Reactive Extensions入门

yycoding.xyz/post/2012/11/3/introducting-reactiveextensions

Original yycoding 2012/11/3 09:54:00 236 Reads

前面我写过7篇文章粗略的介绍了一下Rx及其方方面面。Rx是一个好东西不然我也不会费这么大的力气来写这些东西。本文打算初略的讲一下传统异步编程方法的缺点，以及为啥Rx能够给异步编程带来新的体验。最后我听译了一篇关于Reactive Extension的非常好的一篇演讲，并制作了中英文字幕。希望大家看完这篇文章之后能够对Reactive Extension能够有比较深的印象，并在实际编程中遇到比较纠结的异步编程问题了能够想到Rx。

1. 传统异步编程存在的问题

异步编程比较困难，这一点老赵讲过很多次，在这里就我的理解有以下几点。

1.1 异步编程的方式太多，缺乏统一性

在.NET下面做异步编程其实有很多种选择的，如果基于事件的异步编程，经典的Begin/End异步方法对，以及针对以上两种存在的问题改进的CCR和AsyncEnumerator，还有F#中的异步工作流，以及C#5.0中引入的Task和Async。方法多吧，这些方法在不同的场合具有不同的优缺点，但是要全部掌握对于一般的编程者来说比较难，缺乏一种统一的方式将这些异步编程方法统一的概况进来。

1.2 代码破碎，本地性不好

传统的异步编程方式迫使我们将逻辑分拆为两部分，而大家一般对线性的处理方式比较熟悉，相信大家比较熟悉基于事件的异步编程方式，一般地，你需要注册一个事件，这个事件处理方法通常和注册事件的代码不在同一个地方。在Begin/End异步方法对中，你也需要发起一个Begin异步方法，然后在其回调方法中，调用End方法。这通常会破坏代码的本地性，而且，如果你需要在Begin方法中传递一些参数的话，你还需要进行类型转换。在C#2.0中的闭包，匿名委托, C# 3.0中的lambda表达式，使得你可以在调用回调方法的地方传进去一个异步方法，这样在一定程度上减少了代码的破碎性，但是如果需要在异步方法中再次调用另外一个异步方法，那么代码就会比较难以维护了。再者对异步进行异常处理也是很麻烦的事情，你不得不在每一个地方都写上try catch。

1.3 异步方法难以进行组合

传统的异步编程也难以对方法进行组合，比如说要实现鼠标拖拽(Drag and Drop)这个功能，他涉及到多个事件，你需要记录很多个中间状态。而不像我们熟悉的LINQ那样对事件像对数据那样进行管道(Pineline)操作。

2. 什么是Rx

Rx提供了一种新的组织和协调异步事件的方式，极大的简化了代码的编写。Rx最显著的特性是使用可观察集合(Observable Collection)来达到集成异步(composing asynchronous)和基于事件(event-based)的编程的效果。Rx有一些几个特性，这一点在第一篇文章中也讲过：

- 组合(Composing): Reactive Extension的首要目标之一就是将多种异步操作组合起来是的代码更加简单。要做到这一点，数据流必须定义清楚，这样代码就很清晰集中，使得异步操作代码异步处理代码不会充斥整个应用程序。
- 异步(Asynchronous): 虽然Rx不仅仅能处理异步操作，但是使用Rx，大大简化了异步操作的实现，并且代码容易理解进而容易维护。
- 基于事件(Event-based): Rx简化了传统的异步编程方式，在后面的文章中我们会看到拖放(drag and drop)模式的实现
- Observable集合(Observable collections): Obervable Collection是Rx的核心，它是一种集合，集合的元素在第一次访问的时候可能还没有填充。它对与Rx的重要性类始于Enumerable集合对LINQ的重要性。

3. Rx给异步编程带来的新体验

3.1 统一了异步编程方式

对于基于事件的异步处理，我们可以调用Observable的FromEventPattern方法，如我们如果想监听某个对象的MouseDown事件, 使用Reactive Extension则可以像这样写：

```
IObservable<MouseEventArgs> mouseDown = Observable.FromEventPattern<MouseEventArgs>
(this.myPictureBox, "MouseDown")
                            .Select(x =>x.EventArgs);
```

这样就将一个Image的MouseDown事件转换成了一个IObservable<MouseEventArgs>对象，然后我们再调用这个对象的Subscribe方法就可以进行事件注册，异常处理了。

同样，对于Begin/End异步方法对，也可以调用Observable的FromAsyncPattern的方法，这样你也就可以对该事件进行各种操作了。

```
WebRequest request = WebRequest.Create("www.baudu.com");
           IObservable<WebResponse> webRequest = Observable.FromAsyncPattern<WebResponse>
(request.BeginGetResponse, request.EndGetResponse)
```

3.2 良好的编程体验，代码本地性好

比如上面的例子，如果要异步请求并接着处理返回的请求，那么可以直接紧接着后面对该事件进行操作：

```
IObservable<String> result = webRequest.Select(res =>
                                                {
                                                    using (var stream =
res.GetResponseStream())
                                                    using (var sr = new
StreamReader(stream))
                                                    {
                                                        return sr.ReadToEnd();
                                                    }
                                                });
```

你可以就像写同步代码那样写异步代码，不需要强行的把异步方法写成一个主体一个回调，使得代码更加的紧凑和容易维护。

3.3 可以方便的对异步事件进行组合操作

Rx可以使得可以使用类似LINQ的语法对事件进行操作，比如现在要实现一个功能：在一个form上有一个textbox，现在要监听textbox的事件，并且要在用户输入停止后1秒钟才发起事件。如果您使用Rx，则可以写为：

```
// 取得TextBox控件的TextChanged事件
IObservable<String> textChanage=Observable.FromEventPattern<EventArgs>(textBox,
"TextChanged")
                                          .Select(_ => textBox.Text)
                                          .Throttle(TimeSpan.FromSeconds(1)); // 等待1秒钟
如果没有别的变化则使用最近的变化值
```

这个逻辑在实际中是很常见的，比如你需要根据用户的输入去进行webservice请求，如果用户输入的很快，且频繁更改输入值得话，如果不进行过滤，会进行很多不必要的请求。上面这段代码则很好的满足了要求，如果您用传统的基于事件的方式实现该逻辑的话，我想是非常麻烦的。

另外一个例子：比如说，你在form上有一个图片，你想实现Drag and Drop功能，如果使用Rx的话，代码如下：

```
var mouseDown = Observable.FromEventPattern<MouseEventArgs>(this.myPictureBox,
"MouseDown")
                .Select(x => x.EventArgs.GetPosition(this));
var mouseUp = Observable.FromEventPattern<MouseEventArgs>(this.myPictureBox, "MouseUp")
                            .Select(x => x.EventArgs.GetPosition(this));
var mouseMove = Observable.FromEventPattern<MouseEventArgs>(this.myPictureBox,
"MouseMove")
                        .Select(x => x.EventArgs.GetPosition(this));
var dragandDrop = from down in mouseDown
                    from move in mouseMove.StartWith(down).TakeUntil(mouseUp)
                    select new
                    {
                        x = move.X - down.X,
                        y = move.Y - down.Y
                    };

dragandDrop.Subscribe(value =>
{
    Canvas.SetLeft(this.myPictureBox, value.x);
    Canvas.SetTop(this.myPictureBox, value.y);
});
```

可以看到代码风格是很线性化的，表面上没有回调，没有破碎的代码。可以很方便的对事件进行Pineline的操作。如果上面这部分逻辑您使用传统的方法实现的话，需要保存一系列状态，比如说什么时候开始，什么时候结束，开始和结束的位置等等，而且注册事件会使得代码比较散，可读性不如Rx的好。

4. 一些资源

为了帮助大家更好的了解Reactive Extension，我翻译了一篇在Channel9 上的演讲，DevCamp 2010 Keynote - Rx: Curing your asynchronous programming blues，个人觉得讲得非常好，对IObservable接口如何有IEnumerable演变，以及Rx如何帮助我们简化异步编程都有很详细的讲解。如果您感兴趣的话，可以直接在Channel9上看。当然，我花了很长时间听译并试着翻译了整个视频，有些地方听的不是很懂，不过应该不影响您理解，并制作了中英文的字幕，您可以到Channel9上下载高清视频，然后结合中英文字幕看，我也将该高清视频和字幕压制到一起并上传到youku上了，可能不会很清晰，还是强烈建议您Channel9下载下来看。

当然，除了这个非常好的介绍Rx的演讲之外，下面的一些英文资料也很有参考价值：

1. Rx team blog http://blogs.msdn.com/b/rxteam/
2. DevLab http://msdn.microsoft.com/en-us/data/gg577609
3. Channel 9 http://channel9.msdn.com/Tags/Rx

如果您不喜欢看英文的话，下面的一些也很有参考价值:

1. CSDN上一位网友翻译的几篇日语文章介绍Rx的，个人觉得很好：

    1. Reactive Extensions (Rx) 入门(1) —— Reactive Extensions 概要

2. Reactive Extensions (Rx) 入门(2) —— 安装 Reactive Extensions
3. Reactive Extensions (Rx) 入门(3) —— Rx的事件编程①
4. Reactive Extensions (Rx) 入门(4) —— Rx的事件编程②
5. Reactive Extensions (Rx) 入门(5) —— Rx的事件编程③

2. 上面这些都很好，当然我之前也写了几篇介绍Rx，您要是觉得有时间的话，也不防瞧一瞧：

1. Reactive Extensions入门(1)：LINQ和Rx简单介绍
2. Reactive Extensions入门(2)：LINQ操作符
3. Reactive Extensions入门(3)：Rx操作符
4. Reactive Extensions入门(4)：Rx实战
5. Reactive Extensions入门(5)：ReactiveUI MVVM框架
6. Reactive Extensions入门(6)：使用Rx进行单元测试

最后，希望这些能够给您了解Reactive Extension及对异步编程新方式带来一定的帮助。
