Reactive Extensions入门(3)：Rx操作符

yycoding.xyz/post/2012/5/25/introducting-linq-and-reactiveextensions-core-rx

Original yycoding 2012/5/25 19:27:00 7617 Reads

在第一篇文章里讲过，Reactive Extensions(Rx)是对LINQ的一种扩展，其关键的特性在于他不是基于IEnumerable接口的，Rx是基于IObservable接口，这个接口用来迭代Observable集合。自然Observable集合是基于观察者模式设计的。Observable模式的关键在于被观察的对象有一些行为或者属性，观察者可以注册某些感兴趣的属性或者行为。当被观察者发生状态改变时，会通知观察者（通常是发起一个事件）。

观察者模式是发布和订阅模式(Publish and Subscribe Pattern)的一个子集，发布者可以产生一些告知其订阅者他的一些状态信息。在Rx中Observable集合有订阅者，简单来说，就是观察Observable集合的行为，称之为对象订阅了这一集合。

在Rx中，Observable集合在有新的元素添加到集合中去的时候，就会通知其订阅者。这是Observable集合和Enumerable集合的最本质区别。对象订阅Observable集合时，可能集合中的元素不存在，但是当集合中的元素在未来某一时刻到来时，会通知其订阅者。

1. IObservable 和 IObserver接口

如前所述，Rx的关键在于Observable和其订阅者之间的关系。相比LINQ的IEnumerable和IEnumerator接口，Rx的关键接口为IObservable和IObserver。

IObservable接口定义如下：

```
public interface IObservable<out T>
{
    IDisposable Subscribe(IObserver<T> observer);
}
```

IObserver接口定义如下：

```
public interface IObserver<in T>
{
    void OnCompleted();

    void OnError(Exception error);

    void OnNext(T value);
}
```

为了能够从Observable集合获取到通知，需要使用Subscribe方法传入一个Observer对象。注意到Subscribe方法返回IDisposable对象，这使得可以用来注销对Observable对象的注册。Reactive Extension会为我们处理所有这些注册和注销操作。如Observer接口定义那样，传入Subscribe方法中的Observer对象有三个方法。

2. 创建Observables对象

为了演示使用不同的操作符创建Observables对象，需要建立一个项目。首先创建一个Windows Phone 项目，然后添加Microsoft.Phone.Reactive和System.Observable引用。

在主界面XAML文件中添一个名为Messages 的Listbox控件，并设置HorizontalAlignment 和VerticalAlignment属性，并将这两个属性设置为stretch，这个控件用来输出我们后面测试的数据。

```
<Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0">
    <ListBox
        Name="Messages"
        HorizontalAlignment="Stretch"
        VerticalAlignment="Stretch"
        Margin="10" />
</Grid>
```

然后再后台代码中创建一个Observer方法。该方法注册Observables并将结果输出到Listbox中来，可以在Main方法中调用该方法。

```
private void Observe()
{
    IObservable<Int32> source = Observable.Return(42);
    IDisposable subscription = source.Subscribe(
        x=>Messages.Items.Add("OnNext:"+x),
        ex=>Messages.Items.Add("OnError:"+ex.Message),
        ()=>Messages.Items.Add("OnComplated")
        );
}
```

下面我们演示以不同的方式返回Observables对象，只需要改变

```
IObservable<Int32> source = Observable.Return(42);
```

这句代码即可。

使用Return方法返回Observable对象

上面的代码就演示了如何使用Observable对象的Return方法来产生Observable对象，运行后显示如下结果:

```
OnNext: 42
OnCompleted
```

使用Empty方法返回Observable对象

还有一种比return方法更简单的产生Observable对象的方式是直接调用Observable对象的Empty方法。该方法不返回任何值。我们将之前的那句代码改为

```
IObservable<Int32> source = Observable.Empty<Int32>();
```

运行程序，输出为:

```
OnCompleted
```

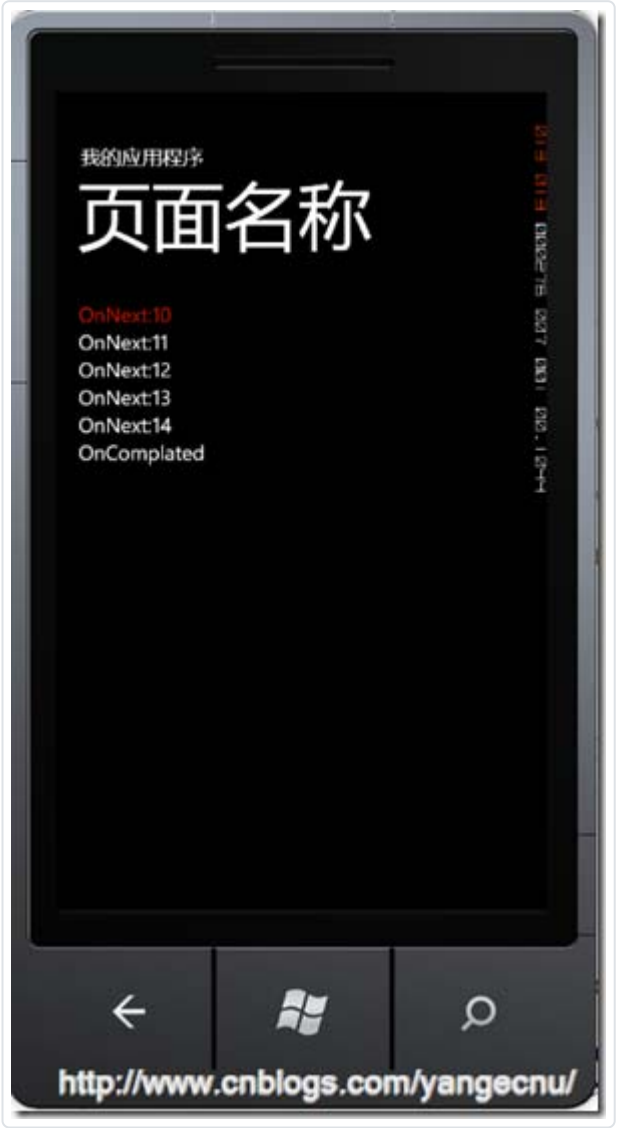
使用Range方法返回Observable对象

比Return和Empty更有用的一个生产Observable对象的是Range方法，他和Enumerable对象的Range方法含义类似，该方法接受两个参数，一个开始值，以及产生值的个数。

我们将之前的那段语句改为:

```
IObservable<Int32> source = Observable.Range(10, 5);
```

运行程序，输出结果为：



我们把下面的语句放到控制台应用程序中：

```
var input = Observable.Range(10, 5);
input.Sum().Subscribe(x => Console.WriteLine("The sum is :"+x));
```

这两句将Observable对象中的数据相加，然后将结果打印到控制台上。运行程序后输出结果为：

```
The Sum is 5050
```

从Array对象产生Observable对象

我们也可以使用IEnumerable对象的ToObservable扩展方法产生Observable对象，如下将数组转换为Observable集合，并将集合元素打印到控制台上。

```
var myArray = new[] { 1, 3, 5, 7, 9 };
varmyObservable = myArray.ToObservable();
myObservable.Subscribe(x => Console.WriteLine("Integer:{0}", x));
```

3. 通过事件创建Observable对象

在通过事件创建Observable对象中，我们可以把键盘或者鼠标想象成是一个按键或者鼠标移动的集合。下面以一个例子来展现如何通过事件创建Observable对象。

搜索维基百科

使用Observable对象的FromEvent方法我们可以很容易的将一系列事件转换为Observable集合。光说可能不容易理解，下面我们以一个例子来展示如何实现。这个例子通过将捕获到的用户输入和搜索事件转换为Observable事件集合来实现从维基百科中根据用户输入的数据查找到想要的结果。

首先我们创建一个新的名为SearchWikipedia的Windows Phone项目，在界面上，我们放置如下对象：

Control	Name	Text	Width
TextBox	Search		Full Width
TextBlock	lblSearch	Searching for...	Full Width
TextBlock	lblProgress	Loading...	Auto
webBrowser	webBrowser1		Full Width

前端代码如下:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
    </Grid.RowDefinitions>

    <!--TitlePanel 包含应用程序的名称和页标题-->
    <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
        <TextBlock x:Name="ApplicationTitle" Text="维基搜索" Style="{StaticResource
PhoneTextNormalStyle}"/>
        <TextBlock x:Name="PageTitle" Text="搜索" Margin="9,-7,0,0" Style="
{StaticResource PhoneTextTitle1Style}"/>
    </StackPanel>

    <!--ContentPanel - 在此处放置其他内容-->
    <StackPanel Grid.Row="1" HorizontalAlignment="Stretch" VerticalAlignment="Stretch">
        <TextBox Name="Search" Text="" HorizontalAlignment="Stretch" />
        <TextBlock Name="lblSearch" Text="Searching for..." />
        <TextBlock Name="lblProgress" Text="Loading..." Visibility="Collapsed"/>
        <phone:WebBrowser Name="webBrowser1" HorizontalAlignment="Stretch" Height="469"
/>
    </StackPanel>
</Grid>
```

如前所属，响应式编程最常用的是观察者模式。一般是一个被观察者和多个观察者。在这个例子中通过键盘输入到TextBox中的内容是被观察者。

但是我们并不是对输入到TextBox中的文字实时响应，而是我们要等到用户暂停或者停止输入时在触发，我们可以通过将KeyDown事件转换为Observable对象，然后使用对象的Throttle扩展方法来告诉Rx我们并不会马上相应TextBox中的内容，而是要等到用户停止输入时再响应，比如，当用户点击键盘半秒之后再触发。

要实现半秒后再触发事件可能需要设定一个计时器，然后调用回调方法，可能还需要写一些代码。使用Rx的话，只需要添加一个Throttle操作符即可，该方法帮我们搞定了计时器及一些相关的代码逻辑。

首先，我们需要从KeyUp事件创建一个Observable对象，这只需要调用Observable对象的FromEvent方法即可，该方法接受一个触发事件的对象（本例中是名为Search的文本框），一个具体的事件名称（本例中是文本框的”KeyUp”）。将下面的代码添加到MainPage的构造函数中：

```
var keys = Observable.FromEvent<KeyEventArgs>(Search, "KeyUp");
```

然后添加Throttle扩展方法：

```
var keys = Observable.FromEvent<KeyEventArgs>(Search, "KeyUp")
                        .Throttle(TimeSpan.FromSeconds(0.5));
```

是在FromeEvent对象返回的结果上调用Throttle方法的。将鼠标移到Throttle方法上可以看到他是IObservable<IEvent<KeyEventArgs>>>对象的扩展方法，而FromEvent对象正好返回的是这个对象。

现在我们基于用户在Textbox中停止输入后半秒这一事件，创建了一个名为Keys的Observable集合。

Observable对象创建完了之后，我们需要注册。注册可能不在UI线程中，如果这样，当我们试图往UI线程中写入东西时，就会抛出异常，这是由于Silverlight中所谓的Thread Affinity限制。因此确保在UI线程中注册Observable对象显得尤为重要。我们可以通过ObserveOn(Deployment.Current.Dispatcher)，这个方法实现。该方法保证我们在UI线程中注册Observable对象。

```
keys.ObserveOn(Deployment.Current.Dispatcher)
```

现在，我们在正确的线程上了，然后我们可以调用Subscribe方法。

```
keys.ObserveOn(Deployment.Current.Dispatcher).Subscribe
```

每次当用户停止输入半秒后，输入的内容就会传入到lambada表达式中，在表达式中我们可以更新两个文本对象（lblSearch和lblProgress）和一个Web Browser对象。

```
keys.ObserveOn(Deployment.Current.Dispatcher).Subscribe(
        evt =>
        {
            lblSearch.Text = "Searching for …" + Search.Text;
            lblProgress.Visibility = Visibility.Visible;
            webBrowser1.Navigate(new Uri("http://en.wikipedia.org/wiki/" + Search.Text));
        }
    );
```

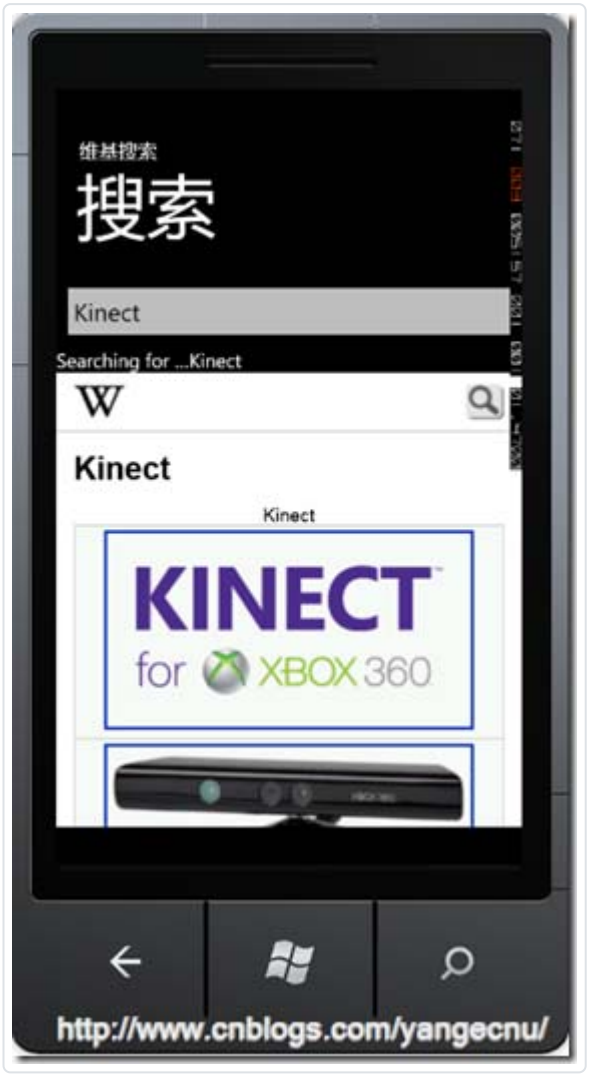
当browser对象跳转到正确的URI对象后，我们需要隐藏lblProgress对象。因此我们需要注册browser对象的Navigated事件，该事件会在broswer对象加载完成之后触发。因此当该事件触发时，我们需要将lblProgress对象的visibility属性更改为Collapsed。

```
var browser = Observable.FromEvent<NavigationEventArgs>(webBrowser1, "Navigated");
browser.ObserveOn(Deployment.Current.Dispatcher).Subscribe(evt =>
        {
            lblProgress.Visibility = Visibility.Collapsed;
        }
    );
```

这几条简单的语句很好的概括了如何从事件中创建Observable对象。下面来仔细分析一下上面的代码：

- var browser：存储从Observable对对象返回的集合。
- Observable.FromEvent:通过FromEvent的方法获取Observable对象。
- <NavigationEventArgs>:从Observable对象中创建的事件的类型。
- (webBrowser1, "Navigated")：定义事件触发者和我们感兴趣的事件。
- browser.ObserveOn(Deployment.Current.Dispatcher)：确保我们处在UI线程中。
- Subscribe：注册处理方法。
- (evt => ： lambda表达式处理逻辑。

这个例子中，我们我们从KeyUp事件中创建了一个Observable集合，并为其注册了一些处理方法。运行程序，结果如下：



一点改进

因为Observable对象是一级对象，和Enumerable一样，我们使用扩展方法可以直接对Observable进行一些类似LINQ化的操作。例如我们想获取事件中Textbox对象text属性。

```
var keys = Observable.FromEvent<KeyEventArgs>(Search, "KeyUp")
                             .Select(x=>((TextBox)x.Sender).Text)
                             .Throttle(TimeSpan.FromSeconds(0.5));
```

上面的语句中，我们对IEvent<KeyEventArgs>数据类型进行查询操作，我们要获取到触发事件的TextBox对象的Text属性，这个和平常对sender对象进行转换非常相似。获取了之后，下面我们可以将之前的语句改写为直接使用参数了，而不是再调用search.Text，这样更加友好和合理。

```
keys.ObserveOn(Deployment.Current.Dispatcher).Subscribe(evt =>
{
    lblSearch.Text = "Search for ……" + evt;
    lblProgress.Visibility = Visibility.Visible;
    webBrowser1.Navigate(new Uri("http://en.wikipedia.org/wiki/"+evt));
});
```

运行程序，结果和之前的完全一样。

4. Rx一些操作符

Rx中的一些操作符和前一篇文章中介绍的LINQ操作符有很多功能是相同的。下面对最常用的take,skip,distinct,using和zip这个操作符进行说明。

Take

Rx中的Take操作符和LINQ中的功能一样，它用来指定获取集合中的前几项。

```
var input = new[] { 1, 2, 3, 4, 5, 4, 3, 2, 1 }.ToObservable();
input.Take(5).Select(x => x * 10).Subscribe(x => Console.WriteLine(x));
```

输出结果为：

```
10
20
30
40
50
```

Skip

Skip语句表示跳过集合中的n条记录。这在有些情况下非常有用，比如前一篇文章中解析文本的时候，可能第一行是表头，所以可以使用skip跳过第一行，从第二行开始读取。还有就是在分页的时候和take一起使用非常方便。

```
var input = new[] { 1, 2, 3, 4, 5, 4, 3, 2, 1 }.ToObservable();
input.Skip(6).Select(x => x * 10).Subscribe(x => Console.WriteLine(x));
```

输出结果为：

```
30
20
10
```

Distinct

Distinct用来去除集合中的非重复数据。

```
var input = new[] { 1, 2, 3, 4, 5, 4, 3, 2, 1 }.ToObservable();
input.Distinct().Select(x => x * 10).Subscribe(x => Console.WriteLine(x));
```

输出结果为：

```
10
20
30
40
50
```

Using

Rx也需要清理资源，当使用到了一些受限制资源或者非托管资源时，需要我们去管理这些资源的释放。

当然，我们可以调用Observable对象的一个称之为Using的静态方法。方法返回一个IObservable<char>类型对象，接受两个参数，第一个参数是一个返回StreamReaderde的Func类型参数，第二个是一个接受第一Func参数返回的StreamReader对象，返回一个类型为char的IObservable集合。

```
var ObservableStrings = Observable.Using<char, StreamReader>(

    );
```

第一个参数创建一个StreamReader对象并打开一个文件。

```
()=>new StreamReader(new FileStream("randomtext.txt",FileMode.Open)),
```

第二个参数使用之前创建的StreamReader对象，调用ReadToEnd()方法并将结果转换为Char数组，最后IEnumerable的ToObservable方法将数组转换为Observable集合。

```
streamReader => (streamReader.ReadToEnd().Select(str => str)).ToObservable()
```

完整代码如下：

```
var ObservableStrings = Observable.Using<char, StreamReader>(
        () => new StreamReader(new FileStream("randomtext.txt", FileMode.Open)),
        streamReader => (streamReader.ReadToEnd().Select(str => str)).ToObservable()
    );
ObservableStrings.Subscribe(Console.Write);
```

randomtext.txt是一个文本文件，里面内容为：

hello reactive framework

运行程序后，输出结果为：

```
hello reactive framework
```

Zip

和LINQ中的Zip操作类似。LINQ中的Zip是将两个集合合并为一个新的集合，在Rx中Zip是将两个Observable对象合并为一个新的Observable对象。

```
var listOne = Observable.Range(0, 100);
var listTwo = new String[] { "北京", "上海", "重庆", "天津", "重庆" }.ToObservable();
var numberCitys = listOne.Zip(listTwo, (num, city) => num + ":" + city);
numberCitys.Subscribe(Console.WriteLine);
```

运行程序，输出结果为：

```
0:北京
1:上海
2:重庆
3:天津
4:重庆
```

掌握了这些操作符之后，下面来以一个有点复杂的例子来展现这些操作符是如何和Observable对象一起使用的。

拖放操作

在这个例子中，我们将鼠标作为一系列位置点的Observable集合，然后将这个注册到其他事件上，当有新的点加入到集合中时，会触发事件。

首先创建一个Windows Phone应用程序。添加一个TextBlock控件和一个Image控件，前端代码如下：

```
<Canvas>
    <TextBlock Name="textBlock" Text="Rx for Silverlight" />
    <Image Name="image"
                Source="avatar.png"
                Width="191"
                Height="206"
                Canvas.Left="12"
                Canvas.Top="50" />
</Canvas>
```

现在我们要从创建Image对象的MouseDown，MouseUp和MouseMove事件分别创建Observable集合。这些集合可以通过之前讲过的Observable对象的FromEvent方法创建。

```
var mousedown = Observable.FromEvent<MouseButtonEventArgs>(image, "MouseLeftButtonDown")
                                        .Select(x => x.EventArgs.GetPosition(image));

var mouseup = Observable.FromEvent<MouseButtonEventArgs>(this, "MouseLeftButtonUp");

var mousemove = Observable.FromEvent<MouseEventArgs>(this, "MouseMove")
                              .Select(x => x.EventArgs.GetPosition(this));
```

注意到只有MouseDown事件的触发者是image控件。我们只关心当用户点击到图片上时的事件。MouseUp和MouseMove不使用image控件的事件，而是使用整个canvas。

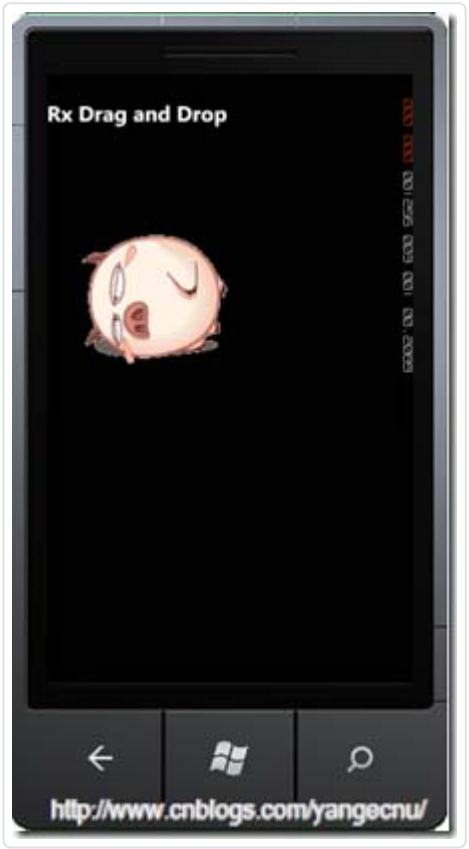
有了这些基于事件的集合之后，我们能够创建出一个新的类在接收到MouseUp之前的鼠标移动的偏移量。

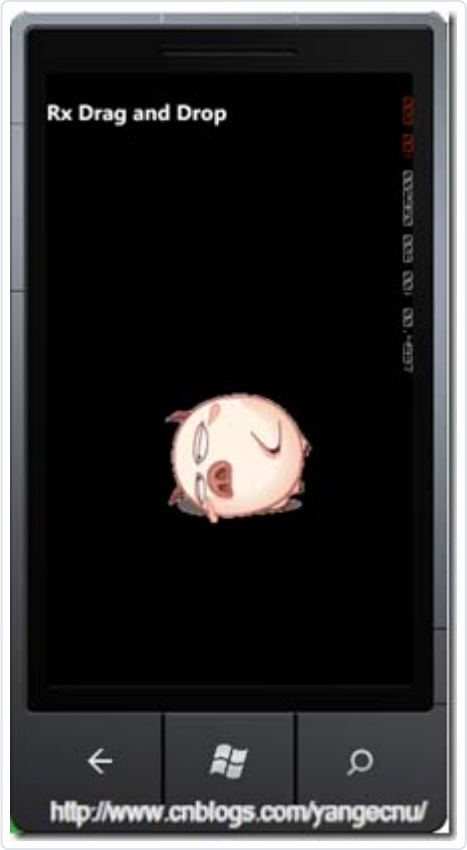
```
var q = from start in mousedown
            from end in mousemove.TakeUntil(mouseup)
        select new
        {
            X = end.X - start.X,
            Y = end.Y - start.Y
        };
```

将鼠标移动到q上，vs会提示该对象是一个IObservable<anonymous>对象，因此可以给对象注册事件，我们使用q对象中的值来创建注册方法。

```
q.Subscribe(value =>
{
    Canvas.SetLeft(image, value.X);
    Canvas.SetTop(image, value.Y);
});
```

运行程序，一个简单的图片随鼠标拖动的效果就实现出来了。





5. 结语

本文首先介绍了创建Observable集合的几种方式，并详细的以web页面查询为例讲解了如何通过事件创建Observable集合。接着讲述了Observable集合的一些操作符，这些操作符可以在实际应用中帮助我们对Observable集合进行一些处理，最后以一个拖放图片例子来综合演示了如何从事件创建Observable集合，以及如何运用Observable的操作符来对集合进行处理。希望本文对您了解Reactive Framework中的Observable对象有所帮助！