# **事件驱动**

## **事件的概念和wx中事件驱动**

**鼠标，键盘操作，都属于事件，用一段代码来对事件的发生作出响应，这个过程就是从事件被映射到代码的过程，叫做事件处理过程；**

## **一些术语**

1. **事件对象：在wx中，它具体代表一个事件，比如移动鼠标，它包含事件数据等属性，它是类wx.Event或者其子类的实例，子类比如wx.MouseEvent**

**Wx.event类中存放的应该是能够处理大多数事件的类，基本的鼠标，键盘事件；而且，既然是它的子类，那就一定可以重写，支持自定义事件。所以，自定义事件，在事件对象的定义上，它一定是wx.event的子类**

1. **事件类型(event type)：wx分配给每个事件一个整数ID，事件类型给出了关于该事件本身更多的信息，比如，wx.MouseEvent事件类型标识了该事件是一个鼠标单击还是移动；它是event类的一个属性，表明事件属于何种类型。比如，左键单击和双击是不同的类型，但都属于mouseevent这个类；**
2. **事件源：任何wx对象都可以产生事件，这种产生事件的对象叫事件源，比如按钮，菜单，列表框等。**
3. **事件驱动：一个程序结构，它的大部分时间花在等待或者响应事件上。**
4. **事件队列：表示已发生的，但是未处理的事件的一个列表**
5. **事件处理器：响应事件时所调用的函数或方法。也称为处理器函数或者处理器方法**
6. **事件绑定器：一个包含特定的窗口部件，特定事件类型，一个事件处理器的wx对象。为了能被使用，所有的事件处理器必须用一个事件绑定器来注册。**

**它相当于是一个汇合点，有一个绑定器对象，作为bind函数的第一个参数，它把事件处理器，特定的事件实例汇聚在一起，让他们之间进行一系列的化学反应；**

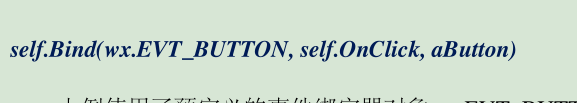
1. **Wx.EvtHandler：一个wx类，它允许它的实例在一个特定类型，一个事件源和一个事件处理器之间创建绑定。注意，这个类与之前定义的事件处理函数或方法不是同一个东西。**

**它其实就是绑定器对象的父类，也就是所有可显示对象的父类，可显示对象比如菜单，按钮，状态栏等。正因为wx.eventhandler中有bind方法和事件绑定器对象，可显示对象都是wx.eventhandler的子类，那么他们就都可以使用bind方法和事件绑定器对象；**

## **事件处理器和事件绑定器**

**事件机制的组成部分是事件绑定器和事件处理器。绑定器是一个预定义的wx对象，每个事件都会有自己的绑定器，绑定事件实例。事件处理器是一个函数或者方法，它要求一个wx事件实例作为参数，当用户触发了适当的事件时，一个处理器就会被调用。**

**用一个例子来说明这个问题**

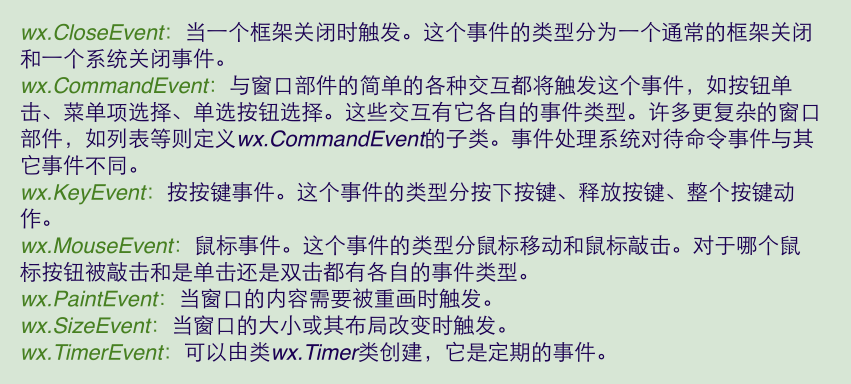
****

**Wx.EVT\_BUTTON是一个预定义的事件绑定器对象，self.OnClick是一个事件处理器，aButton是一个事件实例。Bind方法是wx.EvtHandler类的一个方法，这个方法提供了一个平台，让EVT\_BUTTON绑定器方法把OnClick方法和aButton绑定在一起，wx.EvtHandler是所有可显示对象的父类，所以bind方法可以被放置在任何显示类**

**事件驱动体系通常是分散性的，响应一个窗口事件的代码通常不是定义在该部件中，比如一个按钮单击事件的代码不必在该按钮部件中定义，它可以在按钮所在的框架中，也可以在其他部分。这样做最大的好处是在面向对象设计中，事件处理器的代码可以重用。**

**在wx中，代表事件的是事件对象，事件对象是wx.Event或其子类的一个实例。父类wx.Event相对小且抽象，它只是包含了一些对所有事件通用的信息。**

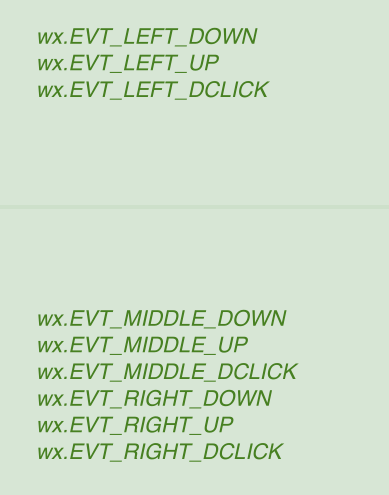
**在wx中，有一些wx.Event的子类，强调：一个事件类可以有多个事件类型，每个事件类型都对应不同的用户行为。比如wx.KeyEvent，按键事件，它是一个大的类，表示按键事件，但事件类型有多个，包含按下按键，释放按键等，每个事件类型都对应用户不同的操作，而这些不同的操作会创建新的子类，继承wx.KeyEvent这个大类。**

****

## **事件绑定器**

**事件绑定器由类wx.PyEventBinder的实例组成，一个预定义的wx.PyEventBinder的实例被提供给所有支持的事件类型。每一个事件类型都会有一个事件绑定器，他们是一一对应的。**

**事件绑定器实例的名字是全局性的，都以wx.EVT\_开头，比如，wx.MouseEvent的事件类型，有14个，对应了鼠标的单击，释放，双击事件等，其中九个涉及到按钮的敲击，他们都对应一个事件绑定器，如下图：**

****

**还有一些是其他的事件类型：**

**Wx.EVT\_MOTION： 产生于用户移动鼠标**

**Wx.ENTER\_WINDOW和wx.LEAVE\_WINDOW：产生于当鼠标进入或离开一个窗口**

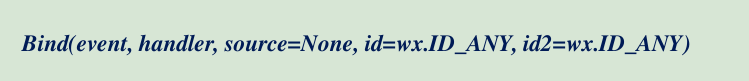
**Wx.EVT\_MOUSEWHEEL产生于鼠标滚轮活动**

**Wx.EVT\_MOUSE\_EVENTS一次绑定所有的鼠标事件到一个函数**

**同样的，wx.CommandEvent有28个不同的事件类型与之关联。他们中的大多数是针对单一窗口部件的，之后会讨论这个类。**

**绑定事件的好处在于可以是wx细化分派事件，并且允许同类的其他类似事件发生并共用数据和功能。**

**使用wx.EvtHandler类的方法，一般情况下，很多方法都用不到，经常使用到的是Bind()方法，它创建事件绑定，用法如下：**

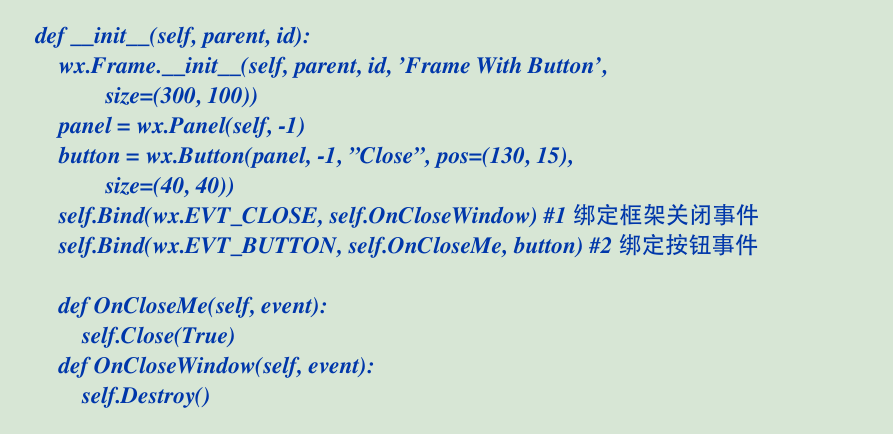
****

**Bind方法用来将一个事件绑定器和一个事件处理器函数关联起来，其中事件绑定器包括两个方面：一是发生的事件对象，二是对这个事件的响应对象。**

**Event参数是必须的，它是绑定器的一个实例；**

**Handler也是必须的，它是一个可调用的python对象，通常是一个函数或者方法。也可以是None，这种情况下，事件没有关联的处理器。**

**Source它用来描述在触发事件的窗口部件与用作事件处理器的窗口部件不相同时使用。它一般是none，因为一般都是在一个定制的框架内进行操作，而且绑定的事件也是在这个框架内。如果父窗口包含了多个按钮敲击事件源，那么就要指定source参数以便wx能区分他们。下面的例子显示了这个方法的一种用法**

****

**1 绑定框架的事件是用在框架上，所以不需要指定source**

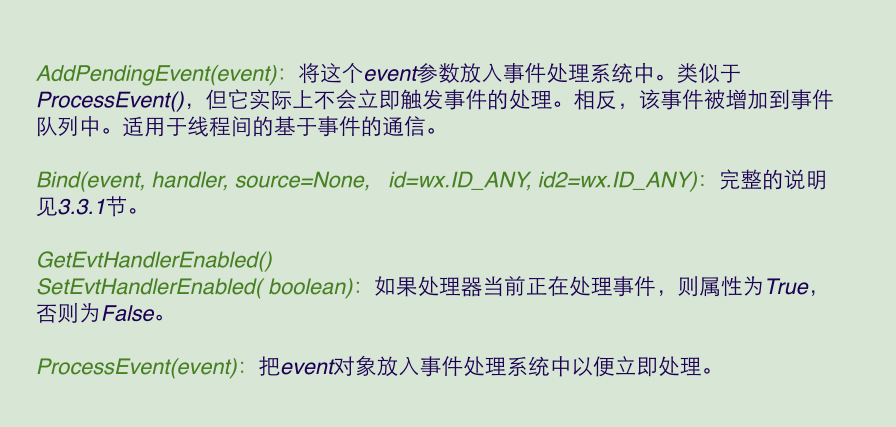
**2 绑定到按钮的事件是使用了OnCloseWindow方法，source指定为button，为了让wx区分事件是由哪一个窗口部件产生的。这里的source也可以不指定，因为触发事件的窗口部件和事件处理器的窗口部件都是一个窗口**

**03.py中的第一个程序显示了如何工作的。**

**03.py的第二个程序显示了使用source指定触发的窗口部件，在这里是框架本身；**

**Bind方法中的id和id2用来指定窗口的事件源，这通常情况下是没必要的，因为可以从source中读取。但有两种情况是需要的：一是使用对话框，用id比source更容易；二是当两个id同时使用，那就会绑定在这两个id之间的窗口事件，这种情况仅仅适用于id和id2是连续的。**

**Wx.EvtHandler除了有bind方法之外，还有一些其他的方法**



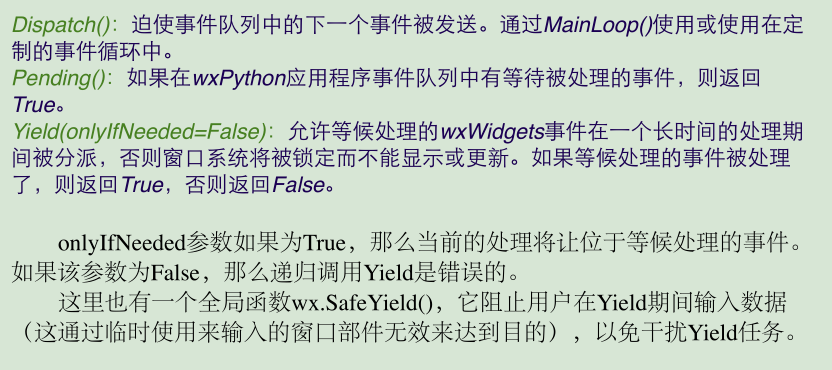
## 事件处理过程

**03.py的第三个程序展示了事件处理的过程**

**03.py的第四个程序显示了如何使用skip方法来让一个操作产生多个事件。**

**Note：当要捕捉低级事件，比如鼠标按下或释放，就需要使用skip方法，来进一步捕捉到其他事件，否则进一步事件将会被阻止。**

**有时候，需要程序继续执行，而不是等待，那么需要改变主循环，下面是一些改变主循环的方法，这些方法不常用，但是有时候很重要。**



## 定制事件 （留到后面看）