عنوان: **لغو اعمال غیرهمزمان**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۰۱/۰۶ ۱۵:۵۰
آدرس: www.dotnettips.info
گروه‌ها: C#, Asynchronous Programming

دات نت 4.5 روش عمومی را جهت لغو اعمال غیرهمزمان طولانی اضافه کرده‌است. برای مثال اگر نیاز است تا چندین عمل با هم انجام شوند تا کار مشخصی صورت گیرد و یکی از آن‌ها با شکست مواجه شود، ادامه‌ی عملیات با سایر وظایف تعریف شده، بی‌حاصل است. لغو اعمال در برنامه‌های دارای رابط کاربری نیز حائز اهمیت است. برای مثال یک کاربر ممکن است تصمیم بگیرد تا عملیاتی طولانی را لغو کند.

مدل لغو اعمال

پایه لغو اعمال، توسط مکانیزمی به نام CancellationToken پیاده سازی شده‌است و آن‌را به عنوان یکی از آرگومان‌های متدهایی که لغو اعمال را پشتیبانی می‌کنند، مشاهده خواهید کرد. به این ترتیب یک عمل خاص می‌تواند دریابد چه زمانی لغو آن درخواست شده‌است. البته باید دقت داشت که این عملیات بر مبنای ایده‌ی همه یا هیچ است. به این معنا که یک درخواست لغو را بار دیگر نمی‌توان لغو کرد.

یک مثال استفاده از CancellationToken

کدهای زیر، یک فایل حجیم را از مکانی به مکانی دیگر کپی می‌کنند. برای این منظور از متد CopyToAsync که در دات نت 4.5 اضافه شده‌است، استفاده کرده‌ایم؛ زیرا از مکانیزم لغو عملیات پشتیبانی می‌کند.

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

namespace Async08
{
    class Program
    {
        static void Main(string[] args)
        {
            var source = @"c:\dir\file.bin";
            var target = @"d:\dir\file.bin";
            using (var inStream = File.OpenRead(source))
            {
                using (var outStream = File.OpenWrite(target))
                {
                    using (var cts = new CancellationTokenSource())
                    {
                        var task = inStream.CopyToAsync(outStream, bufferSize: 4059, cancellationToken:
cts.Token);
                        Console.WriteLine("Press 'c' to cancel.");
                        var key = Console.ReadKey().KeyChar;
                        if (key == 'c')
                        {
                            Console.WriteLine("Cancelling");
                            cts.Cancel();
                        }
                        Console.WriteLine("Wating...");
                        task.ContinueWith(t => { }).Wait();
                        Console.WriteLine("Status: {0}", task.Status);
                    }
                }
            }
        }
    }
}
```

کار با تعریف CancellationTokenSource شروع می‌شود. چون از نوع IDisposable است، نیاز است توسط عبارت using، جهت پاکسازی منابع آن، محصور گردد. سپس در اینجا اگر کاربر کلید c را فشار دهد، متد لغو توکن تعریف شده فراخوانی خواهد شد. این توکن نیز به عنوان آرگومان به متد CopyToAsync ارسال شده‌است.

علت استفاده از ContinueWith در اینجا این است که اگر یک task لغو شود، فراخوانی متد Wait بر روی آن سبب بروز استثناء می‌گردد. به همین جهت توسط ContinueWith یک Task خالی ایجاد شده و سپس بر روی آن Wait فراخوانی گردیده‌است.
همچنین باید دقت داشت که سازنده‌ی CancellationTokenSource امکان دریافت زمان timeout عملیات را نیز دارد. به علاوه متد CancelAfter نیز برای آن طراحی شده‌است. نمونه‌ی دیگری از تنظیم timeout را در قسمت قبل با معرفی متد Task.Delay و استفاده از آن با Task.WhenAny مشاهده کردید.

لغو ظاهری وظایفی که لغو پذیر نیستند

فرض کنید متدی به نام GetBitmapAsync با پارامتر cancellationToken طراحی نشده‌است. در این حالت کاربر قصد دارد با کلیک بر روی دکمه‌ی لغو، عملیات را خاتمه دهد. یک روش حل این مساله، استفاده از متد ذیل است:

```
public static class CancellationTokenExtensions
    {
        public static async Task UntilCompletionOrCancellation(Task asyncOp, CancellationToken ct)
        {
            var tcs = new TaskCompletionSource<bool>();
            using (ct.Register(() => tcs.TrySetResult(true)))
            {
                await Task.WhenAny(asyncOp, tcs.Task);
            }
        }
    }
```

در اینجا از روش Task.WhenAny استفاده شده‌است که در آن دو task ترکیب شده‌اند. Task اول همان وظیفه‌ای اصلی است و task دوم، از یک TaskCompletionSource حاصل شده‌است. اگر کاربر دستور لغو را صادر کند، callback ثبت شده توسط این توکن، اجرا خواهد شد. بنابراین در اینجا TrySetResult به true تنظیم شده و یکی از دو Task معرفی شده در WhenAny خاتمه می‌یابد.
این مورد هر چند task اول را واقعا لغو نمی‌کند، اما سبب خواهد شد تا کدهای پس از await UntilCompletionOrCancellation اجرا شوند.

طراحی متدهای غیرهمزمان لغو پذیر

کلاس زیر را در نظر بگیرید:

```
public class CancellationTokenTest
    {
        public static void Run()
        {
            var cts = new CancellationTokenSource();
            Task.Run(async () => await test(), cts.Token);
            Console.ReadLine();
            cts.Cancel();
            Console.WriteLine("Cancel...");
            Console.ReadLine();
        }

        private static async Task test()
        {
            while (true)
            {
                await Task.Delay(1000);
                Console.WriteLine("Test...");
            }
        }
    }
```

در اینجا cancellationToken متد Task.Run تنظیم شده‌است. همچنین پس از فراخوانی آن، اگر کاربر کلیدی را فشار دهد، متد Cancel این توکن فراخوانی خواهد شد. اما .... خروجی برنامه به صورت زیر است:

```
Test...
```

```
Test...
Test...

Cancel...
Test...
Test...
Test...
Test...
```

بله. وظیفه‌ی شروع شده، لغو شده‌است اما متد test آن هنوز مشغول به کار است.
روش اول حل این مشکل، معرفی پارامتر CancellationToken به متد test و سپس بررسی مداوم خاصیت IsCancellationRequested آن می‌باشد:

```
public class CancellationTokenTest
    {
        public static void Run()
        {
            var cts = new CancellationTokenSource();
            Task.Run(async () => await test(cts.Token), cts.Token);
            Console.ReadLine();
            cts.Cancel();
            Console.WriteLine("Cancel...");
            Console.ReadLine();
        }

        private static async Task test(CancellationToken ct)
        {
            while (true)
            {
                await Task.Delay(1000, ct);
                Console.WriteLine("Test...");

                if (ct.IsCancellationRequested)
                {
                    break;
                }
            }
            Console.WriteLine("Test cancelled");
        }
    }
```

در اینجا اگر متد cts.Cancel فراخوانی شود، مقدار خاصیت ct.IsCancellationRequested مساوی true شده و حلقه خاتمه می‌یابد.
روش دوم لغو عملیات، استفاده از متد Register است. هر زمان که توکن لغو شود، callback آن فراخوانی خواهد شد:

```
private static async Task test2(CancellationToken ct)
        {
            bool isRunning = true;

            ct.Register(() =>
            {
                isRunning = false;
                Console.WriteLine("Query cancelled");
            });

            while (isRunning)
            {
                await Task.Delay(1000, ct);
                Console.WriteLine("Test...");
            }
            Console.WriteLine("Test cancelled");
        }
```

این روش خصوصا برای حالت‌هایی مفید است که در آن‌ها از متدهایی استفاده می‌شود که خودشان امکان لغو شدن را نیز دارند. به این ترتیب دیگر نیازی نیست مدام بررسی کرد که آیا مقدار IsCancellationRequested مساوی true شده‌است یا خیر. هر زمان که callback ثبت شده در متد Register فراخوانی شد، یعنی عملیات باید خاتمه یابد.