Reactive Extensions (Rx) 入门(5) —— Rx的事件编程③

原文：http://www.atmarkit.co.jp/fdotnet/introrx/introrx_02/introrx_02_03.html

作者：河合 宜文

合成用的方法

本章将介绍一些Rx代表性的方法。

○ **SelectMany 方法**

SelectMany 方法是 Rx 中最常用的方法之一。例如将鼠标移动事件插入鼠标按下事件中，甚至对于序列自身的修改替换。另外，从第一个异步结果中启动第2个异步处理，这对于使用Rx进行异步编程是非常重要的。

SelectMany的处理图

根据 A 序列的值，后续用 B 序列的值进行插入替换。

```
// 替换别的Observable的内容
// 结果：10, 10, 11, 10, 11, 12
Observable.Range(1, 3)
   .SelectMany(x => Observable.Range(10, x))
   .Subscribe(Console.WriteLine);
// 实际的替换过程
// { x = 1, y = 10 }
// { x = 2, y = 10 }
// { x = 2, y = 11 }
// { x = 3, y = 10 }
// { x = 3, y = 11 }
// { x = 3, y = 12 }
var query = from x in Observable.Range(1, 3)
            from y in Observable.Range(10, x)
            select new { x, y };
query.Subscribe(Console.WriteLine);
```

Range(1, 3)（A）序列的第一个值(x=1)代入后面的 Range(10, x) 中（B），因此被“10”替换，A 的第二个值(x=2)在 B 中"10", "11”代替，A的第三个值(x=3)则是被 "10","11","12"代替。

○ **Concat方法**

Concat 是将2个序列进行连接的方法。这个时候，直到第一个序列终止前，第二个序列的值就会被忽略掉。我们可以理解是在第一个序列的结尾追加上另一个序列。

代码如下例：

```
1. // 运行结果：1, 2, 3, -1, -1, -1

2. Observable.Range(1, 3)

3.    .Concat(Observable.Repeat(-1, 3))

4.    .Subscribe(Console.WriteLine);
```

不仅仅是2个序列结合，复数(IEnumerable<IObservable<T>>)的连接也是可能的，在想明确规定执行顺序的场景里可以适用。

○ **Merge 方法**

Merge会将所有的值都会合并进来。不只是2个对象的连接，也可以进行多个对象的连接。如果要对应多个控件的共通处理的话，使用Merge是很方便的。

```
// WindowsForm中的4个TextBox控件全部设定为：
// “DragDropEffects.All”
new[] { textBox1, textBox2, textBox3, textBox4 }
   .Select(x => Observable.FromEventPattern<DragEventArgs>(x, "DragEnter"))
   .Merge()
   .Subscribe(x => x.EventArgs.Effect = DragDropEffects.All);
// 上面的Merge方法是下面的代码的变形，
// 修改为：IEnumerable<IObservable<T>>进行Merge
// 代码变得更简洁
Observable.Merge(
   Observable.FromEventPattern<DragEventArgs>(textBox1, "DragEnter"),
   Observable.FromEventPattern<DragEventArgs>(textBox2, "DragEnter"),
   Observable.FromEventPattern<DragEventArgs>(textBox3, "DragEnter"),
   Observable.FromEventPattern<DragEventArgs>(textBox4, "DragEnter")
);
```

乍一看 IEnumerable<IObservable<T>> 对象的Merge好像有点奇怪，但结合 Linq2Object 使用的 IEnumerable<IObservable<T>> 的确能节省代码。

○ **Zip 方法**

Zip方法是A和B中各取1个值为一组（2个值）进行配对处理。一边的值如果发生偏移，那么Zip会直到取到2个值为止才输出。如下图所示：

如下代码所示，使用Zip方法将 Interval 方法（指定时间间隔发行值）和Timestamp（实际时刻）进行组合的结果。

```
// 結果：
// { x = 0@2011/12/20 7:37:15 +09:00, y = 0@2011/12/20 7:37:17 +09:00, now =
2011/12/20 7:37:17 +09:00 }
// { x = 1@2011/12/20 7:37:16 +09:00, y = 1@2011/12/20 7:37:20 +09:00, now =
2011/12/20 7:37:20 +09:00 }
// { x = 2@2011/12/20 7:37:17 +09:00, y = 2@2011/12/20 7:37:23 +09:00, now =
2011/12/20 7:37:23 +09:00 }
// { x = 3@2011/12/20 7:37:18 +09:00, y = 3@2011/12/20 7:37:26 +09:00, now =
2011/12/20 7:37:26 +09:00 }
// { x = 4@2011/12/20 7:37:19 +09:00, y = 4@2011/12/20 7:37:29 +09:00, now =
2011/12/20 7:37:29 +09:00 }
// { x = 5@2011/12/20 7:37:20 +09:00, y = 5@2011/12/20 7:37:32 +09:00, now =
2011/12/20 7:37:32 +09:00 }
// { x = 6@2011/12/20 7:37:21 +09:00, y = 6@2011/12/20 7:37:35 +09:00, now =
2011/12/20 7:37:35 +09:00 }
Observable.Interval(TimeSpan.FromSeconds(1))
  .Timestamp()
  .Zip(Observable.Interval(TimeSpan.FromSeconds(3)).Timestamp(), (x, y) => new { x,
y, now = DateTimeOffset.Now })
  .Subscribe(Console.WriteLine);
```

1秒间隔的timestamp序列和3秒间隔的timestamp序列，进行组合的结果。

○ **CombineLatest 方法**
类似 Zip 方法，两边引发“值”时，取得最新的值输出。如下图所示，两边都引发事件，且需要对两边的事件都需要处理的场景：

A和B的序列，任何一边在引发变化时都会取出两边最新的值输出。

如下，2个Checkbox，进行Check变换时，每次都会将两个Checkbox的Checked状态输出。

```
1. public static class ToggleButtonExtensions
2. {
3.   // WPF / Silverlight / WP7の ToggleButton控件（Checkbox）
4.   // 如果Check状态变化 IsChecked属性值也跟着变化
5.   public static IObservable<bool> IsCheckedAsObservable(this ToggleButton button)
6.   {
7.     var checkedAsObservable = Observable.FromEvent<RoutedEventHandler,
   RoutedEventArgs>(
8.       h => (sender, e) => h(e),
9.       h => button.Checked += h, h => button.Checked -= h);
10.    var uncheckedAsObservable = Observable.FromEvent<RoutedEventHandler,
   RoutedEventArgs>(
11.      h => (sender, e) => h(e),
12.      h => button.Unchecked += h, h => button.Unchecked -= h);
13.    return Observable.Merge(checkedAsObservable, uncheckedAsObservable).Select(_ =>
   button.IsChecked.Value);
14.  }
15. }
16. // checkBox1和checkBox2两个CheckBox
17. // 同时选中时，MessageBox才表示。
18. checkBox1.IsCheckedAsObservable()
19.   .CombineLatest(checkBox2.IsCheckedAsObservable(),
20.     (isChecked1, isChecked2) => new { isChecked1, isChecked2 })
21.   .Where(x => x.isChecked1 && x.isChecked2)
22.   .Subscribe(_ => MessageBox.Show("同时选择！"));
```

“checkBox1.IsCheckedAsObservable()"和"checkBox2.IsCheckedAsObservable()"的序列，双方任何一个状态发生改变时，都会取得两个最新的选择状态值输出。

但是，需要注意的是，在XAML中需要绑定CheckBox的IsChecked属性。XAML中，数据绑定是非常常用的。GUI编程中Rx的利用范围不能说是非常广。为了在XAML中更好的使用 Rx，有几个第三方的库可以利用。一个叫“ReactiveUI”，可以再MVVM模式开发中，对

于ViewModel开发提供强有力的支持。另一个是笔者开发的“ReactiveProperty”，属性自身可以绑定而且是 IObservable<T> ，通过声明式的 CanExecute 利用 ReactiveCommand 可以与 View 分离。

○ **Scan方法**

最后，这个Scan方法不是连接方法而是一个集计的方法。Scan方法是1个前面的“结果”和现在的“值”进行合成输出的。因为可以获得1个前面的结果值，所以进行差分(累计)计算时使用比较方便。如下图所示：

A序列中，1个前面的“结果”(中间褐色的横线)和当前的“值”(上面蓝色的横线）进行合成。Scan 方法就像 Linq2Object 中的 Aggregate 方法在计算时，列举的全部中间结果。

```
1. // 1, 3（＝1＋2）, 6（＝3＋3）, 10（＝6＋4）, 15（＝10＋5）
2. Observable.Range(1, 5)
3.   .Scan((x, y) => x + y)
4.   .Subscribe(Console.WriteLine);
```

Range(1, 5) 的序列中，第一次，当前值为“1”，而累计(x+y的)结果还没有，因此第一次结果为“1”，第二次，当前值为“2”，累计结果为“1”所以结果为“3”，第三次，当前值为“3”累计结果为“3”，所以结果为“6”，这样的中间计算结果通过 Subscribe 发布给订阅者。

到此介绍了 Reactive Extensions(Rx) 的基本设计思想，事件的使用方法，以及一些合成方法。以后将介绍一下关于异步以及关系到时间处理的使用方法。