عنوان: **ایجاد توالی‌ها در Reactive extensions**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۵ ۱۸:۵
آدرس: www.dotnettips.info
گروه‌ها: Asynchronous Programming, Reactive Extensions, Rx

در مطلب « معرفی Reactive extensions » با نحوه‌ی تبدیل IEnumerableها به نمونه‌های Observable آشنا شدیم. اما سایر حالات چطور؟ آیا Rx صرفا محدود است به کار با IEnumerableها؟ در ادامه نگاهی خواهیم داشت به نحوه‌ی تبدیل بسیاری از منابع داده دیگر به توالی‌های Observable قابل استفاده در Rx.

روش‌های متفاوت ایجاد توالی (sequence) در Rx

الف) استفاده از متدهای Factory

1) Observable.Create

نمونه‌ای از استفاده از آن‌را در مطلب « معرفی Reactive extensions » مشاهده کردید.

```
var query = Enumerable.Range(1, 5).Select(number => number);
var observableQuery = query.ToObservable();
var observer = Observer.Create<int>(onNext: number => Console.WriteLine(number));
observableQuery.Subscribe(observer);
```

کار آن، تدارک delegate ایی است که توسط متد Subscribe، به ازای هربار پردازش مقدار موجود در توالی معرفی شده به آن، فراخوانی می‌گردد و هدف اصلی از آن این است که به صورت دستی اینترفیس IObservable را پیاده سازی نکنید (امکان پیاده سازی inline یک اینترفیس توسط Actionها).
البته در این مثال فقط delegate مربوط به onNext را ملاحظه می‌کند. توسط سایر overloadهای آن امکان ذکر delegateهای OnError/OnCompleted نیز وجود دارد.

2) Observable.Return

برای ایجاد یک خروجی Observable از یک مقدار مشخص، می‌توان از متد جنریک Observable.Return استفاده کرد. برای مثال:

```
var observableValue1 = Observable.Return("Value");
var observableValue2 = Observable.Return(2);
```

در ادامه نحوه‌ی پیاده سازی این متد را توسط Observable.Create مشاهده می‌کنید:

```
public static IObservable<T> Return<T>(T value)
        {
            return Observable.Create<T>(o =>
            {
                o.OnNext(value);
                o.OnCompleted();
                return Disposable.Empty;
            });
        }
```

البته دو سطر نوشته شده در اصل معادل هستند با سطرهای ذیل؛ که ذکر نوع جنریک آن‌ها ضروری نیست. زیرا به صورت خودکار از نوع آرگومان معرفی شده، تشخیص داده می‌شود:

```
var observableValue1 = Observable.Return<string>("Value");
var observableValue2 = Observable.Return<int>(2);
```

3) Observable.Empty

برای بازگشت یک توالی خالی که تنها کار اطلاع رسانی onCompleted را انجام می‌دهد.

```
var emptyObservable = Observable.Empty<string>();
```

در کدهای ذیل، پیاده سازی این متد را توسط Observable.Create مشاهده می‌کنید:

```
public static IObservable<T> Empty<T>()
        {
            return Observable.Create<T>(o =>
            {
                o.OnCompleted();
                return Disposable.Empty;
            });
        }
```

4) Observable.Never

برای بازگشت یک توالی بدون قابلیت اطلاع رسانی و notification

```
var neverObservable = Observable.Never<string>();
```

این متد به نحو زیر توسط Observable.Create پیاده سازی شده‌است:

```
public static IObservable<T> Never<T>()
        {
            return Observable.Create<T>(o =>
            {
                return Disposable.Empty;
            });
        }
```

5) Observable.Throw

برای ایجاد یک توالی که صرفا کار اطلاع رسانی OnError را توسط استثنای معرفی شده به آن انجام می‌دهد.

```
var throwObservable = Observable.Throw<string>(new Exception());
```

در ادامه نحوه‌ی پیاده سازی این متد را توسط Observable.Create مشاهده می‌کنید:

```
public static IObservable<T> Throws<T>(Exception exception)
        {
            return Observable.Create<T>(o =>
            {
                o.OnError(exception);
                return Disposable.Empty;
            });
        }
```

6) توسط Observable.Range

به سادگی می‌توان بازه‌ی Observable ایی را ایجاد کرد:

```
var range = Observable.Range(10, 15);
range.Subscribe(Console.WriteLine, () => Console.WriteLine("Completed"));
```

7) Observable.Generate

اگر بخواهیم عملیات Observable.Range را پیاده سازی کنیم، می‌توان از متد Observable.Generate استفاده کرد:

```
public static IObservable<int> Range(int start, int count)
        {
            var max = start + count;
            return Observable.Generate(
                initialState: start,
```

```
                condition: value => value < max,
                iterate: value => value + 1,
                resultSelector: value => value);
        }
```

توسط پارامتر initialState، مقدار آغازین را دریافت می‌کند. پارامتر condition، مشخص می‌کند که توالی چه زمانی باید خاتمه یابد. در پارامتر iterate، مقدار جاری دریافت شده و مقدار بعدی تولید می‌شود. resultSelector کار تبدیل و بازگشت مقدار خروجی را به عهده دارد.

8) Observable.Interval

عموما از انواع و اقسام تایمرهای موجود در دات نت مانند System.Threading.Timer ، System.Timers.Timer و System.Windows.Threading.DispatcherTimer برای ایجاد یک توالی از رخدادها استفاده می‌شود. تمام این‌ها را به سادگی می‌توان توسط متد Observable.Interval، که قابل انتقال به تمام پلتفرم‌هایی است که Rx برای آن‌ها تهیه شده‌است، جایگزین کرد:

```
var interval = Observable.Interval(period: TimeSpan.FromMilliseconds(250));
interval.Subscribe(Console.WriteLine, () => Console.WriteLine("completed"));
```

در اینجا تایمر تهیه شده، هر 450 میلی‌ثانیه یکبار اجرا می‌شود. برای خاتمه‌ی آن باید شیء interval را Dispose کنید.
Overload دوم این متد، امکان معرفی scheduler و اجرای بر روی تردی دیگر را نیز میسر می‌کند.

9) Observable.Timer

تفاوت Observable.Timer با Observable.Interval در مفهوم پارامتر ارسالی به آن‌ها است:

```
var timer = Observable.Timer(dueTime: TimeSpan.FromSeconds(1));
 timer.Subscribe(Console.WriteLine, () => Console.WriteLine("completed"));
```

یکی due time دارد (مدت زمان صبر کردن تا تولید اولین خروجی) و دیگری period (به صورت متوالی تکرار می‌شود).
خروجی Observable.Interval مثال زده شده به نحو زیر است و خاتمه‌ای ندارد:

0
1
2
3
4
5

اما خروجی Observable.Timer به نحو ذیل بوده و پس از یک ثانیه، خاتمه می‌یابد:

0
completed

متد Observable.Timer دارای هفت overload متفاوت است که توسط آن‌ها dueTime (مدت زمان صبر کردن تا تولید اولین خروجی)، period (کار Observable.Timer را به صورت متوالی در بازه‌ی زمانی مشخص شده تکرار می‌کند) و scheduler (تعیین ترد اجرایی عملیات) قابل مقدار دهی هستند.
اگر می‌خواهید Observable.Timer بلافاصله شروع به کار کند، مقدار dueTime آن‌را مساوی TimeSpan.Zero قرار دهید. به این ترتیب یک Observable.Interval را به وجود آورده‌اید که بلافاصله شروع به کار کرده است و تا مدت زمان مشخص شده‌ای جهت اجرای اولین callback خود صبر نمی‌کند.

ب) تبدیلگرهایی که خروجی IObservable ایجاد می‌کنند

برای تبدیل مدل‌های برنامه نویسی Async قدیمی دات نت مانند APM، رخدادها و امثال آن به معادل‌های Rx، متدهای الحاقی خاصی تهیه شده‌اند.

1) تبدیل delegates به معادل Observable

متد Observable.Start، امکان تبدیل یک Func یا Action زمانبر را به یک توالی observable میسر می‌کند. در این حالت به صورت پیش فرض، پردازش عملیات بر روی یکی از تردهای ThreadPool انجام می‌شود.

```
static void StartAction()
        {
            var start = Observable.Start(() =>
            {
                Console.Write("Observable.Start");
                for (int i = 0; i < 10; i++)
                {
                    Thread.Sleep(100);
                    Console.Write(".");
                }
            });
            start.Subscribe(
               onNext: unit => Console.WriteLine("published"),
               onCompleted: () => Console.WriteLine("completed"));
        }

        static void StartFunc()
        {
            var start = Observable.Start(() =>
            {
                Console.Write("Observable.Start");
                for (int i = 0; i < 10; i++)
                {
                    Thread.Sleep(100);
                    Console.Write(".");
                }
                return "value";
            });
            start.Subscribe(
               onNext: Console.WriteLine,
               onCompleted: () => Console.WriteLine("completed"));
        }
```

در اینجا دو مثال از بکارگیری Action و Funcها را توسط Observable.Start مشاهده می‌کنید.
زمانیکه از Func استفاده می‌شود، تابع یک خروجی را ارائه داده و سپس توالی خاتمه می‌یابد. اگر از Action استفاده شود، نوع Observable بازگشت داده شده از نوع Unit است که در برنامه نویسی functional معادل void است و هدف از آن مشخص سازی پایان عملیات Action می‌باشد. Unit دارای مقداری نبوده و صرفا سبب اجرای اطلاع رسانی OnNext می‌شود.
تفاوت مهم Observable.Start و Observable.Return در این است که Observable.Start مقدار تابع را به صورت تنبل (lazily) پردازش می‌کند، اما Observable.Return پردازش حریصانه‌ای (eagrly) را به همراه خواهد داشت. به این ترتیب Observable.Start بسیار شبیه به یک Task (پردازش‌های غیرهمزمان) عمل می‌کند.
در اینجا شاید این سؤال مطرح شود که استفاده از قابلیت‌های Async سی‌شارپ 5 برای اینگونه کارها مناسب است یا Rx؟ قابلیت‌های Async بیشتر به اعمال مخصوص IO bound مانند کار با شبکه، دریافت فایل از اینترنت، کار با یک بانک اطلاعاتی خارج از مرزهای سیستم، مرتبط می‌شوند؛ اما اعمال CPU bound مانند محاسبات سنگین حاصل از توالی‌های observable را به خوبی می‌توان توسط Rx مدیریت کرد.

2) تبدیل Events به معادل Observable

دات نت از روزهای اول خود به همراه یک event driven programming model بوده‌است. Rx متدهایی را برای دریافت یک رخداد و تبدیل آن به یک توالی Observable ارائه داده‌است. برای نمونه ObservableCollection زیر را درنظر بگیرید

```
var items = new System.Collections.ObjectModel.ObservableCollection<string>
  {
         "Item1", "Item2", "Item3"
```

```
  };
```

اگر بخواهیم مانند روش‌های متداول، حذف شدن آیتم‌های آن‌را تحت نظر قرار دهیم، می‌توان نوشت:

```
items.CollectionChanged += (sender, ea) =>
            {
                if (ea.Action == NotifyCollectionChangedAction.Remove)
                {
                    foreach (var oldItem in ea.OldItems.Cast<string>())
                    {
                        Console.WriteLine("Removed {0}", oldItem);
                    }
                }
            };
```

این نوع کدها در WPF زیاد کاربرد دارند. اکنون معادل کدهای فوق با Rx به صورت زیر هستند:

```
var removals =
                Observable.FromEventPattern<NotifyCollectionChangedEventHandler,
NotifyCollectionChangedEventArgs>
                (
                    addHandler: handler => items.CollectionChanged += handler,
                    removeHandler: handler => items.CollectionChanged -= handler
                )
                .Where(e => e.EventArgs.Action == NotifyCollectionChangedAction.Remove)
                .SelectMany(c => c.EventArgs.OldItems.Cast<string>());

            var disposable = removals.Subscribe(onNext: item => Console.WriteLine("Removed {0}",
item));
```

با استفاده از متد Observable.FromEventPattern می‌توان معادل Observable رخداد CollectionChanged را تهیه کرد. پارامتر اول جنریک آن، نوع رخداد است و پارامتر اختیاری دوم آن، EventArgs این رخداد. همچنین با توجه به قسمت Where نوشته شده، در این بین مواردی را که Action مساوی حذف شدن را دارا هستند، فیلتر کرده و نهایتا لیست Observable آن‌ها بازگشت داده می‌شوند. اکنون می‌توان با استفاده از متد Subscribe، این تغییرات را دریافت کرد. برای مثال با فراخوانی

```
items.Remove("Item1");
```

بلافاصله خروجی Removed item1 ظاهر می‌شود.

3) تبدیل Task به معادل Observable

متد ToObservable واقع در فضای نام System.Reactive.Threading.Tasks را بر روی یک Task نیز می‌توان فراخوانی کرد:

```
var task = Task.Factory.StartNew(() => "Test");
var source = task.ToObservable();
source.Subscribe(Console.WriteLine, () => Console.WriteLine("completed"));
```

البته باید دقت داشت استفاده از Task دات نت 4.5 که بیشتر جهت پردازش‌های async اعمال I/O-bound طراحی شده‌است، بر IObservable مقدم است. صرفا اگر نیاز است این Task را با سایر observables ادغام کنید از متد ToObservable برای کار با آن استفاده نمائید.

4) تبدیل IEnumerable به معادل Observable

با این مورد تاکنون آشنا شده‌اید. فقط کافی است متد ToObservable را بر روی یک IEnumerable، جهت تهیه خروجی Observable فراخوانی کرد.

5) تبدیل APM به معادل Observable

APM یا Asynchronous programming model، همان روش کار با متدهای Async با نام‌های BeginXXX و EndXXX است که از نگارش‌های آغازین دات نت به همراه آن بوده‌اند. کار کردن با آن مشکل است و مدیریت آن به همراه پراکندگی‌های بسیاری جهت کار با callbacks آن است. برای تبدیل این نوع روش برنامه نویسی به روش Rx نیز متدهایی پیش بینی شده‌است؛ مانند Observable.FromAsyncPattern.

یک نکته

کتابخانه‌ای به نام Rxx بسیاری از این محصور کننده‌ها را تهیه کرده‌است:

http://Rxx.codeplex.com

ابتدا بسته‌ی نیوگت آن‌را نصب کنید:

```
PM> Install-Package Rxx
```

سپس برای نمونه، برای کار با یک فایل استریم خواهیم داشت:

```
using (new FileStream("file.txt", FileMode.Open)
                .ReadToEndObservable()
                .Subscribe(x => Console.WriteLine(x.Length)))
{
        Console.ReadKey();
}
```

متد ReadToEndObservable یکی از متدهای الحاقی کتابخانه‌ی Rxx است.