Reactive Extensions (Rx) 入门(4) —— Rx的事件编程②

blog.csdn.net/fangxing80/article/details/7685393

原文：http://www.atmarkit.co.jp/fdotnet/introrx/introrx_01/introrx_02_02.html

作者：河合 宜文

事件是什么？用Rx来处理事件的优势
让我们来看看如何用Rx特有的事件Obserable转换事件处理的。在这之前，让我们考虑一下事件在.NET中的应用场景。
最具有代表性的应该是GUI的事件了。比如：按钮点击，鼠标移动等，这些全是通过事件来处理的。而WindowsPhone(简记为WP7)中增加的比如触摸等手势输入都是事件处理，TabletPC和WP7一样，也配备了传感器，还有最近出尽风头的Kinect设备，也配置了传感器。

另外，还有以通知(推送)为目的，例如INotifyPropertyChanged接口就是利用事件来通知属性发生变化了。还有FileSystemWatcher类是对文件或者文件夹进行变化监视通过事件进行通知。其他还有Timer的事件，WebClient的异步方法等也是利用了事件来处理。

GUI事件：合成
那么用Rx来操作事件有什么爽的呢？如果要说的话，首先是可以“合成”。比如：GUI事件中的鼠标按下/移动/放开 想要对这组操作进行组合的场景，一般的作法都是外部做一个Flag来管理，这样可能会使得代码变得复杂且可读性下降，而且除了这种组合之外，再想在按下/移动/放开事件中处理一些别的，则非常容易使原有代码结构发生破坏。

但如果使用Rx来操作那么则无需使用外部的Flag，直接可以将按下/移动/放开合成为一个新的事件。这样的话，即不用Flag来控制也可以在按下/移动/放开里各自处理事件。代码也不会混在一起，更加简洁。另外，合成的事件可读性也被大大提高了。

下面的代码示例如何将 按下/移动/放开 合成一个新的事件并进行处理：

```
1. // WindowsForm的Drag事件：鼠标左键按下/移动/直到放开过程中
2. // 取得鼠标的坐标
3. var drag = from down in this.MouseDownAsObservable()
4.            from move in this.MouseMoveAsObservable().TakeUntil(
5.                                        this.MouseUpAsObservable())
6.            select move.Location;
```

* MouseDownAsObservable、MouseMoveAsObservable、MouseUpAsObservable方法是用Observable类的FromEvent静态方法包装的扩张方法，稍后说明。
另外，对于简单的事件处理，可以只利用IObservable<T>对象的OnNext方法。用Rx来处

理事件，事件完成(OnCompleted方法)，异常处理(OnError方法)是完整的，这比一般事件的表现力更好。

Timer/通知事件
再考虑下Timer。例如轮询，在一定的时间间隔监视某个值的场景。Rx将Timer变为一个序列，另外通过Select方法可以灵活变换输出。这些组合在一起，监视对象的值在一定时间间隔内会自动推送过来，非常容易操作。而且过滤值的处理也很简单，可以只在值发生变化时再接收这样就更简单了。
下面是Rx处理Timer在一定时间里生成一个值并且进行过滤的示例代码。

```
// 每隔1秒监视一下watchTarget.Value的值
var polling =
  Observable.Timer(TimeSpan.Zero, TimeSpan.FromSeconds(1))
  .Select(_ => watchTarget.Value)
  .DistinctUntilChanged(); // 只有在值发生变化时才引发事件(polling)
```

因为使用Rx进行过滤很容易，所以对于手势以及传感器中大量的通知事件的处理上，可以很简单的进行取舍和调整。另外，Rx使得过滤处理不仅仅是去掉了if...else语句，还可以通过时间的过滤。下面是根据时间来过滤的示例代码：

```
// FileSystemWatcher的Changed事件
// 发生一次变化，会触发多个事件
var watcher =
  new FileSystemWatcher("C:\\", "test.txt")
    { EnableRaisingEvents = true };
// "对于1秒内连续发生的事件，进行过滤，只处理最后一个"
// 变成一个相对更容易处理的对象
var changed =
  Observable.FromEventPattern<FileSystemEventArgs>(
                                          watcher, "Changed")
  .Throttle(TimeSpan.FromSeconds(1)); // Throttle方法是只允许通过指定时间和指定值的内容（顾名思义：阀门）
```

UnitTest
Rx的特点是可以自由的处理事件，利用Rx可以解决本来很困难的事件的UnitTest。例如：3

分30秒时生成一个数值"10"，之后的4分0秒时生成一个数值"20"，Rx能模拟这样的时间和值组合的事件。这些测试用的方法都在Microsoft.Reactive.Testing 程序集里(用NuGet "Rx-Testing")。关于Rx的UnitTest，将在后面再详细介绍。

FromEvent方法和FromEventPattern方法
用Rx处理事件，需要用 FromEvent 方法或者 FromEventPattern 方法将事件变为 IObservable<T> 对象。FromEvent 方法可以转换 Action<T> 代理，序列元素则为 T 。FromEventPattern 方法可以转换 EventHandler 代理，序列元素则为 EventPattern<TEventArgs>，它包装了 Object 类型的 sender 和 TEventArgs 类型的 e。

事件的Rx变换：指定事件名
先来看看 FromEventPattern 方法。

```
1. // WPF / Silverlight / WP7 按钮控件（「button1」）的Click事件 Rx 化
2. Observable.FromEventPattern<RoutedEventArgs>(button1, "Click");
```

这是事件的Rx变换最简单的方法，FromEventPattern 方法的类型参数是 EventArgs 类型。（这个例子里是 RoutedEventArgs），第一个参数将控件实例传入，第二个参数则指定事件的名称（这个例子是 Click）。正如你看到的，由于事件名称是字符串，里面的处理要通过反射来做。和非反射的方法（后面会介绍）来比，这样的记述是简单好记的，而且这种变换并不频繁（只要一次），对性能的影响几何可以忽略不计。因此这么用也没什么问题。

事件的Rx变换：指定事件处理方法（非反射方法）
如果不使用反射进行Rx变换的话，可以按照下面的作法：

```
1. // 第一个参数是事件响应处理的代理（这里 h => h.Invoke 是固定）
2. // 第二个参数绑定事件，第三个参数解除事件
3. Observable.FromEventPattern<RoutedEventHandler, RoutedEventArgs>(
4.   h => h.Invoke,
5.   h => button1.Click += h, h => button1.Click -= h);
```

代码量稍微增加了，但对于性能上的考虑这是最好的作法。此外 FromEventPattern 方法还有几个重载。

如何清理事件的Rx变换
指定事件的处理，还有指定的字符串，这样的编码会比较多，实际运用中这样会影响代码的维护性。因此推荐将这些代码作为扩展方法分离出去。

```
1. public static class ButtonBaseExtensions
2. {
3.   // 分离的扩展方法
4.   public static IObservable<RoutedEventArgs> ClickAsObservable(this ButtonBase
   button)
5.   {
6.     return Observable.FromEventPattern<RoutedEventHandler, RoutedEventArgs>(
7.         h => h.Invoke,
8.         h => button.Click += h, h => button.Click -= h)
9.       .Select(x => x.EventArgs);
10.   }
11. }
12. // 实际运用时调用
13. button1.ClickAsObservable().Subscribe(_ => MessageBox.Show("Clicked!"));
```

从上面的例子中，从EventPattern<TEventArgs>对象中进一步 Select 出EventArgs对象。在示例的注册事件的处理方法中，Object的sender 就是扩展方法对象自己(button1)也不需要传了，只把 EventArgs类型的 e 传过去，反而会更简单。

(Data Developer Center版) FromEvent 方法也和 FromEventPattern 方法类似，对象事件的类型是 Action<T> 。虽然 .NET Framework 中，Action<T> 虽然不是标准的Event对应的代理类型，直接使用的场景不是太多，但是对于开发者自定义的Action<T>的事件时可以使用。
另外，通过对代理的变换，就像上面的示例一样，也可以省略sender参数。

```
1. public static class ButtonBaseExtensions
2. {
3.   // 使用FromEvent生成EventPattern对象
4.   // 用Select省略sender
5.   public static IObservable<RoutedEventArgs> ClickAsObservable(this ButtonBase
   button)
6.   {
7.     return Observable.FromEvent<RoutedEventHandler, RoutedEventArgs>(
8.       h => (sender, e) => h(e),
9.       h => button.Click += h, h => button.Click -= h);
10.   }
11. }
```

如果sender参数是必要的时候，也可以通过 Select 重新包装来做。比如：
button1.ClickAsObservable().Select(ev => new { Sender = button1, EventArgs = ev })

接下来会进一步介绍 Rx 如何进行合成的。