عنوان: **مدیریت همزمانی و طول عمر توالی‌های Reactive extensions در یک برنامه‌ی دسکتاپ**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۲/۲۹ ۷:۵
آدرس: www.dotnettips.info
گروه‌ها: Asynchronous Programming, Reactive Extensions, Rx

پس از معرفی و مشاهده‌ی نحوه‌ی ایجاد توالی‌ها در Rx ، بهتر است با نمونه‌ای از نحوه‌ی استفاده از آن در یک برنامه‌ی WPF آشنا شویم.
بنابراین ابتدا دو بسته‌ی Rx-Main و Rx-WPF را توسط نیوگت، به یک برنامه‌ی جدید WPF اضافه کنید:

```
PM> Install-Package Rx-Main
PM> Install-Package Rx-WPF
```

فرض کنید قصد داریم محتوای یک فایل حجیم را به نحو ذیل خوانده و توسط Rx نمایش دهیم.

```
private static IEnumerable<string> readFile(string filename)
        {
            using (TextReader reader = File.OpenText(filename))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    Thread.Sleep(100);
                    yield return line;
                }
            }
        }
```

در اینجا برای ایجاد یک توالی IEnumerable ، از yield return استفاده شده‌است. همچنین Thread.Sleep آن جهت بررسی قفل شدن رابط کاربری در حین خواندن فایل به عمد قرار گرفته است.
UI برنامه نیز به نحو ذیل است:

```
<Window x:Class="WpfApplicationRxTests.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="450" Width="525">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition  Height="*" />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Button Grid.Row="0" Name="btnGenerateSequence" Click="btnGenerateSequence_Click">Generate
sequence</Button>
        <ListBox Grid.Row="1" Name="lstNumbers"  />
        <Button Grid.Row="2" IsEnabled="False" Name="btnStop" Click="btnStop_Click">Stop</Button>
    </Grid>
</Window>
```

با این کدها

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.IO;
using System.Reactive.Concurrency;
using System.Reactive.Linq;
using System.Threading;
using System.Windows;

namespace WpfApplicationRxTests
{
    public partial class MainWindow
    {
        public MainWindow()
        {
            InitializeComponent();
```

```
        }

        private static IEnumerable<string> readFile(string filename)
        {
            using (TextReader reader = File.OpenText(filename))
            {
                string line;
                while ((line = reader.ReadLine()) != null)
                {
                    Thread.Sleep(100);
                    yield return line;
                }
            }
        }

        private IDisposable _subscribe;
        private void btnGenerateSequence_Click(object sender, RoutedEventArgs e)
        {
            btnGenerateSequence.IsEnabled = false;
            btnStop.IsEnabled = true;

            var items = new ObservableCollection<string>();
            lstNumbers.ItemsSource = items;
            _subscribe = readFile("test.txt").ToObservable()
                                .SubscribeOn(ThreadPoolScheduler.Instance)
                                .ObserveOn(DispatcherScheduler.Current)
                                .Finally(finallyAction: () =>
                                {
                                    btnGenerateSequence.IsEnabled = true;
                                    btnStop.IsEnabled = false;
                                })
                                .Subscribe(onNext: line =>
                                {
                                    items.Add(line);
                                },
                                onError: ex => { },
                                onCompleted: () =>
                                {
                                    //lstNumbers.ItemsSource = items;
                                });
        }

        private void btnStop_Click(object sender, RoutedEventArgs e)
        {
            _subscribe.Dispose();
        }
    }
}
```

توضیحات

حاصل متد readFile را که یک توالی معمولی IEnumerable را ایجاد می‌کند، توسط فراخوانی متد ToObservable، تبدیل به یک خروجی IObservable کرده‌ایم تا بتوانیم هربار که سطری از فایل مدنظر خوانده می‌شود، نسبت به آن واکنش نشان دهیم.
متد SubscribeOn مشخص می‌کند که این توالی Observable باید بر روی چه تردی اجرا شود. در اینجا از
ThreadPoolScheduler.Instance استفاده شده‌است تا در حین خواندن فایل، رابط کاربری در حالت هنگ به نظر نرسد و ترد جاری (ترد اصلی برنامه) به صورت خودکار آزاد گردد.
از متد ObserveOn با پارامتر DispatcherScheduler.Current استفاده کرده‌ایم، تا نتیجه‌ی واکنش‌های به خوانده شدن سطرهای یک فایل مفروض، در ترد اصلی برنامه صورت گیرد. در غیر اینصورت امکان کار کردن با عناصر رابط کاربری در یک ترد دیگر وجود نخواهد داشت و برنامه کرش می‌کند.
در قسمت‌های قبل، صرفا متد Subscribe را مشاهده کرده بودید. در اینجا از متد Finally نیز استفاده شده‌است. علت اینجا است که اگر در حین خواندن فایل خطایی رخ دهد، قسمت onError متد Subscribe اجرا شده و دیگر به پارامتر onCompleted آن نخواهیم رسید. اما متد Finally آن همیشه در پایان عملیات اجرا می‌شود.
خروجی حاصل از متد Subscribe، از نوع IDisposable است. Rx به صورت خودکار پس از پردازش آخرین عنصر توالی، این شیء را Dispose می‌کند. اینجا است که callback متد Finally یاد شده فراخوانی خواهد شد. اما اگر در حین خواندن یک فایل طولانی، کاربر علاقمند باشد تا عملیات را متوقف کند، تنها کافی است که به صورت صریح، این شیء را Dispose نماید. به همین جهت است که مشاهده می‌کنید، این خروجی به صورت یک فیلد تعریف شده‌است تا در متد Stop بتوانیم آن‌را در صورت نیاز

Dispose کنیم.

مثال فوق را از اینجا نیز می‌توانید دریافت کنید:

WpfApplicationRxTests.zip