عنوان: **استفاده از async و await در برنامه‌های کنسول و سرویس‌های ویندوز NT**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۱۸ ۰:۵۸
آدرس: www.dotnettips.info
گروه‌ها: C#, Asynchronous Programming

فرض کنید می‌خواهید از await در متد Main یک برنامه‌ی کنسول به نحو ذیل استفاده کنید:

```
using System;
using System.Net;

namespace Async15
{
    class Program
    {
        static void Main(string[] args)
        {
            using (var webClient = new WebClient { })
            {
                webClient.Headers.Add("User-Agent", "AsyncContext 1.0");
                var data = await webClient.DownloadStringTaskAsync("http://www.dotnettips.info");
                Console.WriteLine(data);
            }
        }
    }
}
```

کامپایلر چنین اجازه‌ای را نمی‌دهد. زیرا از await جایی می‌توان استفاده کرد که متد فراخوان آن با async مزین شده باشد و همچنین دارای یک Context باشد تا نتیجه را بتواند دریافت کند. اگر در اینجا سعی کنید async را به امضای متد Main اضافه نمائید، کامپایلر مجددا خطای an entry point cannot be marked with the 'async' modifier را صادر می‌کند.
اضافه کردن واژه‌ی کلیدی async به روال‌های رخدادگردان void برنامه‌های دسکتاپ مجاز است؛ با توجه به اینکه متد async پیش از پایان کار به فراخوان بازگشت داده می‌شوند (ذات متدهای async به این نحو است). در برنامه‌های دسکتاپ، این بازگشت به UI event loop است؛ بنابراین برنامه بدون مشکل به کار خود ادامه خواهد داد. اما در اینجا، بازگشت متد Main، به معنای بازگشت به OS است و خاتمه‌ی برنامه. به همین جهت کامپایلر از async کردن آن ممانعت می‌کند.
برای حل این مشکل در برنامه‌های کنسول و همچنین برنامه‌های سرویس ویندوز NT که دارای یک async-compatible context نیستند، می‌توان از یک کتابخانه‌ی کمکی سورس باز به نام Nito AsyncEx استفاده کرد. برای نصب آن دستور ذیل را در کنسول پاورشل نیوگت وارد کنید:

```
PM> Install-Package Nito.AsyncEx
```

پس از نصب برای استفاده از آن خواهیم داشت:

```
using System;
using System.Net;
using Nito.AsyncEx;

namespace Async15
{
    class Program
    {
        static void Main(string[] args)
        {
            AsyncContext.Run(async () =>
            {
                using (var webClient = new WebClient())
                {
                    webClient.Headers.Add("User-Agent", "AsyncContext 1.0");
                    var data = await webClient.DownloadStringTaskAsync("http://www.dotnettips.info");
                    Console.WriteLine(data);
                }
            });
        }
    }
}
```

Context ارائه شده در اینجا برخلاف مثال‌های قسمت‌های قبل، نیازی به فراخوانی متد همزمان Wait و یا خاصیت Result که هر دو از نوع blocking هستند ندارد و یک فراخوانی async واقعی است. همچنین می‌شد یک متد async void را نیز در اینجا برای استفاده از DownloadStringTaskAsync تعریف کرد (تا برنامه کامپایل شود). اما پیشتر عنوان شد که هدف از این نوع متدهای خاص async void صرفا استفاده از آن‌ها در روال‌های رخدادگردان UI هستند. زیرا ماهیت آن‌ها fire and forget است و برای دریافت نتیجه‌ی نهایی به نحوی باید ترد اصلی را قفل کرد. برای مثال در یک برنامه‌ی کنسول متد Console.ReadLine را در انتهای کار فراخوانی کرد. اما با استفاده از AsyncContext.Run نیازی به این کارها نیست.

async lambda

در مثال فوق از یک async lambda، برای فراخوانی استفاده شده است که به همراه دات نت 4.5 ارائه شده‌اند:

```
Action, () => { }
Func<Task>, async () => { await Task.Yield(); }

Func<TResult>, () => { return 13; }
Func<Task<TResult>>, async () => { await Task.Yield(); return 13; }
```

آرگومان متد AsyncContext.Run از نوع Func of Task است. بنابراین برای مقدار دهی inline آن توسط lambda expressions مطابق مثال‌های فوق می‌توان از async lambda استفاده کرد.

روش دوم استفاده از AsyncContext.Run و مقدار دهی Func of Task، تعریف یک متد مستقل async Task دار، به نحو ذیل است:

```
class Program
{
  static async Task<int> AsyncMain()
  {
    ..
  }

  static int Main(string[] args)
  {
    return AsyncContext.Run(AsyncMain);
  }
}
```

رخدادهای مرتبط با طول عمر برنامه را async تعریف نکنید

همانند متد Main که async تعریف کردن آن سبب بازگشت آنی روال کار به OS می‌شود و برنامه خاتمه می‌یابد، روال‌های رخدادگردانی که با طول عمر یک برنامه‌ی UI سر و کار دارند مانند Application_Launching، Application_Closing، Application_Activated و Application_Deactivated (خصوصا در برنامه‌های ویندوز 8) نیز نباید async void تعریف شوند (چون مطابق ذات متدهای async، بلافاصله به برنامه اعلام می‌کنند که کار تمام شد). در این موارد خاص نیز می‌توان از متد AsyncContext.Run برای انجام اعمال async استفاده کرد.