عنوان: .NET Just-In-Time Optimization
نویسنده: غلامرضا ربال
تاریخ: ۱۳۹۴/۰۱/۰۵ ۱۰:۳۰
آدرس: www.dotnettips.info
گروه‌ها: Performance, JIT, .NET 4.5

هدف از توابع خطی(Inline)

استفاده از توابع، مقداری بر زمان اجرای برنامه می‌افزاید؛ هرچند که این زمان بسیار کم و در حد میلی ثانیه است، اما باری را بر روی برنامه قرار می‌دهد و علت این تاخیر زمانی این است که در فراخوانی و اعلان توابع، کامپایلر یک کپی از تابع مورد نظر را در حافظه قرار می‌دهد و در فراخوانی تابع، به آدرس مذکور مراجعه می‌کند و در عین حال آدرس موقعیت توقف دستورات در تابع main را نیز ذخیره می‌کند تا پس از پایان تابع، به آدرس قبل برگردد و ادامه‌ی دستورات را اجرا کند. در نتیجه این آدرس دهی‌ها و نقل و انتقالات بین آنها بار زمانی را در برنامه ایجاد می‌کند که در صورت زیاد بودن توابع در برنامه و تعداد فراخوانی‌های لازم، زمان قابل توجهی خواهد شد.

یکی از تکنیک‌های بهینه که برای کاهش زمان اجرای برنامه توسط کامپایلرها استفاده می‌شود استفاده از توابع خطی (inline) است. این امکان در زبان C با عنوان توابع ماکرو(Macro function) و در C++ با عنوان توابع خطی (inline function) وجود دارد.

در واقع توابع خطی به کامپایلر پیشنهاد می‌دهند، زمانی که سربار فراخوانی تابع بیشتر از سربار بدنه خود متد باشد، برای کاهش هزینه و زمان اجرای برنامه از تابع به صورت خطی استفاده کند و یک کپی از بنده‌ی تابع را در قسمتی که تابع ما فراخوانی شده است، قرار دهد که مورد آدرس دهی از میان خواهد رفت!

نمونه ای از پیاده سازی این تکنیک در زبان C++ :

```
inline type name(parameters)
{
    ...
}
```

بررسی متدهای خطی در سی شارپ به مثال زیر توجه کنید:

قسمت‌های getter و setter مربوط به پراپرتی‌ها سربار اضافی بر کلاس Vector می‌افزایند. این موضوع شاید آنچنان مسئله‌ی مهمی نباشد. ولی فرض کنید این پراپرتی‌ها به شکل زیر داخل حلقه‌ای طولانی قرار گیرند. اگر با استفاده از یک پروفایلر زمان اجرای برنامه را زیر نظر بگیرید، خواهید دید که بیش از 90 درصد آن صرف فراخوانی‌های متدهای بخش‌های get , set پراپرتی‌ها است. برای این منظور باید مطمئن شویم که فراخوانی این متدها، به صورت خطی صورت می‌گیرد!

```
public class Vector
  {
    public double X { get; set; }
    public double Y { get; set; }
    public double Z { get; set; }

    // ...
  }
```

برای این منظور آزمایشی را انجام می‌دهیم. فرض کنید کلاسی را به شکل زیر داشته باشیم:

```
public class MyClass
  {
    public int A { get; set; }
    public int C;
  }
```

و برای استفاده از آن به شکل زیر عمل کنیم:

```
static void Main()
{
    MyClass target = new MyClass();
    int a = target.A;
    Console.WriteLine("A = {0}", a);
    int c = target.C;
```

```
    Console.WriteLine("C = {0}", c);
  }
```

بعد از دیباگ برنامه و مشاهده‌ی کدهای ماشین مربوط به آن خواهیم دید که متد مربوط به getter پراپرتی A به صورت خطی فراخوانی نشده است:

```
int a = target.A;
  0000003e  mov         ecx,edi
  00000040  cmp         dword ptr [ecx],ecx
  00000042  call        dword ptr ds:[05FA29A8h]
  00000048  mov         esi,eax
  0000004a  mov         dword ptr [esp+4],esi
             int c = target.C;
  00000098  mov         edi,dword ptr [edi+4]
    MyClass.get_A() looks like this:
  00000000  push        esi
  00000001  mov         esi,ecx
  00000003  cmp         dword ptr ds:[03B701DCh],0
  0000000a  je          00000011
  0000000c  call        76BA6BA7
  00000011  mov         eax,dword ptr [esi+0Ch]
  00000014  pop         esi
  00000015  ret
```

چه اتفاقی افتاده است؟

کامپایلر سی شارپ در زمان کامپایل، کدهای برنامه را به کدهای IL تبدیل می‌کند و JITکامپایلر، این کدهای IL را گرفته و کد ساده‌ی ماشین را تولید می‌کند. لذا به دلیل اینکه JIT با معماری پردازنده آشنایی کافی دارد، مسئولیت تصمیم گیری اینکه کدام متد به صورت خطی فراخوانی شود برعهده‌ی آن است. در واقع این JIT است که تشخیص می‌دهد که آیا فراخونی متد به صورت خطی مناسب است یا نه و به صورت یک معاوضه کار بین خط لوله دستورالعمل‌ها و کش است.

اگر شما برنامه‌ی خود را با (F5) و همگام با دیباگ اجرا کنید، تمام بهینه سازی‌های JIT که Inline Method هم یکی از آنهاست، از کار خواهند افتاد. برای مشاهده‌ی کد بهینه شده باید با بدون دیباگ (CTRL+F5) برنامه خود را اجرا کنید که در آن صورت مشاهده خواهید کرد، متد getter مربوط به پراپرتی A به صورت خطی استفاده شده است.

```
int a = target.A;
00000024  mov         ebx,dword ptr [edi+0Ch]
```

JIT محدودیت هایی برای فراخونی به صورت خطی متدها دارد :

متد هایی که حجم کد IL آنها بیشتر از 32 بایت است.

متدهای بازگشتی.

متدهایی که با اتریبیوت MethodImpl علامتگذاری شدند و MethodImplOptions.NoInlining اعمال شده بر آن

متدهای virtual

متدهایی که دارای کد مدیریت خطا هستند

Methods that take a large value type as a parameter

Methods with complicated flowgraphs

برای اینکه در سی شارپ به کامپایلر اعلام کنیم تا متد مورد نظر به صورت خطی مورد استفاده قرار گیرد، در دات نت 4.5 توسط اتریبیوت MethodImpl و اعمال MethodImplOptions.AggressiveInlining که یک نوع شمارشی است می‌توان این کار را انجام داد. مثال:

```
using System;
using System.Diagnostics;
using System.Runtime.CompilerServices;
class Program
{
    const int _max = 10000000;
    static void Main()
```

```
    {
         // ... Compile the methods.
        Method1();
        Method2();
        int sum = 0;
        var s1 = Stopwatch.StartNew();
        for (int i = 0; i < _max; i++)
           {
              sum += Method1();
           }
         s1.Stop();
         var s2 = Stopwatch.StartNew();
         for (int i = 0; i < _max; i++)
             {
                sum += Method2();
             }
         s2.Stop();
         Console.WriteLine(((double)(s1.Elapsed.TotalMilliseconds * 1000000) /
        _max).ToString("0.00 ns"));
        Console.WriteLine(((double)(s2.Elapsed.TotalMilliseconds * 1000000) /
      _max).ToString("0.00 ns"));
        Console.Read();
      }
      static int Method1()
      {
         // ... No inlining suggestion.
         return "one".Length + "two".Length + "three".Length +
              "four".Length + "five".Length + "six".Length +
              "seven".Length + "eight".Length + "nine".Length +
               "ten".Length;
    }
    [MethodImpl(MethodImplOptions.AggressiveInlining)]
    static int Method2()
    {
        // ... Aggressive inlining.
        return "one".Length + "two".Length + "three".Length +
           "four".Length + "five".Length + "six".Length +
           "seven".Length + "eight".Length + "nine".Length +
            "ten".Length;
    }
}
Output
7.34 ns    No options
0.32 ns    MethodImplOptions.AggressiveInlining
```

در واقع با اعمال این اتریبیوت، محدودیت شماره یک مبنی بر محدودیت حجم کد IL مربوط به متد، در نظر گرفته نخواهد شد .

مطالعه بیشتر: http://dotnet.dzone.com/news/aggressive-inlining-clr-45-jit

/http://www.ademiller.com/blogs/tech/2008/08/c-inline-methods-and-optimization

http://www.dotnetperls.com/aggressiveinlining

http://blogs.msdn.com/b/ericgu/archive/2004/01/29/64644.aspx

https://msdn.microsoft.com/en-us/library/ms973858.aspx#highperfmanagedapps_topic10