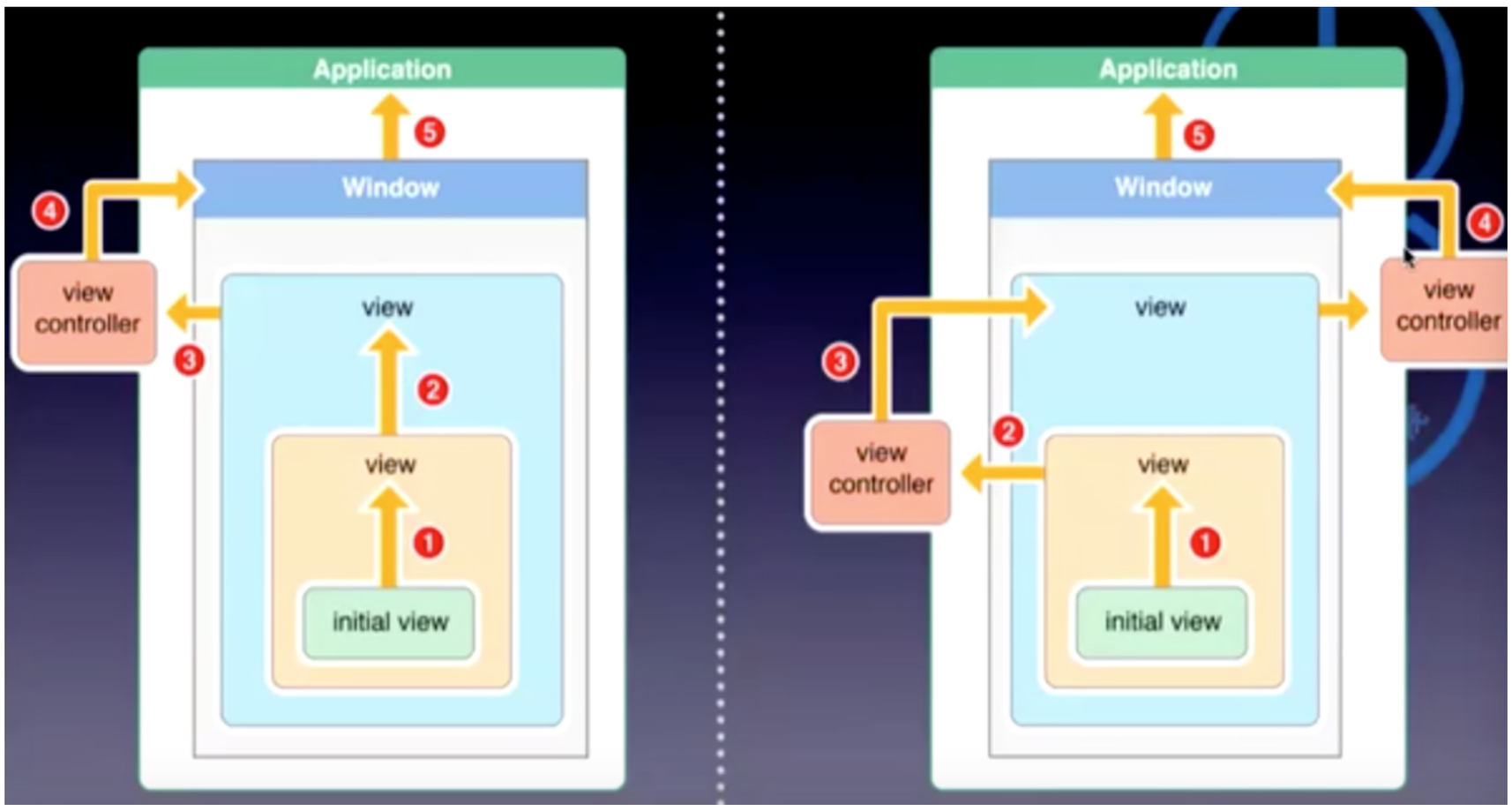
- 红色为button
- 蓝色为放大后的目标点击区域
- 稍微注意是在bounds的基础上修改
- button 内部的 `hitTest` 通过 `pointInside` 的确认，来决定是否返回自己

```
@implementation EOCCustomButton

- (BOOL)pointInside:(CGPoint)point withEvent:(UIEvent *)event
{
    NSLog(@"%s", __func__);

    //扩大它的响应范围
    CGRect frame = [self getScaleFrame];
    return CGRectContainsPoint(frame, point);

   //  return [super pointInside:point withEvent:event];
}

- (CGRect)getScaleFrame {

    CGRect rect = self.bounds;

    if (rect.size.width < 40.f) {
        rect.origin.x -= (40-rect.size.width)/2;
    }

    if (rect.size.height < 40.f) {
        rect.origin.y -= (40-rect.size.height)/2;
    }

```

```
27      rect.size.width = 40.f;
28      rect.size.height = 40.f;
29
30      return rect;
    }
```

2.5 UIRespond 与 响应链的组成

- 当我们通过`hitTest`找到视图后，我们产生的touch事件，他是怎么一层层响应的?



respondch.png

- 响应链是通过 `nextResponder` 属性组成的一个链表
- 点击的 view 有 superView ,nextResponder 就是 superView ;
- `view.nextResponder .nextResponder` 是viewController 或者是 view.superView. view
- `view.nextResponder .nextResponder. nextResponder` 是 UIWindow (非严谨,便于理解)
- `view.nextResponder .nextResponder. nextResponder. nextResponder` 是UIApplication 、UIAppdelate、直到nil (非严谨,便于理解)
- `touch`事件就是根据响应链的关系来层层调用（我们**重写touch** 要记得 `super` 调用，不然**响应链会中断**）
- 比如我们监听`self.view`的`touch`事件，也是因为`subviews`的`touch`都在同一个响应链里

3. 手势事件

3.1 手势 与 hitTest 的关系

- 相同上面的学习我们可以推测出，手势的响应也得必须经过`hitTest`先找到视图才能触发（已验证）

3.2 手势与 触摸事件的关系

- `touch`事件是`UIView`内部的东西，而手势叠加上去的触摸事件
- `subview`会响应`superview`的手势， 但是同级的`subview`不会响应

3.3 系统如何分辨手势种类

- 首先我们想在手势中调用 `touches` 方法必须要导入

  `#import <UIKit/UIGestureRecognizerSubclass.h>`

  因为`gesture`继承的是`NSObject` 而不是 `UIRespond`

- 通过尝试不调用 **tap手势** 的`touchesBegan`，发现tap手势无法响应
- 通过尝试调用`touchesBegan`，但是不调用 **pan手势** 的`touchesMoved`，发现pan手势无法响应
- 我们通过`UITouch`的实例，可以看到里面有很多属性，比如点击的次数，上次的位置等，结合这个属性系统与`touches`方法就可以判断出你使用的是什么手势

3.4 手势与view的touches事件的关系

- 首先通过触摸事件，先响应`touchesBegan`以及`touchesMoved`，直到手势被识别出来，调用`touchesCancelled`，全权交给手势处理。
- 但是我们可以改变这种关系

```
下面是系统的默认设置
 tapGesture.delaysTouchesBegan = NO;
///是否延迟view的touch事件识别；如果延迟了（YES），并且手势也识别到了，touch事件会被抛弃

 tapGesture.cancelsTouchesInView = YES;
///识别手势之后，是否取消view的touch事件
// 如果为NO，touchesCancelled 不会调用，取而代之的是手势结束后touchesEnd
```

4. button事件

4.1 系统是如何分辨`UIControlEvent`

- 我们还是通过`button`内部的`touches`来实践
- 实践过程略，与手势同理
- 比如说`touchUpInside`，通过查看堆栈调用，我们发现在`touchesEnd`后完成对`touchUpInside`的识别，然后再调起`sendAction:`方法

5. 触摸事件的运用

- 上文已经讲过一个button点击范围扩大的案例，再讲一个案例

case2.png

- 与上个例子不同的是，当我们点击`黑色`的时候，因为在`greyview`的外面，别说响应黑色`button`了，我们直接不会响应`greyview`了，怎么办?
- 一种是在`self.view`的`-pointInside`返回 YES, 不过这种在交互复杂的场景不存在实用性
- 我们可以重写`self.view`的`-hitTest`, 把当前触摸的点分别转化为`subviews`上的坐标系的点，在用`subviews`的`pointinside`判断此点，然后返回对应的`subviews`

被以下专题收入，发现更多相似内容

+收入我的专题

推荐阅读

更多精彩内容 >

iOS Masonry学习和探究

前言 开发中对UI进行布局，有很多种，常用的包括frame，Autolayout，storyboard，Mason...


炒河粉儿

iOS深拷贝和浅拷贝

浅拷贝：只创建一个新的指针，指向原指针指向的内存深拷贝：创建一个新的指针，并开辟新的内存空间，内容拷贝自原指针指向...


小宝二代

iOS底层原理探索 — weak实现原理

探索底层原理，积累从点滴做起。大家好，我是Mars。 往期回顾 iOS底层原理探索 — OC对象的本质iOS底层原...


劳模007_Mars

iOS 数组、字典排序总结

1、针对数组简单元素排序数组元素为字符串或基本数据类型时，可直接使用系统定义的函数进行排序 NSString类具有...

小熊_07cb

oc和swift的KVC

小白的简书集合 kvc是什么我就不讲了，网上的资料有很多。 kvc set 大家看下面这张图：大概就是setVal...

小白的天空