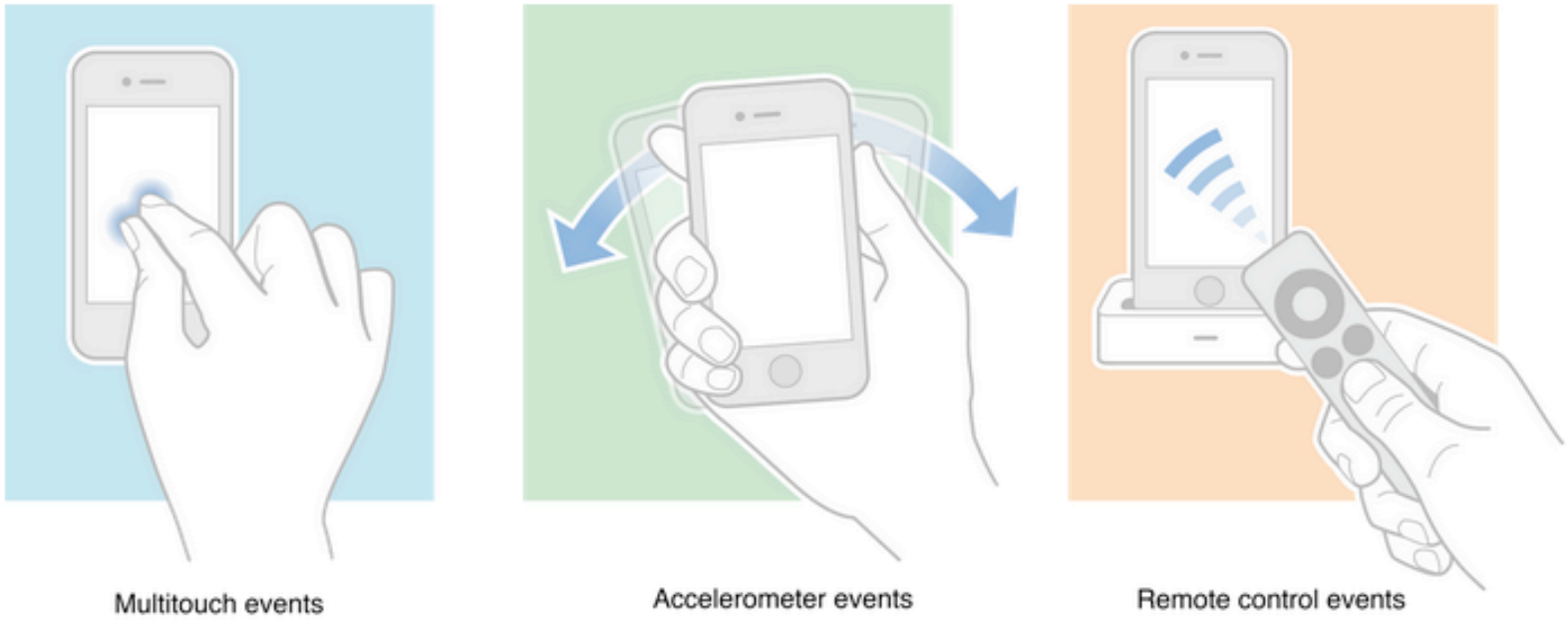
UIKit基础(4)

这节课我会学到什么？

- iOS的事件处理
- 如何捕捉和处理事件
- 如何识别手势

iOS的事件机制

- 在iOS中,事件分为三类:



- 触控事件（单点、多点触控以及各种手势操作）
- 传感器事件（重力、加速度传感器等）
- 远程控制事件（远程遥控iOS设备多媒体播放等）

iOS的事件机制

触控事件就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

响应者对象UIResponder

- 响应者对象是能够响应并且处理事件的对象
- UIResponder是所有响应者对象的父类，包括UIApplication、UIView和UIViewController都是UIResponder的子类。也就意味着所有的View和ViewController都是响应者对象，能够接收并处理事件。

UIResponder响应触摸的方法

UIResponder提供下面四个响应事件的方法，UIView是UIResponder的子类, 所以我们可以通过覆盖下列四个方法去处理view上的触摸事件。当触摸从开始->移动->结束的过程中(有时还会发生触摸取消),系统会自动调用这些方法.

touchesBegan:withEvent: 在手指触摸屏幕时报告UITouchPhaseBegan事件

touchesMoved:withEvent: 在手指在屏幕上移动时报告UITouchPhaseMoved事件

touchesEnded:withEvent: 在手指离开屏幕时报告UITouchPhaseEnded事件

touchesCancelled:withEvent: 在因接听电话或其他因素导致取消触摸时报告UITouchPhaseCancelled事件

第一响应者

第一响应者是第一个接收事件的响应者对象（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。

UITouch类

UITouch对象是一个手指接触到屏幕并在屏幕上移动或离开屏幕时创建的,它可以理解为一个手指一次的触摸。

它有几个属性和实例方法：

phase：属性，返回一个阶段常量，指出触摸开始、继续、结束或被取消，分别对应UITouchPhaseBegan、UITouchPhaseMoved等

tapCount：属性，轻按屏幕的次数

timeStamp：属性，触摸发生的时间

view：属性，触摸始于那个视图

window：属性，触摸始于哪个窗口

lacationInView：方法，触摸在指定视图中的当前位置

previousLocationView：方法，触摸在指定视图中的前一个位置

UIEvent类

- UIEvent对象包含一组相关的UITouch对象

可以理解成一个完整的触摸操作是一个UIEvent，每次手指状态的变化都对应事件动作处理过程中的一个阶段, 通过Began-Moved-Ended这几个阶段的动作(Touch)共同构成了一次事件(UIEvent) 。

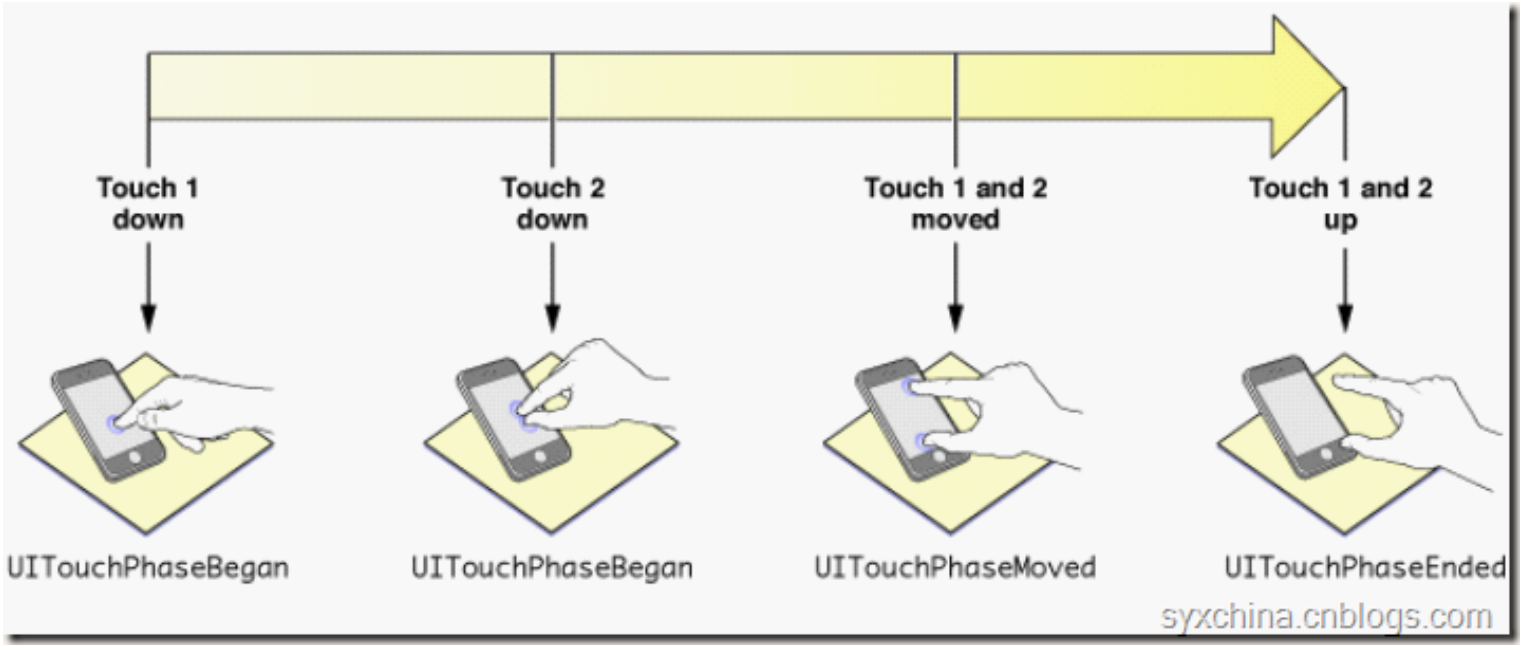
- UIEvent的作用是获取在某个view上的触摸对象(UITouch) 。

UIView默认是不支持多点触摸的，如果想要响应多点触摸，需要把UIView的 multipleTouchEnabled 属性设置为YES。

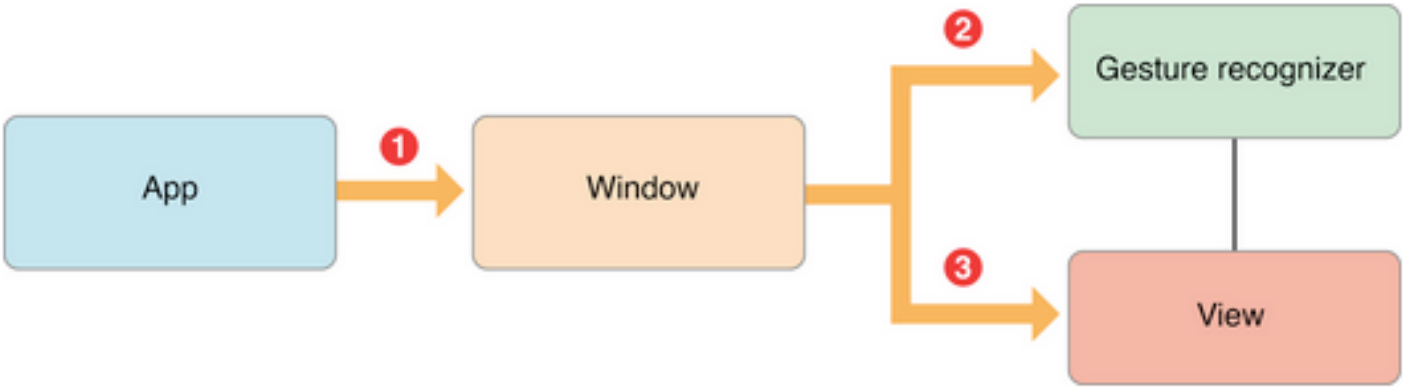
触摸 事件

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）。
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。



事件传递



发生触摸事件后，系统会将该事件加入到一个由UIApplication管理的事件队列中

UIApplication从队列中取出最前面的事件，并将事件分发下去以便处理，通常，先发送事件给应用程序的主窗口（keyWindow）主窗口会在视图结构中找到一个最合适的视图来处理触摸事件

找到合适的视图控件后，就会调用视图控件的touches方法来作具体的事件处理(touchesBegan,touchesMoved...等)

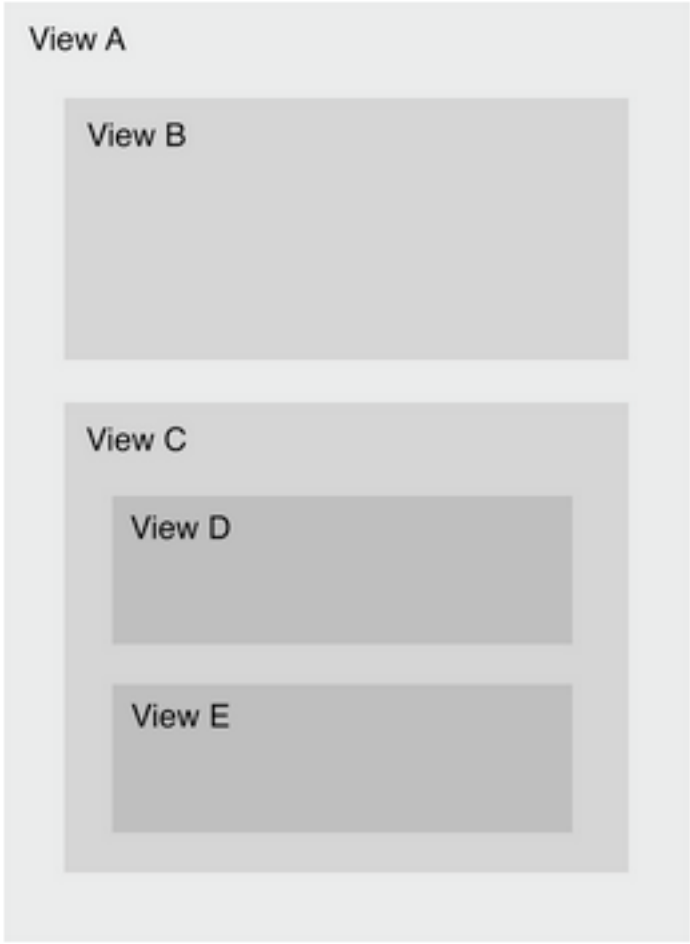
能够处理触摸事件的视图其userInteractionEnabled属性需为yes,另外如果视图透明度为0,或者hidden属性设置为yes,也不响应触摸

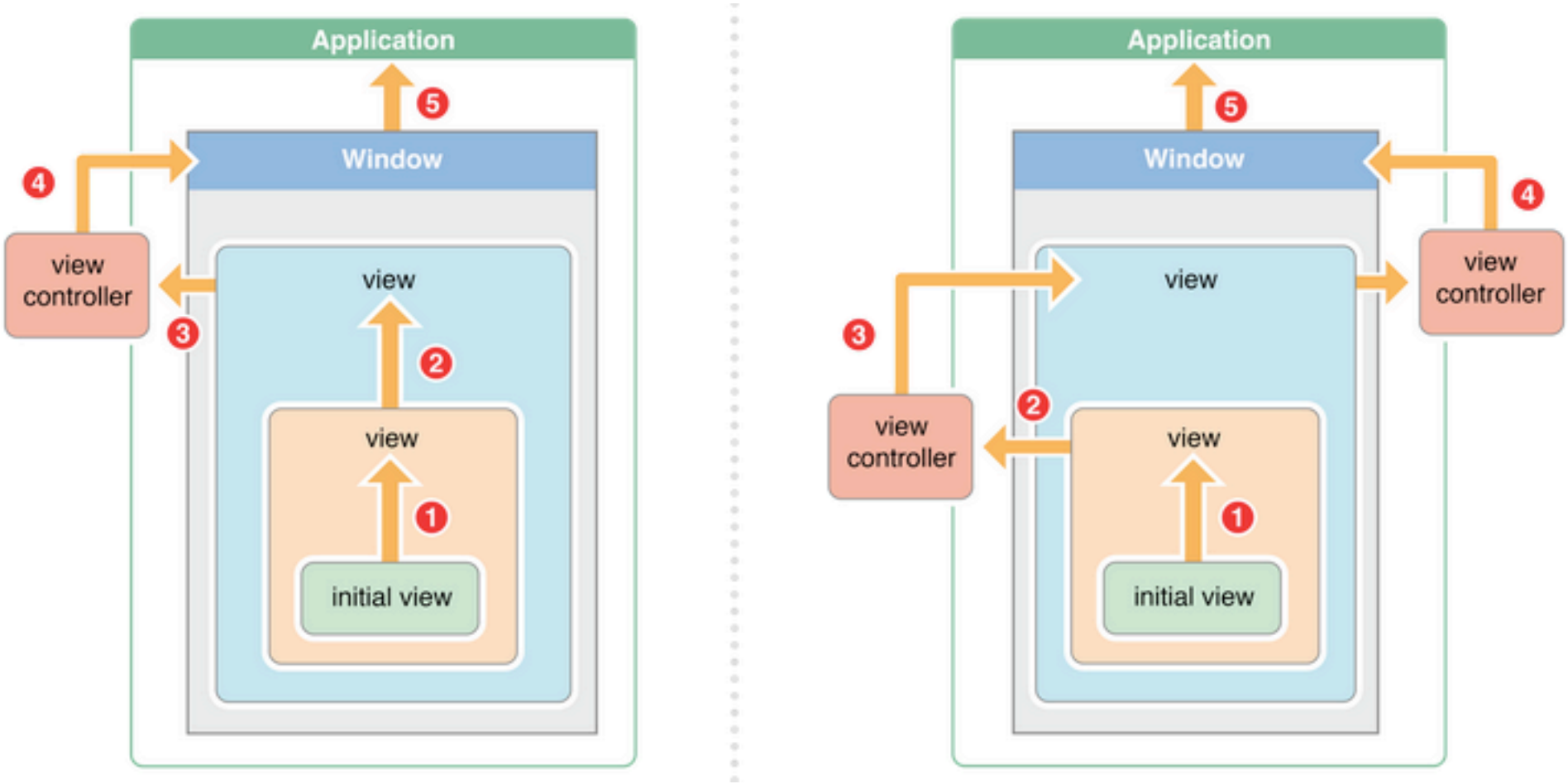
响应者链

通常，一个iOS应用中，在一块屏幕上通常有很多的UI控件，也就是有很多的View，那么当一个事件发生时，如何来确定是哪个View响应了这个事件呢?

寻找hit-test view

什么是hit-test view呢？简单来说就是你触发事件所在的那个View，寻找hit-test view的过程就叫做Hit-Testing。那么，系统是如何来执行Hit-Testing呢，首先假设现在有如下这么一个UI布局，一种有ABCDE五个View。假设一个单击事件发生在了View D里面，系统首先会从最顶层的View A开始寻找，发现事件是在View A或者其子类里面，那么接着从B和C找，发现事件是在C或者其子类里面，那么接着到C里面找，这时发现事件是在D里面，并且D已经没有子类了，那么 hit-test view就是View D啦





如果hit-test view不能处理当前事件，那么事件将会沿着响应者链(Responder Chain)进行传递，直到遇到能处理该事件的响应者(Responsder Object)。

一次完整的触摸事件的传递响应的过程

- UIApplication --> UIWiondw -->递归找到最适合处理事件的控件-->控件调用touches方法-->判断是否实现touches方法-->没有实现默认会将事件传递给上一个响应者-->找到上一个响应者

- UIResponder可以通过在touchesBegan…方法里面调用[super touchesBegan…]方法来使多个响应者同时响应触摸事件

响应者链

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。

可以通过nextResponder方法来获取下一个响应者

手势

如果想获得一个view上面的触摸事件，我们需要实现view的touches方法，在方法内部实现具体处理代码。但是使用touches方法获取view触摸事件，首先要是自定义view，在view内部的touches方法中监听触摸事件，这样外界就不易获取这个触摸事件；而且也不容易区分用户的具体行为（单击双击长按等。。。）

手势帮我们解决了这个问题，UIGestureRecognizer 类给我们提供了一些现成的手势，我们不用自己计算手指移动轨迹，大大简化了开发工作。

手势识别器

UIKit中包含了UIGestureRecognizer类，用于检测发生在设备中的手势。UIGestureRecognizer是一个抽象类，定义了所有手势的基本行为，它有下面一些子类用于处理具体的手势：

1、拍击UITapGestureRecognizer (任意次数的拍击)

2、捏合UIPinchGestureRecognizer (用于缩放)

3、拖拽UIPanGestureRecognizer

4、滑动UISwipeGestureRecognizer

5、旋转UIRotationGestureRecognizer (手指朝相反方向移动)

6、长按UILongPressGestureRecognizer

对于不同类型的手势识别器，具有不同的配置属性。比如UITapGestureRecognizer，可以配置拍击次数。界面接收到手势之后，可以发送一个消息，用于处理响应手势动作后的任务。当然，不同的手势识别器，发送的消息方法也会有所不同

敲击手势


```
// 创建一个手势识别器
UITapGestureRecognizer *oneFingerTwoTaps =
 [[UITapGestureRecognizer alloc] initWithTarget:self
action:@selector(oneFingerTwoTaps:)];

// 需要敲击二次
[oneFingerTwoTaps setNumberOfTapsRequired:2];
//  需要一根手指
[oneFingerTwoTaps setNumberOfTouchesRequired:1];

// 把手势添加到对应的view上
[[self view] addGestureRecognizer:oneFingerTwoTaps];

// 事件触发处理
- (void)oneFingerTwoTaps
{
  NSLog(@"Action: One finger, two taps");
}
```

长按手势


```
UILongPressGestureRecognizer *longPress = [[UILongPressGestureRecognizer
alloc]initWithTarget:self action:@selector(longPress:)];
//  设置需要长按的时间
    longPress.minimumPressDuration = 2;
//  按下后可活动的距离(单位为像素),超过这个距离长按手势就响应失败了
    longPress.allowableMovement = 50;
    [self.imageView addGestureRecognizer:longPress];

- (void)longPress:(UILongPressGestureRecognizer *)ges
{
    NSLog(@"长按");
}
```

滑动手势


```
UISwipeGestureRecognizer *swipeGes = [[UISwipeGestureRecognizer
alloc]initWithTarget:self action:@selector(swipe:)];
//  设置滑动手势触发的方向
swipeGes.direction = UISwipeGestureRecognizerDirectionLeft;
    [self.imageView addGestureRecognizer:swipeGes];

- (void)swipe:(UISwipeGestureRecognizer *)ges
{
    NSLog(@”滑动手势调用的方法");
}
```

旋转手势


```
UIRotationGestureRecognizer *twoFingersRotate =
  [[UIRotationGestureRecognizer alloc] initWithTarget:self
action:@selector(twoFingersRotate:)];
// 让图片跟着手指一起旋转
[self.imageView addGestureRecognizer:twoFingersRotate];

// 旋转过程中方法会不停被调用
- (void)twoFingersRotate:(UIRotationGestureRecognizer *)recognizer
{
  NSLog(@"Rotation in degrees : %f", [recognizer rotation] );
self.imageView.transform = CGAffineTransformRotate(self.imageView.transform,
recognizer.rotation);
// 图片旋转的角度应该是每两次进入此方法的差值,所以这里要归零一下
recognizer.rotation = 0;
}
```

捏合手势


```
UIPinchGestureRecognizer *twoFingerPinch =
  [[UIPinchGestureRecognizer alloc] initWithTarget:self
action:@selector(twoFingerPinch:)];
//  让图片跟随手指的捏合缩放
[self.imageView addGestureRecognizer:twoFingerPinch];

- (void)twoFingerPinch:(UIPinchGestureRecognizer *)recognizer
{
  NSLog(@"Pinch scale: %f", recognizer.scale);
self.imageView.transform = CGAffineTransformScale(self.imageView.transform,
recognizer.scale, recognizer.scale);
//  每次进入此方法都会缩放当前图片的recognizer.scale比例,所以每次把scale
归1
    ges.scale = 1;
}
```

拖拽手势


```
    UIPanGestureRecognizer *panGes = [[UIPanGestureRecognizer
alloc]initWithTarget:self action:@selector(pan:)];
//  让图片随着手指拖动
    [self.imageView addGestureRecognizer:panGes];

- (void)pan:(UIPanGestureRecognizer*)ges
{
    CGPoint trans = [ges translationInView:self.imageView];
    self.imageView.center = CGPointMake(self.imageView.center.x + trans.x,
                                        self.imageView.center.y + trans.y);
//  移动的距离清零
    [ges setTranslation:CGPointZero inView:self.imageView];

}
```

手势的代理方法


```
// 若返回no,则此次手势直接进入识别失败状态(默认为YES)
- (BOOL)gestureRecognizerShouldBegin:(UIGestureRecognizer *)gestureRecognizer

// 默认情况下一个view不能同时响应多个手势,在此方法返回yes,则可以同时响应
(BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
shouldRecognizeSimultaneouslyWithGestureRecognizer:(UIGestureRecognizer *)otherGestureRecognizer

//此方法在window对象在有触摸事件发生时，调用gesture recognizer的touchesBegan:withEvent:方法之前调用，如果返回NO,则gesture recognizer不会看到此触摸事件。(默认为YES).
- (BOOL)gestureRecognizer:(UIGestureRecognizer *)gestureRecognizer
shouldReceiveTouch:(UITouch *)touch
```

手势使用注意

1.一个手势只能添加到一个View上,但一个View可以有多个手势

2.当一个view添加多个手势时，在缺省情况下，没有为优先执行哪个手势做排序，每次发生的次序可能不同

3.有些手势是有关联的,比如单击和双击,如果一个视图上既添加了单击手势,又添加了双击手势,那么在进行双击的过程中,会在第一次点击的时候先被识别为单击事件,然后又触发双击事件,这种情况下,我们允许让单击手势在另外一个双击手势可以开始分析触摸事件之前先失败.

```
[singleTap requireGestureRecognizerToFail:doubleTap];
```

这个方法使singleTap在doubleTap识别失败的时候才分析事件，如果doubleTap识别出双击事件，则singleTap不会有任何动作.

在这种情况下,如果用户进行单击操作，需要一段延时(即doubleTap识别失败)，singleTap才会识别出单击动作，进行单击处理，这段时间很短，对实际使用几乎没有影响。

实例：点击非输入框隐藏键盘


```
- (BOOL)textFieldShouldBeginEditing:(UITextField *)textField
{
    UITapGestureRecognizer *tapGestureRecognizer = [[UITapGestureRecognizer
alloc] initWithTarget:self action:@selector(done:)];
//  将手势添加到控制器视图上
    [self.view addGestureRecognizer: tapGestureRecognizer];
    return YES;
}
-(void)done:(id)sender
{
    [self.view endEditing:YES]
}
```

手势在使用完成或者离开页面时候要移除