IOS消息机制之————Hit Testing 伪代码

写作原因

虽然官方文档上给出了Hit Testing的大略描述,但是发现很多同学在使用中还会有各种疑惑以及错误，而且**任何自然语言的描述都比不上源代码更能让程序员信服**，so~~

先上结果：

```
- (UIView *)hitTest:(CGPoint)point withEvent:(UIEvent *)event
{
    if(self.hidden||self.userInteractionEnabled==NO||self.alpha<0.01)
    {/*此处见文档
      This method ignores view objects that are hidden, that have disabled
user interactions, or have an alpha level less than 0.01.
      */
        return nil;
    }
    if (![self pointInside:point withEvent:event]) {
        return nil;//如果这个点不在本身处理范围内，返回nil
    }
    NSArray* sortedSubViews=[self subviews];/*对subview按照index由大到小排序,i
ndex为视图层级，0为添加的第一个视图*/;
    for (UIView* subview in sortedSubViews) {
        //将父视图的坐标点转换为子视图坐标点
        CGPoint covertPoint=[self convertPoint:point toView:subview];
        //递归子视图
        UIView* hitView=[subview hitTest:covertPoint withEvent:event];
        if (hitView) {
            return hitView;
        }
    }
    return self;
}
```

推导实验：Hit Testing

首先，重写一下UIView的hitTest及pointInside代码

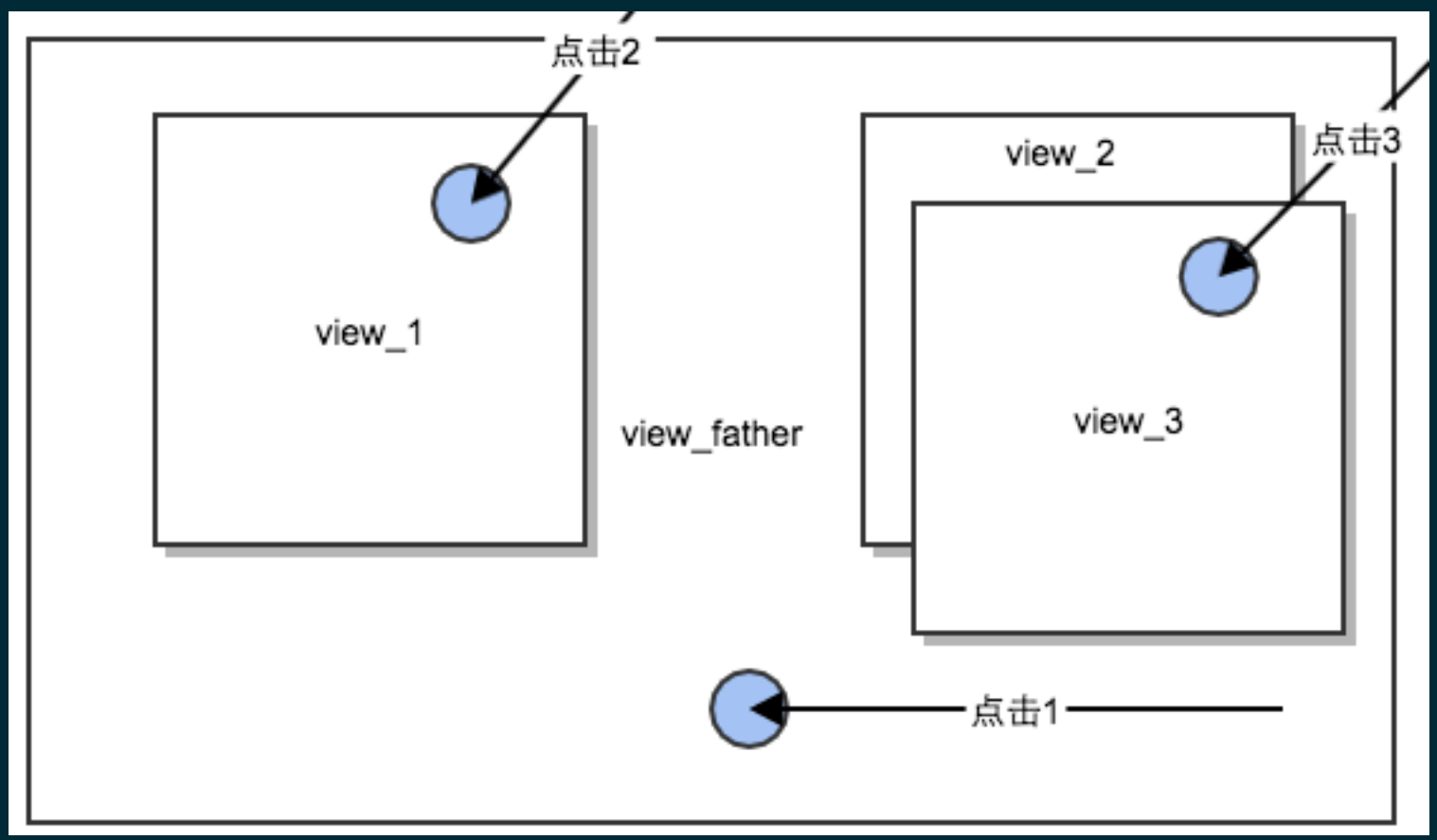
```
-(UIView*)hitTest:(CGPoint)point withEvent:(UIEvent *)event
{
    NSLog(@"name:::%@___%@",self.name,NSStringFromSelector(_cmd));
    UIView* view = [super hitTest:point withEvent:event];
    if (view!=nil) {
        NSLog(@"result view ;;; %@------------------",((VWTansverse*)view)
.name);
    }
    return view;
}

-(BOOL)pointInside:(CGPoint)point withEvent:(UIEvent *)event
```

```
{
    NSLog(@"name:::%@___%@",self.name,NSStringFromSelector(_cmd));
    return [super pointInside:point withEvent:event];
}
```

第二，按照下图布局建立view

先后添加view_1、view_2 、view_3 添加到父视图view_father上，其中view_2,view_3重叠，view_3.userInteractionEnable=NO

第三，按照点击1、2、3 分别点击视图相关位置，查看输出结果



点击位置1输出结果：

> *name:::view_father___hitTest:withEvent:*
> *name:::view_father___pointInside:withEvent:*
> *name:::view_3___hitTest:withEvent:*
> *name:::view_2___hitTest:withEvent:*
> *name:::view_2___pointInside:withEvent:*
> *name:::view_1___hitTest:withEvent:*
> *name:::view_1___pointInside:withEvent:*
> *result view ;;; view_father————————————-*

注：实际上输出结果是上述结果的二倍，因为touchbegin和end都会走这条路线，为了好分析已经去掉重复的一半，下同

点击位置二输出结果：

> *name:::view_father___hitTest:withEvent:*
> *name:::view_father___pointInside:withEvent:*
> *name:::view_3___hitTest:withEvent:*
> *name:::view_2___hitTest:withEvent:*
> *name:::view_2___pointInside:withEvent:*
> *name:::view_1___hitTest:withEvent:*

> *name:::view_1___pointInside:withEvent:*
> *result view ;;; view_1————————————-*
> *result view ;;; view_1————————————-*

点击位置三输出结果：

> *name:::view_father___hitTest:withEvent:*
> *name:::view_father___pointInside:withEvent:*
> *name:::view_3___hitTest:withEvent:*
> *name:::view_2___hitTest:withEvent:*
> *name:::view_2___pointInside:withEvent:*
> *result view ;;; view_2————————————-*
> *result view ;;; view_2————————————-*

分析实验结果
*注意，点击位置2、3最后的resultview 输出两遍是递归的原因

- 通过打印函数的调用顺序可知

  > *HitTest向子视图递归的顺序与添加到父视图的顺序相反*

- view_3立刻返回

  > *用户交互被废除，视图隐藏以及视图的alpha小于0.01的视图会被忽略*

- 点击位置1 返回view_father

  > *如果满足条件的子视图的 hitTest 结果都为空，那么返回本身*

总结：

> *Hit-test得到的View被给予了第一次处理触摸事件的机会，如果它不能处理的话，将由响应链（Responders Chain）来寻找可处理事件的对象。*

对于Hit-test应用、Responders Chain、手势识别类等将会在后续文章中跟进