**事件：event**

**委托和事件的理解和区分：**

1. 事件的里面其实就是封装了一个多播委托
2. 委托和事件的作用是一样的，也是让委托、事件指向某个函数，然后该委托、事件则执行这个函数，但还是有很大区别的
3. 安全性能与调用条件：

委托 --> 能够指向一个函数的类型，以面向对象编程设计的思想来讲，它是不安全的：因为委托是能够在命名空间下声明，并且在任何地方都能够被调用到的，在客户端可以对它进行随意的赋值等操作，严重破坏对象的封装性

事件 --> 一个类型安全的委托：只能在特定条件才能够触发事件，并且只能声明在类中，并且只能在声明事件所在的类中才能调用，其本质就是一个委托对象

1. 两者间的性质：

一个类中，存在着字段，字段对应的便是属性，而属性的作用在于保护字段，不给随意调用或者赋值。委托和事件就类似如此：我们在声明一个事件的时候，程序在系统内部执行到声明事件的这行代码的时候，其实在其内部又声明了一个私有的委托，这时候事件就封装了这个私有的委托，因为不能随意调用和赋值的限制，事件也间接的保护了这个在内部自动产生的私有的委托，就好比类中属性保护字段的性质

1. 委托和事件的赋值语法：

-> 委托

委托：当我们想让委托指向某一个函数的时候我们可以通过：

委托名 **=** new 委托名(函数名) **或** 委托名 = 函数名

-> 事件

事件：当我们想让事件指向某一个函数的时候我们需要通过：

事件名 +=/-= new 委托名(事件处理函数) **或** 事件名 +=/-= 事件处理函数

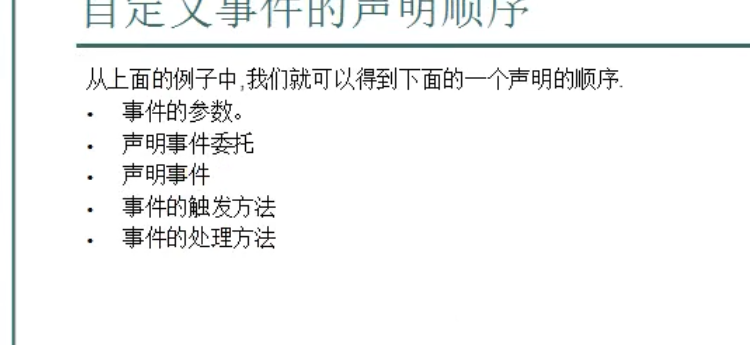
**-> 共同**

相同类型的委托/事件的实例 可以直接赋值给另一个 相同类型的委托/事件的实例

当赋值成功后，被赋值的 委托/事件的实例 享有赋值的 委托/事件的实例 中所指向的函数

(引用类型，传递地址)

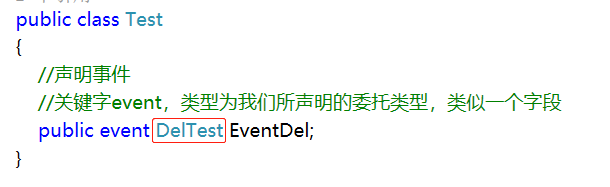
**事件的使用步骤：**

****

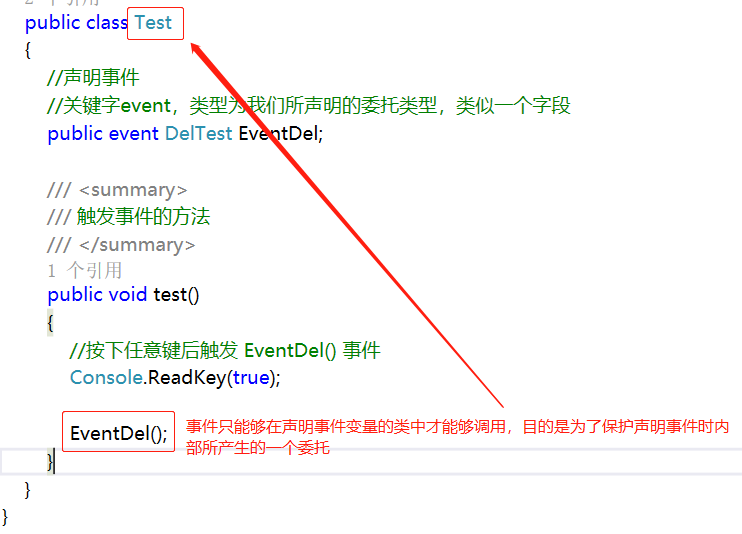
第一步：我们需要先谋定一个事件被触发后所执行的函数需要的参数和返回值，并且声明一个与该函数具有相同签名的委托



第二步：我们新建一个类，这个类其实就是事件发布者类，并且在该类中通过event关键字，写一个我们所声明委托的类型的一个变量，这样我们就成功声明了一个事件 --> EventDel



第三步：我们写一个触发事件的一个函数(条件)，因为事件要当满足特定条件才能够被触发，如下图：我们设定了EventDel()的触发方式为按下任意键，但是这时候的EventDel其实是没有东西的，因为我们只写了触发这个事件的方法，并没有写触发事件后的处理方法并让事件EventDel去注册(指向)这个方法



第四步：我们在主函数中创建事件发布者类的对象，并为我们所声明的事件的字段注册(订阅)，注册(订阅)一个触发事件后事件EventDel需要执行的处理方法，然后，我们通过该类中的触发事件的函数，来触发这个事件，并让事件执行一个处理事件的方法

