عنوان: **Lambda Syntax و کارآیی**
نویسنده: وحید محمّدطاهری
تاریخ: ۱۳۹۴/۰۸/۱۰ ۱۲:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Performance

در این مطلب می‌خواهیم کارآیی event handlers پیاده سازی شده با روش‌های متفاوتی را مورد بررسی قراردهیم.
به مثال زیر توجه کنید:

```
class EventSource : System.Progress<int>
{
    public async System.Threading.Tasks.Task<int> PerformExpensiveCalculation()
    {
        var sum = 0;
        for (var i = 0; i < 100; i++)
        {
            await System.Threading.Tasks.Task.Delay(100);
            sum += i;
            this.OnReport(sum);
        }
        return sum;
    }
}

static class Program
{
    static void Main(string[] args)
    {
        var source = new EventSource();
        System.EventHandler<int> handler = (_, progress) => System.Console.WriteLine(progress);
        source.ProgressChanged += handler;
        System.Console.WriteLine(source.PerformExpensiveCalculation().Result);
        source.ProgressChanged -= handler;

        source.ProgressChanged += ProgressChangedMethod;
        System.Console.WriteLine(source.PerformExpensiveCalculation().Result);
        source.ProgressChanged -= ProgressChangedMethod;
    }

    private static void ProgressChangedMethod( object sender, int e )
    {
        System.Console.WriteLine(e);
    }
}
```

در مثال بالا دو نسخه‌ی مختلف از event handler را با دو روش، (روش اول) با استفاده از Lambda syntax و (روش دوم) با استفاده از یک متد، به صورت جدا تعریف شده، پیاده سازی کرده‌ایم.
خوب؛ برای اندازه گیری کارآیی این دو روش باید کمی فکر کنیم که چه چیزی کارآیی این دو روش را تغییر می‌دهد؟ آیا پردازش event با اضافه کردن و حذف کردن event handler؟ و یا پردازش درون event باعث تغییر در کارآیی می‌شود؟ این، سوال مهمی در تست کارآیی این دو روش مختلف است. اگر پردازش درون event باعث ایجاد تفاوت کارآیی می‌شود، با استفاده از این برنامه می‌توان آن را اندازه گیری کرد. با این حال اگر تفاوت کارآیی با اضافه کردن و حذف کردن event handler اتفاق می‌افتد، با این برنامه بعید است بتوان این روش را تست کرد چرا که فقط یکبار این عمل انجام می‌شود.
قبل از شروع به اندازه گیری کارآیی این دو روش، اجازه بدهید ابتدا به کد IL آن‌ها نگاهی کنیم. (روش اول با استفاده از Lambda syntax)

```
IL_0007:  ldsfld     class [mscorlib]System.EventHandler`1<int32>
LambdaPerformance.Program/'<>c'::'<>9__0_0'
IL_000c:  dup
IL_000d:  brtrue.s   IL_0026
IL_000f:  pop
IL_0010:  ldsfld     class LambdaPerformance.Program/'<>c' LambdaPerformance.Program/'<>c'::'<>9'
IL_0015:  ldftn      instance void LambdaPerformance.Program/'<>c'::'<Main>b__0_0'(object, int32)
IL_001b:  newobj     instance void class [mscorlib]System.EventHandler`1<int32>::.ctor(object, native
int)
IL_0020:  dup
IL_0021:  stsfld     class [mscorlib]System.EventHandler`1<int32>
LambdaPerformance.Program/'<>c'::'<>9__0_0'
IL_0026:  stloc.1
IL_0027:  ldloc.0
```

```
IL_0028:  ldloc.1
IL_0029:  callvirt   instance void class [mscorlib]System.Progress`1<int32>::add_ProgressChanged(class
[mscorlib]System.EventHandler`1<!0>)
IL_002e:  nop
IL_002f:  ldloc.0
IL_0030:  callvirt   instance class [mscorlib]System.Threading.Tasks.Task`1<int32>
LambdaPerformance.EventSource::PerformExpensiveCalculation()
IL_0035:  callvirt   instance !0 class [mscorlib]System.Threading.Tasks.Task`1<int32>::get_Result()
IL_003a:  call       void [mscorlib]System.Console::WriteLine(int32)
IL_003f:  nop
IL_0040:  ldloc.0
IL_0041:  ldloc.1
IL_0042:  callvirt   instance void class
[mscorlib]System.Progress`1<int32>::remove_ProgressChanged(class [mscorlib]System.EventHandler`1<!0>)
```

در بالا 5 دستورالعمل برای اضافه کردن event handler وجود دارد (از IL_0010 تا IL_0029) و یک دستور برای حذف handler وجود دارد (IL_0042).

قبل از شروع مقایسه، کد IL روش دوم را نیز بررسی می‌کنیم:

```
IL_004a:  ldftn      void LambdaPerformance.Program::ProgressChangedMethod(object, int32)
IL_0050:  newobj     instance void class [mscorlib]System.EventHandler`1<int32>::.ctor(object, native
int)
IL_0055:  callvirt   instance void class [mscorlib]System.Progress`1<int32>::add_ProgressChanged(class
[mscorlib]System.EventHandler`1<!0>)
IL_005a:  nop
IL_005b:  ldloc.0
IL_005c:  callvirt   instance class [mscorlib]System.Threading.Tasks.Task`1<int32>
LambdaPerformance.EventSource::PerformExpensiveCalculation()
IL_0061:  callvirt   instance !0 class [mscorlib]System.Threading.Tasks.Task`1<int32>::get_Result()
IL_0066:  call       void [mscorlib]System.Console::WriteLine(int32)
IL_006b:  nop
IL_006c:  ldloc.0
IL_006d:  ldnull
IL_006e:  ldftn      void LambdaPerformance.Program::ProgressChangedMethod(object, int32)
IL_0074:  newobj     instance void class [mscorlib]System.EventHandler`1<int32>::.ctor(object, native
int)
IL_0079:  callvirt   instance void class
[mscorlib]System.Progress`1<int32>::remove_ProgressChanged(class [mscorlib]System.EventHandler`1<!0>)
```

همانطور که مشاهده می‌کنید در روش دوم برای اضافه کردن event handler تنها 3 خط وجود دارند (IL_004a تا IL_0055) و برای حذف کردن آن نیز 3 خط وجود دارند (IL_006e تا IL_0079).

برای اندازه گیری دقیق، برنامه‌ی بالا را کمی تغییر می‌دهیم. ما میزان اضافه و حذف شدن event handler را می‌خواهیم اندازه‌گیری کنیم و کاری به زمان اجرای یک عملیات نداریم. بنابراین فراخوانی PerformExpensiveCalculation() را comment کرده و به صورت خیلی ساده فقط handler را اضافه و حذف می‌کنیم.

```
static class Program
{
    static void Main(string[] args)
    {
        for (var repeats = 10; repeats <= 1000000; repeats *= 10)
        {
            VersionOne(repeats);
            VersionTwo(repeats);
        }
    }

    private static void VersionOne(int repeats)
    {
        var timer = new System.Diagnostics.Stopwatch();
        timer.Start();

        var source = new EventSource();
        for (var i = 0; i < repeats; i++)
        {
            System.EventHandler<int> handler = (_, progress) => System.Console.WriteLine(progress);
            source.ProgressChanged += handler;
            // Console.WriteLine(source.PerformExpensiveCalculation().Result);
            source.ProgressChanged -= handler;
        }
```

```
        timer.Stop();

        System.Console.WriteLine($"Version one: {repeats} add/remove takes
{timer.ElapsedMilliseconds}ms");
    }

    private static void VersionTwo(int repeats)
    {
        var timer = new System.Diagnostics.Stopwatch();
        timer.Start();

        var source = new EventSource();
        for (var i = 0; i < repeats; i++)
        {
            source.ProgressChanged += ProgressChangedMethod;
            // Console.WriteLine(source.PerformExpensiveCalculation().Result);
            source.ProgressChanged -= ProgressChangedMethod;
        }

        timer.Stop();

        System.Console.WriteLine($"Version two: {repeats} add/remove takes
{timer.ElapsedMilliseconds}ms");
    }

    private static void ProgressChangedMethod(object sender, int e)
    {
        System.Console.WriteLine(e);
    }
}
```

و چنین خروجی را تولید می‌کند (البته نسبت به سرعت CPU این زمان‌ها متفاوت خواهد بود)

```
Version one: 10 add/remove takes 0ms
Version two: 10 add/remove takes 0ms
Version one: 100 add/remove takes 0ms
Version two: 100 add/remove takes 0ms
Version one: 1000 add/remove takes 0ms
Version two: 1000 add/remove takes 0ms
Version one: 10000 add/remove takes 0ms
Version two: 10000 add/remove takes 1ms
Version one: 100000 add/remove takes 8ms
Version two: 100000 add/remove takes 13ms
Version one: 1000000 add/remove takes 93ms
Version two: 1000000 add/remove takes 121ms
```

خوب؛ اگر در یک اجرای برنامه، شما یک میلیون بار event handler را اضافه و حذف کنید، 28ms می‌توانید صرفه جویی کنید (در روش اول).

توجه: اگر در برنامه‌ی شما یک میلیون بار event handler اضافه و حذف می‌شوند، نیاز به بازنگری مجدد در طراحی کلی برنامه تان دارد.

یک اشتباه بزرگ با ایجاد یک تغییر در روش اول (Lambda syntax)، ممکن است تاثیر بسیار زیادی را در عملکرد برنامه مشاهده کنید:

```
private static void VersionOne(int repeats)
{
    var timer = new System.Diagnostics.Stopwatch();
    timer.Start();

    var source = new EventSource();
    for (var i = 0; i < repeats; i++)
    {
        // System.EventHandler<int> handler = (_, progress) => System.Console.WriteLine(progress);
        source.ProgressChanged += (_, progress) => System.Console.WriteLine(progress);
        // Console.WriteLine(source.PerformExpensiveCalculation().Result);
        source.ProgressChanged -= (_, progress) => System.Console.WriteLine(progress);
    }
```

```
    timer.Stop();

    System.Console.WriteLine($"Version one: {repeats} add/remove takes {timer.ElapsedMilliseconds}ms");
}
```

به جای تعریف یک متغیر محلی برای عبارت Lambda، دستور اضافه و حذف کردن event handler را به صورت inline استفاده کردیم. خروجی این روش به صورت زیر می‌شود:

```
Version one: 10 add/remove takes 0ms
Version two: 10 add/remove takes 0ms
Version one: 100 add/remove takes 1ms
Version two: 100 add/remove takes 0ms
Version one: 1000 add/remove takes 102ms
Version two: 1000 add/remove takes 0ms
Version one: 10000 add/remove takes 10509ms
Version two: 10000 add/remove takes 1ms
Version one: 100000 add/remove takes 1039014ms
Version two: 100000 add/remove takes 11ms
```

همانطور که مشاهده می‌کنید، روش اول خیلی خیلی آهسته است. توجه کنید من بعد از یکصد هزار بار اضافه و حذف کردن handler، به دلیل طولانی شدن، عملیات را قطع کردم. خب دلیل این همه کندی چیست؟ بیایید نگاهی به کد IL درون حلقه‌ی روش اول بیاندازیم.

```
IL_0018:  nop
  IL_0019:  ldloc.1
  IL_001a:  ldsfld     class [mscorlib]System.EventHandler`1<int32>
LambdaPerformance.Program/'<>c'::'<>9__1_0'
  IL_001f:  dup
  IL_0020:  brtrue.s   IL_0039
  IL_0022:  pop
  IL_0023:  ldsfld     class LambdaPerformance.Program/'<>c' LambdaPerformance.Program/'<>c'::'<>9'
  IL_0028:  ldftn      instance void LambdaPerformance.Program/'<>c'::'<VersionOne>b__1_0'(object,
int32)
  IL_002e:  newobj     instance void class [mscorlib]System.EventHandler`1<int32>::.ctor(object, native
int)
  IL_0033:  dup
  IL_0034:  stsfld     class [mscorlib]System.EventHandler`1<int32>
LambdaPerformance.Program/'<>c'::'<>9__1_0'
  IL_0039:  callvirt   instance void class
[mscorlib]System.Progress`1<int32>::add_ProgressChanged(class [mscorlib]System.EventHandler`1<!0>)
  IL_003e:  nop
  IL_003f:  ldloc.1
  IL_0040:  ldsfld     class [mscorlib]System.EventHandler`1<int32>
LambdaPerformance.Program/'<>c'::'<>9__1_1'
  IL_0045:  dup
  IL_0046:  brtrue.s   IL_005f
  IL_0048:  pop
  IL_0049:  ldsfld     class LambdaPerformance.Program/'<>c' LambdaPerformance.Program/'<>c'::'<>9'
  IL_004e:  ldftn      instance void LambdaPerformance.Program/'<>c'::'<VersionOne>b__1_1'(object,
int32)
  IL_0054:  newobj     instance void class [mscorlib]System.EventHandler`1<int32>::.ctor(object, native
int)
  IL_0059:  dup
  IL_005a:  stsfld     class [mscorlib]System.EventHandler`1<int32>
LambdaPerformance.Program/'<>c'::'<>9__1_1'
  IL_005f:  callvirt   instance void class
[mscorlib]System.Progress`1<int32>::remove_ProgressChanged(class [mscorlib]System.EventHandler`1<!0>)
  IL_0064:  nop
  IL_0065:  nop
  IL_0066:  ldloc.2
  IL_0067:  stloc.3
  IL_0068:  ldloc.3
  IL_0069:  ldc.i4.1
  IL_006a:  add
  IL_006b:  stloc.2
  IL_006c:  ldloc.2
  IL_006d:  ldarg.0
  IL_006e:  clt
  IL_0070:  stloc.s    V_4
  IL_0072:  ldloc.s    V_4
  IL_0074:  brtrue.s   IL_0018
```

به خطهای (IL_0028 و IL_0034 و IL_004e و IL_005a) در کد بالا دقت کنید. توجه داشته باشید که event handler اضافه شده با event handler حذف شده، با هم متفاوت هستند. حذف کردن event handler ای که به جایی متصل نیست باعث ایجاد خطا نمی‌شود ولی کاری هم انجام نمی‌دهد. بنابراین اتفاقی که در روش اول درون حلقه می‌افتد این است که بیش از یک میلیون بار event handler به delegate اضافه می‌شود. همه‌ی آنها یکسان هستند؛ اما همچنان CPU و حافظه مصرف می‌کنند.

ممکن است شما به این نتیجه رسیده باشید که استفاده از Lambda syntax برای اضافه و حذف کردن event handler آهسته‌تر از، استفاده از متد جدا است، این یک *اشتباه بزرگ* است. در صورتی که شما اضافه و حذف کردن event handler را با استفاده از Lambda syntax به شکل صحیح انجام ندهید، به سرعت، در معیارهای کارآیی خود را نشان می‌دهد.

دانلود برنامه بالا