عنوان: **تبدیل روش‌های قدیمی کدنویسی غیرهمزمان به async سی شارپ 5**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۴ ۱۷:۳۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Asynchronous Programming

در قسمت اول این سری، با مدل برنامه نویسی Event based asynchronous pattern ارائه شده از دات نت 2 و همچنین APM یا Asynchronous programming model موجود از نگارش یک دات نت، آشنا شدیم (به آن الگوی IAsyncResult هم گفته می‌شود).
نکته‌ی مهم این الگوها، استفاده‌ی گسترده از آن‌ها در کدهای کلاس‌های مختلف دات نت فریم ورک است و برای بسیاری از آن‌ها هنوز async API سازگار با نگارش مبتنی بر Taskهای سی‌شارپ 5 ارائه نشده‌است. هرچند دات نت 4.5 سعی کرده‌است این خلاء را پوشش دهد، برای مثال متد الحاقی DownloadStringTaskAsync را به کلاس WebClient اضافه کرده‌است و امثال آن، اما هنوز بسیاری از کلاس‌های دیگر دات نتی هستند که معادل Task based API ایی برای آن‌ها طراحی نشده‌است. در ادامه قصد داریم بررسی کنیم چگونه می‌توان این الگوهای مختلف قدیمی برنامه نویسی غیرهمزمان را با استفاده از روش‌های جدیدتر ارائه شده بکار برد.

نگاشت APM به یک Task

در قسمت اول، نمونه مثالی را از APM، که در آن کار با BeginGetResponse آغاز شده و سپس در callback نهایی توسط EndGetResponse، نتیجه‌ی عملیات به دست می‌آید، مشاهده کردید. در ادامه می‌خواهیم یک محصور کننده‌ی جدید را برای این نوع API قدیمی تهیه کنیم، تا آن‌را به صورت یک Task ارائه دهد.

```
public static class ApmWrapper
    {
        public static Task<int> ReadAsync(this Stream stream, byte[] data, int offset, int count)
        {
            return Task<int>.Factory.FromAsync(stream.BeginRead, stream.EndRead, data, offset, count,
null);
        }
    }
```

همانطور که در این مثال مشاهده می‌کنید، یک چنین سناریوهایی در TPL یا کتابخانه‌ی Task parallel library پیش بینی شده‌اند.
در اینجا یک محصور کننده برای متدهای BeginRead و EndRead کلاس Stream دات نت ارائه شده‌است. به عمد نیز به صورت یک متد الحاقی تهیه شده‌است تا در حین استفاده از آن اینطور به نظر برسد که واقعا کلاس Stream دارای یک چنین متد Async ایی است. مابقی کار توسط متد Task.Factory.FromAsync انجام می‌شود. متد FromAsync دارای امضاهای متعددی است تا اکثر حالات APM را پوشش دهد.
در مثال فوق BeginRead و EndRead استفاده شده از نوع delegate هستند. چون خروجی EndRead از نوع int است، خروجی متد نیز از نوع Task of int تعیین شده‌است. همچنین سه پارامتر ابتدایی BeginRead ، دقیقا data، offset و count هستند. دو پارامتر آخر آن callback و state نام دارند. پارامتر callback توسط متد FromAsync فراهم می‌شود و state نیز در اینجا null درنظر گرفته شده‌است.
یک مثال استفاده از آن‌را در ادامه مشاهده می‌کنید:

```
using System;
using System.IO;
using System.Threading.Tasks;

namespace Async06
{
    public static class ApmWrapper
    {
        public static Task<int> ReadAsync(this Stream stream, byte[] data, int offset, int count)
        {
            return Task<int>.Factory.FromAsync(stream.BeginRead, stream.EndRead, data, offset, count,
null);
        }
    }

    class Program
    {
        static void Main(string[] args)
```

```
        {
            using (var stream = File.OpenRead(@"..\..\program.cs"))
            {
                var data = new byte[10000];
                var task = stream.ReadAsync(data, 0, data.Length);
                Console.WriteLine("Read bytes: {0}", task.Result);
            }
        }
    }
}
```

File.OpenRead، خروجی از نوع استریم دارد. سپس متد الحاقی ReadAsync بر روی آن فراخوانی شده‌است و نهایتا تعداد بایت خوانده شده نمایش داده می‌شود.
البته همانطور که پیشتر نیز عنوان شد، استفاده از خاصیت Result، اجرای کد را بجای غیرهمزمان بودن، به حالت همزمان تبدیل می‌کند.
در اینجا چون خروجی متد ReadAsync یک Task است، می‌توان از متد ContinueWith نیز بر روی آن جهت دریافت نتیجه استفاده کرد:

```
using (var stream = File.OpenRead(@"..\..\program.cs"))
{
    var data = new byte[10000];
    var task = stream.ReadAsync(data, 0, data.Length);
    task.ContinueWith(t => Console.WriteLine("Read bytes: {0}", t.Result)).Wait();
}
```

یک نکته

پروژه‌ی سورس بازی به نام Async Generator در GitHub، سعی کرده‌است برای ساده سازی نوشتن محصور کننده‌های مبتنی بر Task روش APM، یک Code generator تولید کند. فایل‌های آن‌را از آدرس ذیل می‌توانید دریافت کنید:

https://github.com/chaliy/async-generator

نگاشت EAP به یک Task

نمونه‌ای از Event based asynchronous pattern یا EAP را در قسمت اول، زمانیکه روال رخدادگردان webClient.DownloadStringCompleted را بررسی کردیم، مشاهده نمودید. کار کردن با آن نسبت به APM بسیار ساده‌تر است و نتیجه‌ی نهایی عملیات غیرهمزمان را در یک روال رخدادگران، در اختیار استفاده کننده قرار می‌دهد. همچنین در روش EAP، اطلاعات در همان Synchronization Context ایی که عملیات شروع شده‌است، بازگشت داده می‌شود. به این ترتیب اگر آغاز کار در ترد UI باشد، نتیجه نیز در همان ترد دریافت خواهد شد. به این ترتیب دیگر نگران دسترسی به مقدار آن در کارهای UI نخواهیم بود؛ اما در APM چنین ضمانتی وجود ندارد.
متاسفانه TPL همانند روش FromAsync معرفی شده در ابتدای بحث، راه حل توکاری را برای محصور سازی متدهای روش EAP ارائه نداده‌است. اما با استفاده از امکانات TaskCompletionSource آن می‌توان چنین کاری را انجام داد. در ادامه سعی خواهیم کرد همان متد الحاقی توکار DownloadStringTaskAsync ارائه شده در دات نت 4.5 را از صفر بازنویسی کنیم.

```
public static class WebClientExtensions
    {
        public static Task<string> DownloadTextTaskAsync(this WebClient web, string url)
        {
            var tcs = new TaskCompletionSource<string>();

            DownloadStringCompletedEventHandler handler = null;
            handler = (sender, args) =>
            {
                web.DownloadStringCompleted -= handler;

                if (args.Cancelled)
                {
                    tcs.SetCanceled();
                }
                else if(args.Error!=null)
```

```
                {
                    tcs.SetException(args.Error);
                }
                else
                {
                    tcs.SetResult(args.Result);
                }
            };

            web.DownloadStringCompleted += handler;
            web.DownloadStringAsync(new Uri(url));

            return tcs.Task;
        }
    }
```

روش انجام کار را در اینجا ملاحظه می‌کنید. ابتدا باید تعاریف delaget مرتبط با رخدادگردان Completed اضافه شوند. یکبار += را ملاحظه می‌کنید و بار دوم -= را. مورد دوم جهت آزاد سازی منابع و جلوگیری از نشتی حافظه‌ی روال رخدادگردان هنوز متصل، ضروری است.

سپس از TaskCompletionSource برای تبدیل این عملیات به یک Task کمک می‌گیریم. اگر args.Cancelled مساوی true باشد، یعنی عملیات دریافت فایل لغو شده‌است. بنابراین متد SetCanceled منبع Task ایجاد شده را فراخوانی خواهیم کرد. این مورد استثنایی را در کدهای فراخوان سبب می‌شود. به همین دلیل بررسی خطا با یک if else پس از آن انجام شده‌است. برای بازگشت خطای دریافت شده از متد SetException و برای بازگشت نتیجه‌ی واقعی دریافتی، از متد SetResult می‌توان استفاده کرد.

به این ترتیب متد الحاقی غیرهمزمان جدیدی را به نام DownloadTextTaskAsync برای محصور سازی متد EAP ایی به نام DownloadStringAsync و همچنین رخدادگران آن تهیه کردیم.