عنوان: **استفاده از Async و Await در برنامه‌های دسکتاپ**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۸ ۱۸:۳۳
آدرس: www.dotnettips.info
گروه‌ها: C#, Asynchronous Programming

امکان استفاده از قابلیت‌های غیرهمزمان دات نت 4.5 در برنامه‌های WPF نیز به روش‌های مختلفی میسر است که در ادامه دو روش مرسوم آن‌را بررسی خواهیم کرد.

تهیه مقدمات بحث

ابتدا یک برنامه‌ی WPF جدید را آغاز کنید. سپس کدهای MainWindow.xaml آن‌را به نحو ذیل تغییر دهید.

```
<Window x:Class="Async10.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <DockPanel>
        <DockPanel Dock="Top">
            <Button Name="BtnGo" Content="Go" Click="BtnGo_OnClick" />
            <ProgressBar Name="ProgressBar" IsIndeterminate="True" Visibility="Collapsed"/>
        </DockPanel>
        <TextBox Name="Results"/>
    </DockPanel>
</Window>
```

قصد داریم اطلاعاتی را از وب دریافت و سپس در TextBox قرار گرفته در صفحه نمایش دهیم.
در این مثال از کلاس جدید HttpClient نیز استفاده خواهیم کرد. برای استفاده از آن نیاز است ارجاعی را به اسمبلی استاندارد System.Net.Http.dll نیز به پروژه اضافه کنید.

روش اول

در ادامه کدهای فایل MainWindow.xaml.cs را به نحو ذیل تغییر داده و سپس برنامه را اجرا کنید.

```
using System.Net.Http;
using System.Windows;

namespace Async10
{
    public partial class MainWindow
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void BtnGo_OnClick(object sender, RoutedEventArgs e)
        {
            BtnGo.IsEnabled = false;
            ProgressBar.Visibility = Visibility.Visible;

            var url = "http://www.dotnettips.info";
            var client = new HttpClient(); // make sure you have an assembly reference to
System.Net.Http.dll
            client.DefaultRequestHeaders.UserAgent.ParseAdd("Test Async");
            var task = client.GetStringAsync(url);
            task.ContinueWith(t =>
            {
                Results.Text = t.Result;

                BtnGo.IsEnabled = true;
                ProgressBar.Visibility = Visibility.Collapsed;
            });
        }
    }
}
```

روال رخدادگردان BtnGo_OnClick به نحو مرسوم آن نوشته شده است. بنابراین جهت دریافت نتیجه‌ی متد GetStringAsync می‌توان از متد ContinueWith بر روی task دریافت اطلاعات از وب، استفاده کرد. همچنین در اینجا مستقیما اطلاعات و نتیجه‌ی دریافتی را به عناصر UI انتساب داده‌ایم.
اگر پروژه را اجرا کنید، برنامه با استثنای زیر متوقف می‌شود:

```
The calling thread cannot access this object because a different thread owns it.
```

چون task آغاز شده در ترد دیگری نسبت به ترد UI اجرا می‌شود، مجوز تغییری را در کدهای UI ندارد. برای حل این مشکل می‌توان از دو روش ذیل استفاده کرد:

الف) با استفاده از SynchronizationContext.Current و متد Post آن

```
var context = SynchronizationContext.Current;
task.ContinueWith(t => context.Post(state =>
{
   Results.Text = t.Result;

   BtnGo.IsEnabled = true;
   ProgressBar.Visibility = Visibility.Collapsed;
}, null));
```

با این روش در قسمت اول آشنا شدید. SynchronizationContext.Current در اینجا چون در ابتدای متد و خارج از ContinueWith دریافت اطلاعات، اجرا می‌شود، به ترد UI یا ترد اصلی برنامه اشاره می‌کند. سپس همانطور که ملاحظه می‌کنید، توسط متد Post آن می‌توان اطلاعات را در زمینه‌ی تردی که SynchronizationContext به آن اشاره می‌کند اجرا کرد.

ب) با استفاده از امکانات TaskScheduler

```
var taskScheduler = TaskScheduler.FromCurrentSynchronizationContext();
task.ContinueWith(t =>
{
   Results.Text = t.Result;

   BtnGo.IsEnabled = true;
   ProgressBar.Visibility = Visibility.Collapsed;
}, taskScheduler);
```

وقتی یک task اجرا می‌شود، TPL یا task parallel library نیاز دارد بداند، این task بر روی چه تردی و چه زمانی قرار است اجرا شود. به صورت پیش فرض از thread pool استفاده می‌کند، اما الزامی به آن نیست. با استفاده از TaskScheduler می‌توان بر روی نحوه‌ی رفتار تردهای TPL تاثیر گذاشت و یا حتی آن‌ها را سفارشی سازی کرد. متد FromCurrentSynchronizationContext، یک TaskScheduler جدید را در اختیار ما قرار می‌دهد که کدهای آن بر اساس SynchronizationContext.Current کار می‌کند؛ در اینجا Context به UI اشاره می‌کند و در یک برنامه‌ی وب، به یک درخواست رسیده.
برای مثال اگر در برنامه‌های وب یک Task جدید را اجرا کنید شاید اینطور به نظر برسد که به HttpContext دسترسی ندارید. این نقیصه را می‌توان توسط کار با SynchronizationContext جاری برطرف کرد.
در مثال فوق، چون taskScheduler پیش از فراخوانی متد ContinueWith ایجاد شده‌است، به ترد UI اشاره می‌کند. در این حالت برای نمایش اطلاعات در همان ترد اصلی برنامه کافی است این taskScheduler را به عنوان پارامتر متد ContinueWith معرفی کنیم.

روش دوم

در دات نت 4.5 می‌توان روال رخدادگردان تعریف شده را به صورت async نیز معرفی کرد (یعنی مجاز هستیم امضای متد پیش فرض تولید شده را تغییر دهیم):

```
private async void BtnGo_OnClick(object sender, RoutedEventArgs e)
```

سپس استفاده از await در کدهای برنامه میسر خواهد شد:

```
private async void BtnGo_OnClick(object sender, RoutedEventArgs e)
        {
            BtnGo.IsEnabled = false;
            ProgressBar.Visibility = Visibility.Visible;

            var url = "http://www.dotnettips.info";
            var client = new HttpClient(); // make sure you have an assembly reference to
System.Net.Http.dll
            client.DefaultRequestHeaders.UserAgent.ParseAdd("Test Async");
            Results.Text = await client.GetStringAsync(url);
            BtnGo.IsEnabled = true;
            ProgressBar.Visibility = Visibility.Collapsed;
        }
```

در این حالت دیگر نیازی به استفاده از ContinueWith و مباحث SynchronizationContext نیست. زیرا تمام آن‌ها به صورت توکار اعمال می‌شوند. به علاوه کدنهایی نیز بسیار خواناتر شده‌است.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۳ ۱۱:۳۰

یک نکته‌ی تکمیلی

اگر از الگوی MVVM استفاده می‌کنید، یک پیاده سازی AsyncCommand را در اینجا می‌توانید ملاحظه کنید:

Patterns for Asynchronous MVVM Applications: Commands