عنوان: پردازش توالی توالی‌ها در Reactive extensions
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۳۰ ۱۴:۰
آدرس: www.dotnettips.info
گروه‌ها: Asynchronous Programming, Reactive Extensions, Rx

به صورت پیش فرض، Rx هر بار تنها یک مقدار را بررسی می‌کند. اما گاهی از اوقات نیاز است تا در هربار، بیشتر از یک مقدار دریافت و پردازش شوند. برای این منظور Rx متدهای الحاقی ویژه‌ای را به نام‌های Scan، Buffer و Window تدارک دیده‌است تا بتواند از یک توالی، چندین توالی را تولید کند (توالی توالی‌ها = Sequence of sequences).

متد Scan

فرض کنید قصد دارید تعدادی عدد را با هم جمع بزنید. برای اینکار عموما عدد اول با عدد دوم جمع زده شده و سپس حاصل آن با عدد سوم جمع زده خواهد شد و به همین ترتیب تا آخر توالی. کار متد Scan نیز دقیقا به همین نحو است. هربار که قرار است توالی پردازش شود، حاصل عملیات مرحله‌ی قبل را در اختیار مصرف کننده قرار می‌دهد.
در مثال ذیل، قصد داریم حاصل جمع اعداد موجود در آرایه‌ای را بدست بیاوریم:

```
var sequence = new[] { 12, 3, -4, 7 }.ToObservable();
var runningSum = sequence.Scan((accumulator, value) =>
{
    Console.WriteLine("accumulator {0}", accumulator);
    Console.WriteLine("value {0}", value);
    return accumulator + value;
});
runningSum.Subscribe(result => Console.WriteLine("result {0}\n", result));
```

با این خروجی

```
result 12

accumulator 12
value 3
result 15

accumulator 15
value -4
result 11

accumulator 11
value 7
result 18
```

در اولین بار اجرای متد Subscribe، کار مقدار دهی accumulator با اولین عنصر آرایه صورت می‌گیرد.
در دفعات بعدی، مقدار این accumulator با عدد جاری جمع زده شده و حاصل این عملیات در تکرار آتی، مجددا توسط accumulator قابل دسترسی خواهد بود.

یک نکته: اگر علاقمند باشیم که مقدار اولیه‌ی accumulator، اولین عنصر توالی نباشد، می‌توان آن‌را توسط پارامتر seed متد Scan مقدار دهی کرد:

```
var runningSum = sequence.Scan(seed: 10, accumulator: (accumulator, value) =>
```

متد Buffer

متد بافر، کار تقسیم یک توالی را به توالی‌های کوچکتر، بر اساس زمان، یا تعداد عنصر مشخص شده، انجام می‌دهد. برای مثال در برنامه‌های دسکتاپ شاید نیازی نباشد تا به ازای هر عنصر توالی، یکبار رابط کاربری را به روز کرد. عموما بهتر است تا تعداد

مشخصی از عناصر یکجا پردازش شده و نتیجه‌ی این پردازش به تدریج نمایش داده شود.

```
var sequence = Enumerable.Range(1, 200)
                 .ToObservable()
                 .Buffer(count: 10);

sequence.Subscribe(onNext: numbers =>
{
   Console.WriteLine(numbers.Sum());
});
```

در اینجا نحوه‌ی استفاده از متد بافر را به همراه مشخص کردن تعداد اعضای بافر ملاحظه می‌کنید. هربار که onNext متد Subscribe فراخوانی شود، 10 عنصر از توالی را در اختیار خواهیم داشت (بجای یک عنصر حالت متداول بافر نشده).
به این ترتیب می‌توان فشار حجم اطلاعات ورودی با فرکانس بالا را کنترل کرد و در نتیجه از منابع موجود بهتر استفاده نمود. برای مثال اگر می‌خواهید عملیات bulk insert را انجام دهید، می‌توان بر اساس یک batch size مشخص، گروه گروه اطلاعات را به بانک اطلاعاتی اضافه کرد تا فشار کار کاهش یابد.

همینکار را بر اساس زمان نیز می‌توان انجام داد:

```
var sequence = Enumerable.Range(1, 200)
                  .ToObservable()
                  .Buffer(timeSpan: TimeSpan.FromSeconds(2));
```

در مثال فوق هر 2 ثانیه یکبار، مجموعه‌ای از عناصر به متد onNext ارسال خواهند شد.

متد Window

متد Window نیز دقیقا همان پارامترهای متد بافر را قبول می‌کند. با این تفاوت که هربار، یک توالی obsevable را به متد onNext ارسال می‌کند.
نوع numbers پارامتر onNext، در حین بکارگیری متد بافر در مثال‌های فوق، IList of int است. اما اگر متدهای Buffer را تبدیل به متد Window کنیم، اینبار نوع numbers، معادل IObservable of int خواهد شد.

```
var sequence = Enumerable.Range(1, 200)
                          .ToObservable()
                          .Window(timeSpan: TimeSpan.FromSeconds(2));

sequence.Subscribe(onNext: numbers =>
{
       numbers.Subscribe(onNext: number => Console.WriteLine(number));
});
```

چه زمانی باید از Buffer استفاده کرد و چه زمانی از Window؟

در متد بافر، به ازای هر توالی که به پارامتر onNext ارسال می‌شود، یکبار وهله‌ی جدیدی از توالی مدنظر در حافظه ایجاد و ارسال خواهد شد. در متد Window صرفا اشاره‌گرهایی به این توالی را در اختیار داریم؛ بنابراین مصرف حافظه‌ی کمتری را شاهد خواهیم بود. متد Window بسیار مناسب است برای اعمال aggregation. مثلا اگر نیاز است جمع، میانگین، حداقل و حداکثر عناصر دریافتی محاسبه شوند، بهتر است از متد Window استفاده شود که نهایتا قابلیت استفاده از متدهای الحاقی Sum و Max و Min را به همراه دارد. با این تفاوت که حاصل این‌ها نیز یک IObservable است که باید Subscribe آن‌را برای دریافت نتیجه فراخوانی کرد:

```
var sequence = Enumerable.Range(1, 200)
                          .ToObservable()
                          .Window(10);

sequence.Subscribe(onNext: numbers =>
{
     numbers.Sum().Subscribe(onNext: number => Console.WriteLine(number));
```

```
});
```

در این حالت متد Window، برخلاف متد Buffer، توالی numbers را هربار کش نمی‌کند و به این ترتیب می‌توان به مصرف حافظه‌ی کمتری رسید.

کاربردهای دنیای واقعی

در اینجا دو مثال از بکارگیری متد Buffer را جهت پردازش مجموعه‌های عظیمی از اطلاعات و نمایش همزمان آن‌ها در رابط کاربری ملاحظه می‌کنید.
مثال اول: فرض کنید قصد دارید تمام فایل‌های درایو C خود را توسط یک TreeView نمایش دهید. در این حالت نباید رابط کاربری برنامه در حالت هنگ به نظر برسد. همچنین به علت زیاد بودن تعداد فایل‌ها و نمایش همزمان آن‌ها در UI، نباید CPU Usage برنامه تا حدی باشد که در کار سایر برنامه‌ها اخلال ایجاد کند. در این مثال‌ها با استفاده از Rx و متد بافر آن، هربار مثلا 1000 آیتم را بافر کرده و سپس یکجا در TreeView نمایش می‌دهند. به این ترتیب دو شرط یاد شده محقق می‌شوند.
The Rx Framework By Example

مثال دوم: خواندن تعداد زیادی رکورد از بانک اطلاعاتی به همراه نمایش همزمان آن‌ها در UI بدون اخلالی در کار سیستم و همچنین هنگ کردن برنامه.
Using Reactive Extensions for Streaming Data from Database