


Reactive Extensions (Rx) 入门(3) —— Rx的事件编程①

 blog.csdn.net/fangxing80/article/details/7628322

原文: http://www.atmarkit.co.jp/fdotnet/introRx/introRx_01/introRx_01_02.html

作者: 河合 宜文

前面两章介绍了Rx的概要和安装方法, 本章开始重点介绍Rx的具体的使用方法。首先会介绍一下使用Rx的基本操作, 然后会重点介绍Rx所代表的2大特性: 事件处理和异步处理中的事件处理。

基本方法

首先来看看最简单的Rx (Observable对象)代码和 foreach(Enumerable对象)代码的对比:

```
1. using System.Linq;
2. using System.Reactive.Linq;
3. // Observable对象
4. Observable.Range(1, 5)
5. .Subscribe(x => Console.WriteLine(x));
6. // Enumerable对象
7. foreach (var x in Enumerable.Range(1, 5))
8.     Console.WriteLine(x);
```

*需要添加 System.Reactive Assembly引用

上述代码, 运行结果都是输出1到5。Enumerable对象属于Pull型序列, Observable对象属于Push型序列(*1), 都能实现相同的效果。Enumerable是一种序列(Range), 通过foreach消费。Rx也是一种序列, 和foreach相似的, 不同的是使用Subscribe方法来进行基本的处理。

*1 Enumerable属于设计模式中的Iterator模式, Observable属于Observer模式
接下来, 具体看一下Rx的接口设计。Rx里主要的接口有两个, IObservable<T>和IObserver<T>

```
1. public interface IObservable<out T>
2. {
3.     IDisposable Subscribe(IObserver<T> observer);
4. }
5. public interface IObserver<in T>
6. {
7.     void OnCompleted();
8.     void OnError(Exception error);
9.     void OnNext(T value);
10. }
```

在最上面的实例代码中，这两个接口都用到了。IObservable<T>对象表示序列，IObserver<T>对象则表示消费的内容。也就是从 Observable.Range 获得对象是 IObservable<int>。但是 IObserver<T> 并不是能够直接看出来的，实际上 IObserver<T> 对象是 Subscribe 方法"自动"传递到 Lambda 表达式中的。下面是详细的说明：

```
1. var observer = Observer.Create<int>(
2.     x => Console.WriteLine(x), // onNext参数 (delegate)
3.     ex => { throw ex; }, // onError参数 (delegate)
4.     () => { }); // onCompleted参数 (delegate)
5. Observable.Range(1, 5).Subscribe(observer);
```

Observer.Create 方法接受3个delegate参数，分别是符合 IObserver<T>接口的 OnNext/OnError/OnCompleted 方法定义的delegate，用于实现 IObserver<T>接口。但是，这样特意去使用 Observer.Create 方法生成 IObserver<T> 实例有些麻烦。因此，IObservable<T>接口里的 Subscribe 方法不仅仅是接受 IObserver<T> 对象，还定义了许多重载方法(*2)，用于接受各种 delegate 来简化这一过程，这样就能和 Enumerable 的 foreach 达到相同的使用效果了。

*2 ObservableExtension 类里，OnNext/OnError/OnCompleted 的组合共有5个重载方法

IObserver<T>对象详解

最上面的“Rx和foreach的示例代码”中 IObserver<T> 接口的成员中，只用到了 OnNext 代理，它是在 Observer 收到并执行调用的最基本的操作。其余的2个，OnError是指发生异常时调用的方法，OnCompleted 则是序列完成之后调用的方法。同时指定这3个代理的使用，如下面代码中 Subscribe所示：

```
1. // 正常结束的时候

2. // 运行结果: 1, 2, 3, 4, 5, Completed

3. Observable.Range(1, 5)

4. .Subscribe(

5.     x => Console.WriteLine(x),

6.     ex => Console.WriteLine("Error"),

7.     () => Console.WriteLine("Completed"));

8. // 中间发生异常的时候

9. // 运行结果: 1, 2, Error

10. Observable.Range(1, 5)

11. .Do(x => { if (x == 3) throw new Exception(); })

12. .Subscribe(

13.     x => Console.WriteLine(x),

14.     ex => Console.WriteLine("Error"),

15.     () => Console.WriteLine("Completed"));
```



上面的代码执行时，Range方法生成1~5的数结束之后，调用 OnCompleted 方法(代理)，如果中间发生异常则调用OnError。另外，OnError 或者 OnCompleted 方法并不是只调用其中的一方。即不是在 OnError 之后调用 OnCompleted，也不可能是在 OnError 调用或者 OnCompleted 之后调用 OnNext。

为什么要有这样的限制呢，通过 foreach 来看看会比较清楚些：

foreach 中枚举全部完成之后，就不会再进入 foreach 循环中。而且，一旦枚举中发生异常也不会再进入循环中。

也可以看出，正如前篇里介绍的“IObservable<T> / IObservable<T>接口是由 IEnumerable<T> / IEnumerator<T>反推得来的”，但 IObservable<T>对象也并不是完全遵守IEnumerable<T>的特性，而且也不必如此。

Dispose的必要性

Subscribe方法的返回值是一个实现了 IDisposable 接口的对象，在上面的示例中中间变量被忽略了。关于Rx的IDisposable对象与一般意义上的“有可能是必须释放的资源”是有所不同的。

如果 Rx 处理的是事件的时候，那么 Dispose 表示“分离”，Timer的时候则表示“中止”，异步的时候是“取消”的意思。好吧，也确实是有“结束“的意思（但此时 OnCompleted 并没有被执行），相反，在很多情况下，Dispose 并不是必须去调用的。有有些场景下不去调用的例子反而更多。所以，就算忽略Subscribe方法的返回值也不会有什么问题的。

Observable对象的生成器

关于Eumerable序列，也就是实现了 IEnumerable接口的对象，比如数组，List，Dictionary等，有很多。但是Rx相关的序列，直接实现IObservable<T>接口的对象比较少，因此，为了生成IObservable<T>对象，准备了大量的API。

下面是Observable的生成器的整理列表。这个表中所列的是 Rx 安定版中的生成器，Experimental实验版中又追加了不少。

方法名	功能介绍
Create	生成任意的Observable
Defer	推迟执行直到被订阅为止
Empty	只执行 OnCompleted 方法
FromAsyncPattern	生成 Begin-End Pattern
FromEvent *4	从 Action 代理的事件生成
FromEventPattern	从 EventHandler 代理的事件生成
Generate	模拟for推送值
Interval	一段时间推送一个值
Never	什么也不做
Range	推送一个区间内的值
Repeat	指定次数推送同一个值
Return	只返回一个结果
Start	指定一个Schedula立即执行，完成后返回一个值
Throw	只执行 OnError

Timer	在某个时间推送
ToAsync	返回一个 Func<Observable<T>>
Using	完成后，生成将指定的资源Dispose掉的代理

这样从最开始的代码开始，经过 Where，Select 等 Linq 查询，再利用 Rx 的方法，对对象进行改造，最后交给 Subscribe 进行订阅。这就是 Rx 的基本流程了。

上图所示就是用Where过滤出偶数，并用Select 取出值乘2的结果。