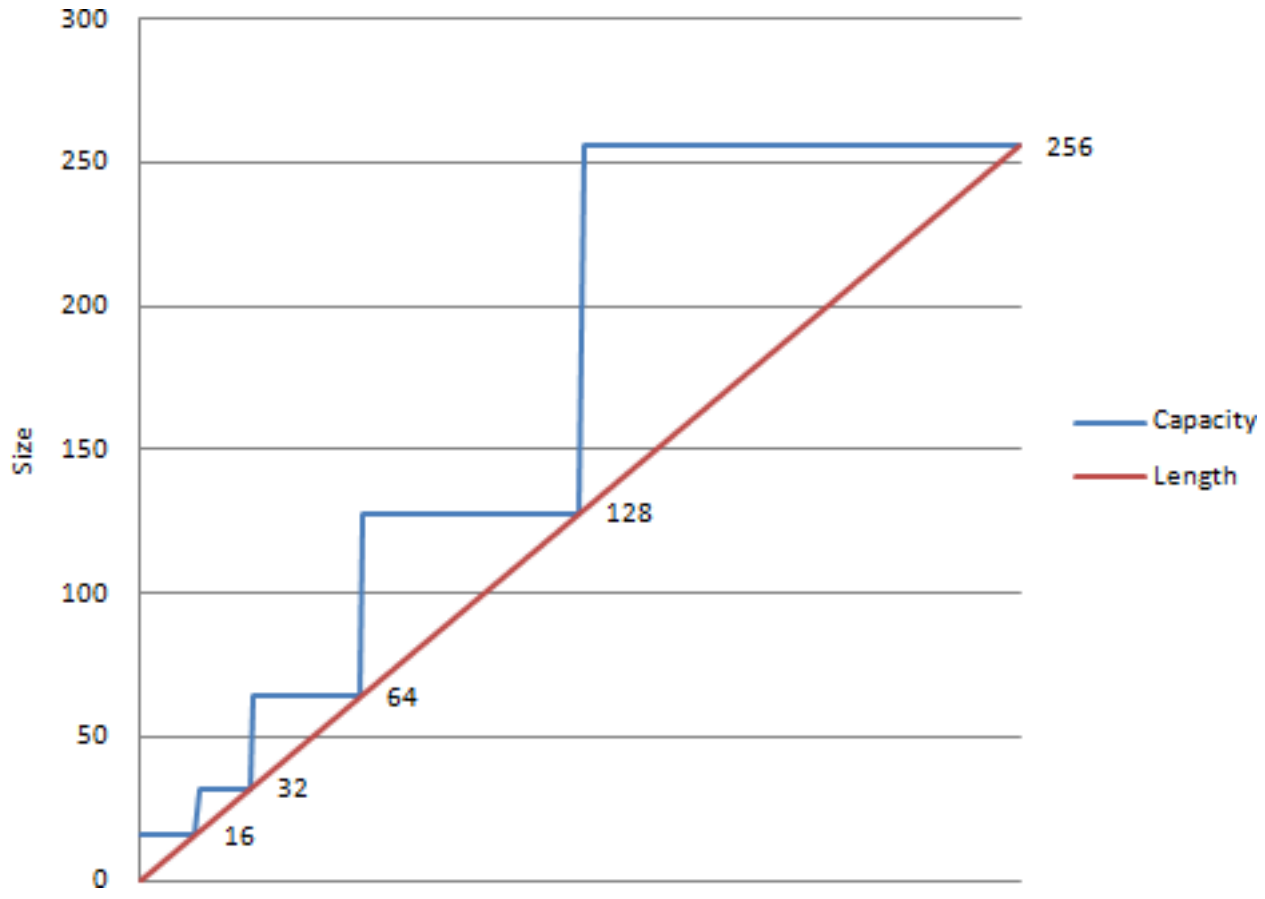
عنوان: **StringBuilder**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۴/۰۶ ۱۶:۴۲
آدرس: www.dotnettips.info
برچسب‌ها: C#, StringBuilder

بهترین روش برای تولید و دستکاری یک رشته (string) طولانی و یا دستکاری متناوب و تکراری یک رشته استفاده از کلاس StringBuilder است. این کلاس در فضای نام System.Text قرار داره. شی String در دات‌نت‌فریمورک تغییرناپذیره (immutable)، بدین معنی که پس از ایجاد نمی‌توان محتوای اونو تغییر داد. برای مثال اگر شما بخواین محتوای یک رشته رو با اتصال به رشته‌ای دیگه تغییر بدین، اجازه اینکار را به شما داده نمی‌شه. درعوض به‌صورت خودکار رشته‌ای جدید در حافظه ایجاد میشه و محتوای دو رشته موجود پس از اتصال به هم درون اون قرار می‌گیره. این کار درصورتی‌که تعداد عملیات مشابه زیاد باشه می‌تونه تاثیر منفی بر کارایی و حافظه خالی در دسترس برنامه بگذاره.

کلاس StringBuilder با استفاده از آرایه‌ای از کاراکترها، راه‌حل مناسب و بهینه‌ای رو برای این مشکل فراهم کرده. این کلاس در زمان اجرا به شما اجازه می‌ده تا بدون ایجاد نمونه‌های جدید از کلاس String، محتوای یک رشته رو تغییر بدین. شما می‌تونید نمونه‌ای از این کلاس رو به‌صورت خالی و یا با یک رشته اولیه ایجاد کنید، سپس با استفاده از متدهای متنوع موجود، محتوای رشته رو با استفاده از انواع داده مختلف و به‌صورت دلخواه دستکاری کنید. هم‌چنین با استفاده از متد معروف ()ToString این کلاس می‌تونید در هر لحظه دلخواه رشته تولیدی رو بدست بیارین. دو پراپرتی مهم کلاس StringBuilder رفتارش رو درهنگام افزودن داده‌های جدید کنترل می‌کنن:

Capacity , Length

پراپرتی Capacity اندازه بافر کلاس StringBuilder را تعیین می‌کنه و Length طول رشته جاری موجود در این بافر رو نمایش می‌ده. اگر پس از افزودن داده جدید، طول رشته از اندازه بافر موجود بیشتر بشه، StringBuilder باید یه بافر جدید با اندازه‌ای مناسب ایجاد کنه تا رشته جدید رو بتونه تو خودش نگه داره. اندازه این بافر جدید به‌صورت پیش‌فرض دو برابر اندازه بافر قبلی درنظر گرفته می‌شه. بعد تمام رشته قبلی رو تو این بافر جدید کپی میکنه.



از برنامه ساده زیر میتونین برای بررسی این مسئله استفاده کنین:

```
using System.IO;
using System.Text;

class Program
{
  static void Main()
  {
    using (var writer = new StreamWriter("data.txt"))
    {
      var builder = new StringBuilder();
      for (var i = 0; i <= 256; i++)
      {
        writer.Write(builder.Capacity);
        writer.Write(",");
        writer.Write(builder.Length);
        writer.WriteLine();
        builder.Append('1'); // <-- Add one character
      }
    }
  }
}
```

دقت کنین که برای افزودن یک کاراکتر استفاده از دستور Append با نوع داده char (همونطور که در بالا استفاده شده) بازدهی بهتری نسبت به استفاده از نوع داده string (با یک کاراکتر) داره. خروجی کد فوق به صورت زیره:

```
16, 0
16, 1
16, 2
...
16,14
16,15
16,16
32,17
```

```
...
```

استفاده نامناسب و بی‌دقت از این کلاس می‌تونه منجر به بازسازی‌های متناوب این بافر شده که درنهایت فواید کلاس StringBuilder رو تحت تاثیر قرار میده. درهنگام کار با کلاس StringBuilder اگر از طول رشته موردنظر و یا حد بالایی برای Capacity آن آگاهی حتی نسبی دارین، می‌تونید با مقداردهی مناسب این پراپرتی از این مشکل پرهیز کنید.

نکته : مقدار پیش‌فرض پراپرتی Capacity برابر 16 است.

هنگام مقداردهی پراپرتی‌های Capacity یا Length به موارد زیر توجه داشته باشید:

- مقداردهی Capacity به میزانی کمتر از طول رشته جاری (پراپرتی Length)، منجر به خطای زیر می‌شه:

System.ArgumentOutOfRangeException

خطای مشابهی هنگام مقداردهی پراپرتی Capacityبه بیش از مقدار پراپرتی MaxCapacity رخ می‌دهه.البته این مورد تنها درصورتی‌که بخواین اونو به بیش از حدود 2 گیگابایت (Int32.MaxValue) مقداردهی کنید پیش میاد!

- اگر پراپرتی Length را به مقداری کمتر از طول رشته جاری تنظیم کنید، رشته به اندازه طول تنظیمی کوتاه (truncate) میشه.

- اگر مقدار پراپرتی Length را به میزانی بیشتر از طول رشته جاری تنظیم کنید، فضای خالی موجود در بافر با space پر میشه.

- تنظیم مقدار Length بیشتر از Capacity، منجر به مقداردهی خودکار پراپرتی Capacity به مقدار جدید تنظیم شده برای Length میشه.

در ادامه به یک مثال برای مقایسه کارایی تولید یک رشته طولانی با استفاده از این کلاس میپردازیم. تو این مثال از دو روش برای تولید رشته‌های طولانی استفاده میشه. روش اول که همون روش اتصال رشته‌ها (Concat) به هم هستش و روش دوم هم که استفاده از کلاس StringBuilder است. در قطعه کد زیر کلاس مربوط به عملیات تست رو مشاهده میکنین:

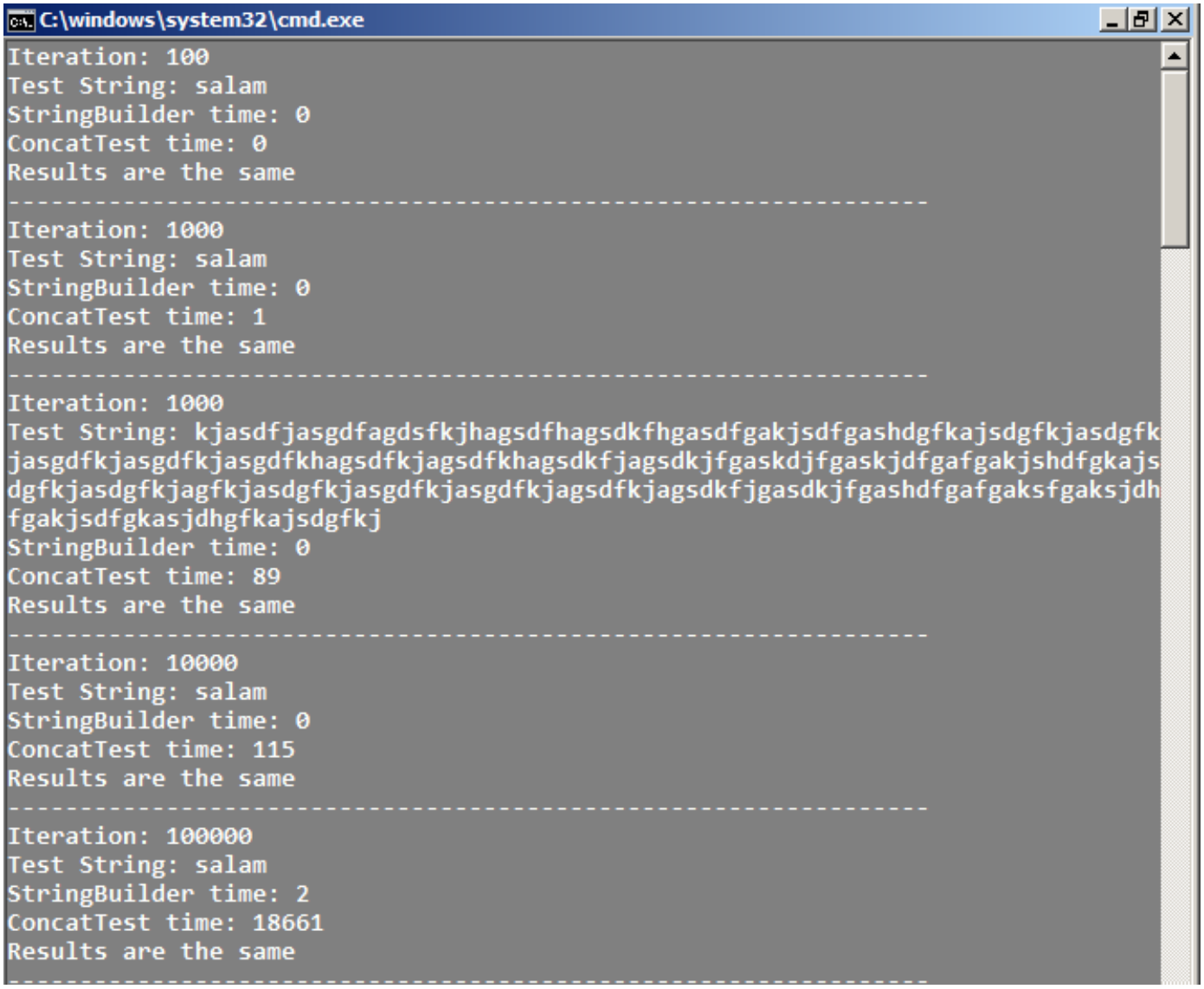
```
namespace StringBuilderTest
{
  internal class SbTest1
  {
    internal Action<string> WriteLog;
    internal int Iterations { get; set; }
    internal string TestString { get; set; }

    internal SbTest1(int iterations, string testString, Action<string> writeLog)
    {
      Iterations = iterations;
      TestString = testString;
      WriteLog = writeLog;
    }

    internal void StartTest()
    {
      var watch = new Stopwatch();

      //StringBuilder
      watch.Start();
      var sbTestResult = SbTest();
      watch.Stop();
      WriteLog(string.Format("StringBuilder time: {0}", watch.ElapsedMilliseconds));

      //Concat
      watch.Start();
      var concatTestResult = ConcatTest();
      watch.Stop();
      WriteLog(string.Format("ConcatTest time: {0}", watch.ElapsedMilliseconds));

      WriteLog(string.Format("Results are{0} the same", sbTestResult == concatTestResult ? string.Empty
```

```
: " NOT"));
    }

    private string SbTest()
    {
      var sb = new StringBuilder(TestString);
      for (var x = 0; x < Iterations; x++)
      {
        sb.Append(TestString);
      }
      return sb.ToString();
    }

    private string ConcatTest()
    {
      string concat = TestString;
      for (var x = 0; x < Iterations; x++)
      {
        concat += TestString;
      }
      return concat;
    }
  }
}
```

دو روش بحث‌شده در کلاس مورد استفاده قرار گرفته و مدت زمان اجرای هر کدوم از عملیات‌ها به خروجی فرستاده میشه. برای استفاده از این کلاس هم میشه از کد زیر در یک برنامه کنسول استفاده کرد:

```
do
{
  Console.Write("Iteration: ");
  var iterations = Convert.ToInt32(Console.ReadLine());
  Console.Write("Test String: ");
  var testString = Console.ReadLine();
  var test1 = new SbTest1(iterations, testString, Console.WriteLine);
  test1.StartTest();
  Console.WriteLine("------------------------------------------------------------------");
} while (Console.ReadKey(true).Key == ConsoleKey.C); // C = continue
```

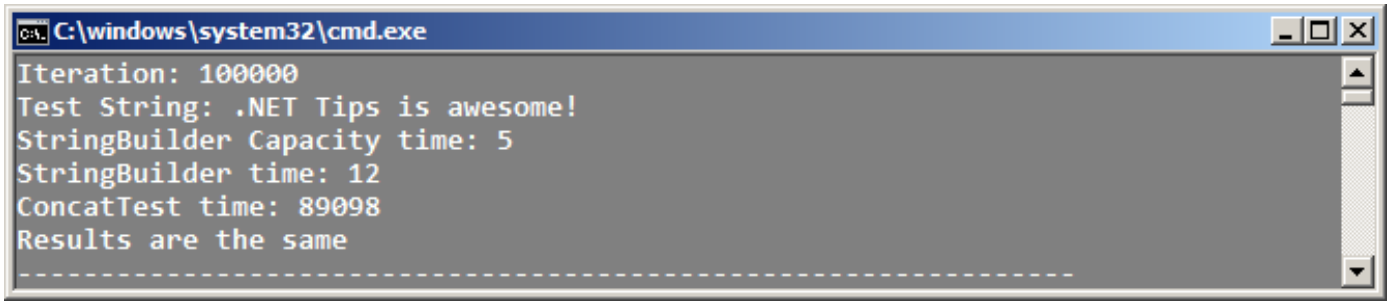
برای نمونه خروجی زیر در لپ‌تاپ من (Corei7 2630QM) بدست اومد:



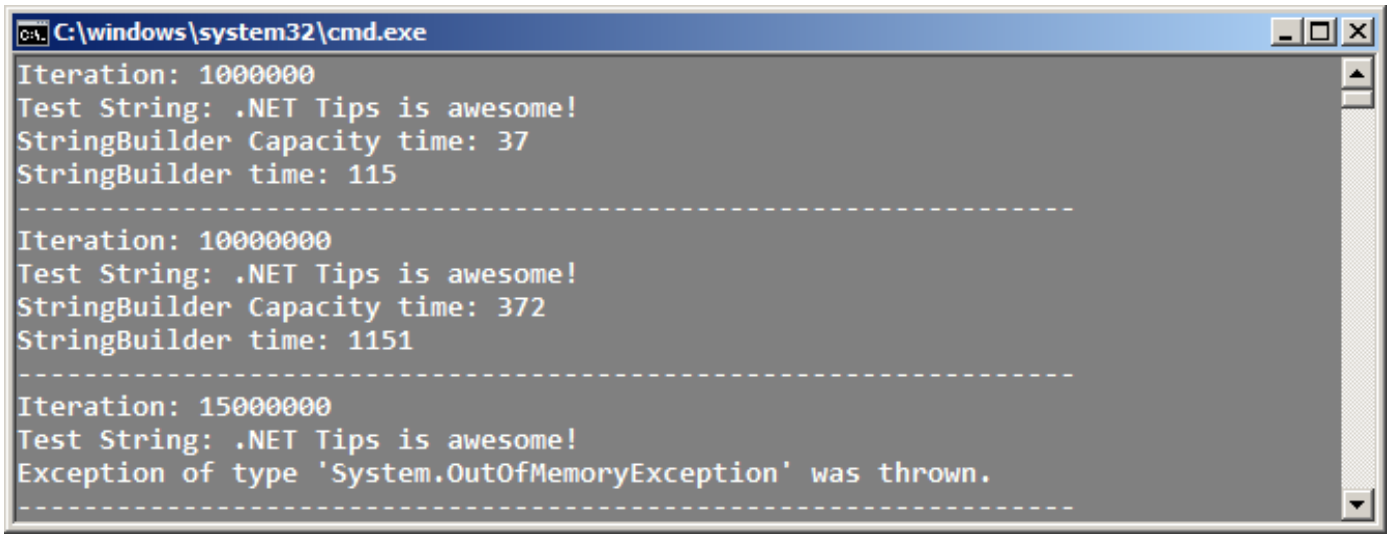
تنظیم خاصیت Capacity به یک مقدار مناسب میتونه تو کارایی تاثیرات زیادی بگذاره. مثلا در مورد مثال فوق میشه یه متد دیگه برای آزمایش تاثیر این مقداردهی به صورت زیر به کلاس برناممون اضافه کنیم:

```
private string SbCapacityTest()
{
  var sb = new StringBuilder(TestString) { Capacity = TestString.Length * Iterations };
  for (var x = 0; x < Iterations; x++)
  {
    sb.Append(TestString);
  }
  return sb.ToString();
}
```

تو این متد قبل از ورود به حلقه مقدار خاصیت Capacity به میزان موردنظر تنظیم شده و نتیجه بدست اومده:

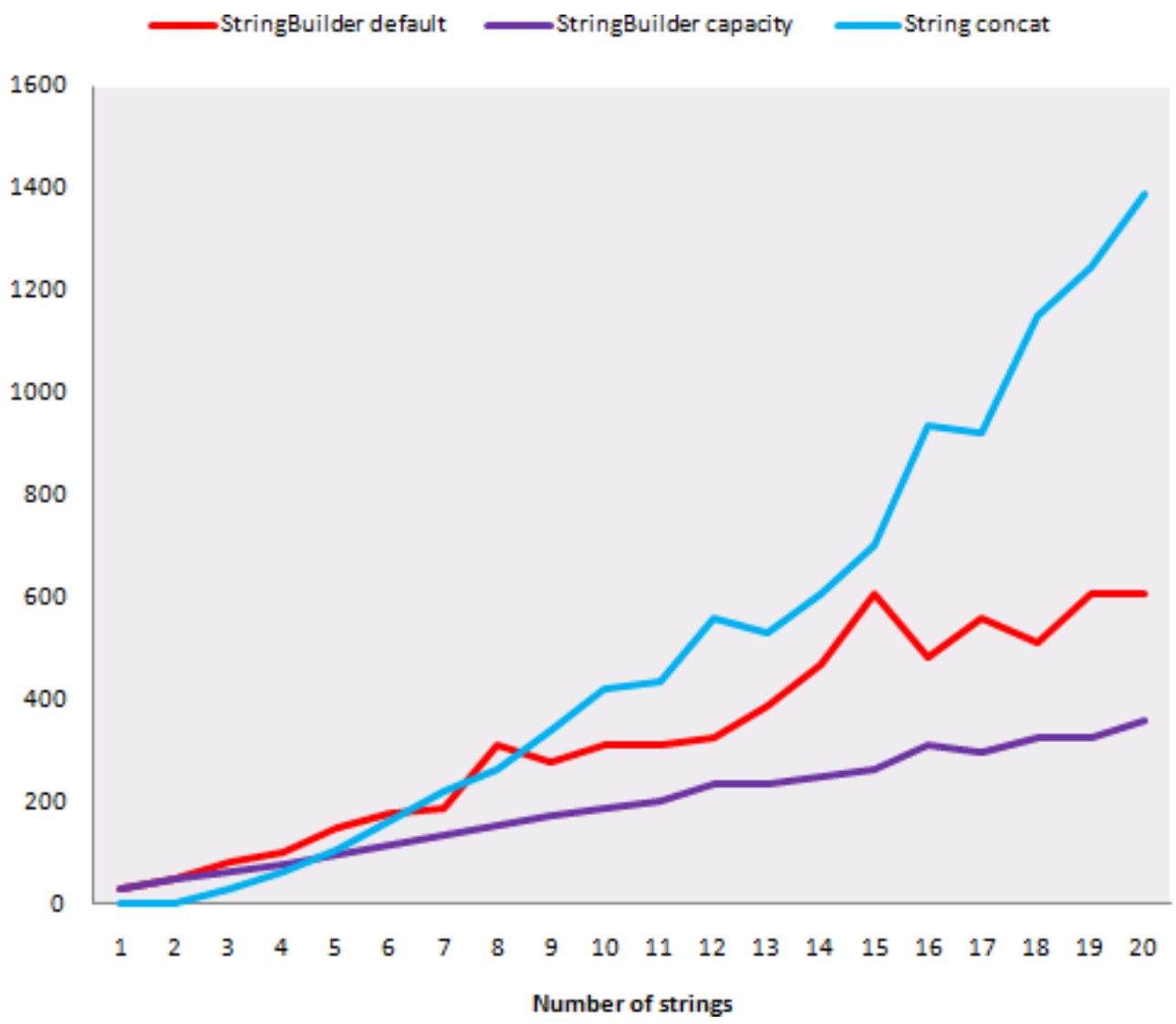


مشاهده میشه که روش concat خیلی کنده (دقت کنین که طول رشته اولیه هم بیشتر شده) و برای ادامه کار مقایسه اون رو کامنت کردم و نتایج زیر بدست اومد:



می‌بینین که استفاده مناسب از مقداردهی به خاصیت Capacity میتونه تا حدود 300 درصد سرعت برنامه ما رو افزایش بده. البته همیشه اینطوری نخواهد بود. ما در این مثال مقدار دقیق طول رشته نهایی رو میدونستیم که باعث میشه عملیات افزایش بافر کلاس StringBuilder هیچوقت اتفاق نیفته. این امر در واقعیت کمتر پیش میاد.

مقاله موجود در سایت dotnetperls شکل زیر رو به عنوان نتیجه تست بازدهی ارائه میده:



- در مواقعی که عملیاتی همچون مثال بالا طولانی و حجیم ندارین بهتره که از این کلاس استفاده نکنین چون عملیات‌های داخلی این کلاس در عملیات کوچک و سبک (مثل ابتدای نمودار فوق) موجب کندی عملیات میشه. همچنین استفاده از اون نیاز به کدنویسی بیشتری داره.

- این کلاس فشار کمتری به حافظه سیستم وارد میکنه. درمقابل استفاده از روش concat موجب اشغال بیش از حد حافظه میشه که خودش باعث اجرای بیشتر و متناوب‌تر GC میشه که در نهایت کارایی سیستم رو کاهش میده.

- استفاده از این کلاس برای عملیات Replace (و یا عملیات مشابه) در حلقه‌ها جهت کار با رشته‌های طولانی و یا تعداد زیادی رشته میتونه بسیار سریعتر و بهتر عمل کنه چون این کلاس برخلاف کلاس string اشیای جدید تولید نمیکنه.

- یه اشتباه بزرگ در استفاده از این کلاس استفاده از "+" برای اتصال رشته‌های درون StringBuilder هست. هرگز از این کارها نکنین. (فکر کنم واضحه که چرا)

نظرات خوانندگان

نویسنده: پویان
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۰:۲۲

بسیار مفید بود این اطلاعات ممنون

نویسنده: بهروز راد
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۳:۱۹

بهتره بعد از اتمام کار با شی ایجاد شده از کلاس StringBuilder، متد Clear اون رو فراخوانی کنید تا حافظه‌ی اشغال شده سریعتر توسط GC آزاد بشه.

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۵:۸

دقیقا (با تشکر بابت یادآوری).
اما این متد از دات نت 4.0 به بعد اضافه شده. برای نسخه‌های قدیمی میشه از مقداردهی متغیر موردنظر به یک اینستنس جدید از کلاس (برای از بین بردن تمام ریفرنسهای اولیه به آبجکت قدیمی تا GC این نمونه قدیمی رو garbage تشخیص بده) استفاده کرد. یا از راه حل ساده‌تر مقداردهی پراپرتی Length به صفر بهره برد. (یا ^)

نویسنده: hossein101211
تاریخ: ۱۳۹۲/۰۳/۲۳ ۱۴:۱۸

با تشکر از مطلب خوبتون
میخواستم ببینم چه تفاوتی با stringWriter داره؟
اگه امکان داره دلایل استفاده از هر کدوم رو بیان نمایید
آیا این درسته که زمانی از string Writer استفاده میکنیم که فقط قصد نوشتن متن رو در memory داریم بدون اینکه مثلا عملیات اضافی مثل append داشته باشیم؟
وزمانی از StringBuilder استفاده میکنیم که میخوایم عملیات‌های اضافی‌تری و طولانی‌تری انجام بدیم؟

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۲۴ ۲۰:۴۲

StringWriter یک TextWriter است (درواقع از آن مشتق شده است) که برای نوشتن متن درون حافظه موقت سیستم طراحی شده است. بنابراین هرجا که نیاز به یک TextWriter باشد میتوان از آن استفاده کرد تا رشته تولیدی درون حافظه نگهداری شود. مثل این مثال درباره HtmlTextWriter .
درضمن StringWriter برای تولید تکست، از StringBuilder استفاده میکند و حتی متدی برای برگرداندن نمونه داخلی از StringBuilder خود دارد.
بنابراین برای تولید رشته با تعداد زیاد عملیات چسباندن رشته‌ها (معمولا بیش از 5 بار) از StringBuilder استفاده میشود و درصورت نیاز به یک نمونه از TextWriter بدون اینکه مکان ذخیره تکست نهایی برای ما مهم باشد از StringWriter استفاده میشود (چون این تکست به صورت موقت درون خافظه سیستم نگهداری میشود). اطلاعات بیشتر
و بیشترتر

عنوان: **Microbenchmark**
نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۵:۴۳
آدرس: www.dotnettips.info
برچسب‌ها: StringBuilder, Microbenchmark

What Is Micro Benchmark? Micro benchmark is a benchmark designed to measure the performance of a very small and specific piece of code. (^)

البته این موضوع امروزه بیشتر در Java مطرحه تا دات نت (^ و ^ و ^) اما مفاهیم اصلی مختص یک زبان یا پلتفرم نیست.
وقتی در مورد آزمایش بار برای مقایسه کارایی کلاس StrigBuilder تحقیق میکردم به مطلب جالبی برخورد کردم. خلاصش این میشه که برای تست بار قسمتهایی از کدتون میتونین زمان موردنیاز برای اجرای اون کد رو بررسی کنین و چون ممکنه انجام این کار چندین بار نیاز بشه بهتره از متد زیر برای اینکار استفاده کنین:

```
static void Profile(string description, int iterations, Action func) {
    // clean up
    GC.Collect();
    GC.WaitForPendingFinalizers();
    GC.Collect();

    // warm up
    func();

    var watch = Stopwatch.StartNew();
    for (int i = 0; i < iterations; i++) {
        func();
    }
    watch.Stop();
    Console.Write(description);
    Console.WriteLine(" Time Elapsed {0} ms", watch.ElapsedMilliseconds);
}
```

سه خط اول متد بالا برای آماده‌سازی حافظه جهت اجرای تست موردنظر است. برای آشنایی بیشتر با نحوه عملکرد Garbage Collector (^ و ^ و ^) خوندن کتاب فوق العاده CLR via C# - 3rd ed رو پیشنهاد میکنم (فصلهای 21 و 22).
سپس یکبار اکشن موردنظر (که حاوی قطعه کد موردنظره) اجرا میشه تا مسائل مربوطه به بارگذاری‌های اولیه در نتیجه تست تاثیر نزاره (warm up).
در نهایت هم آزمایش بار برای تعداد تکرار درخواست شده انجام میشه و زمان اجرای اون در خروجی چاپ میشه.
برای استفاده از متد فوق میشه از کد زیر استفاده کرد:

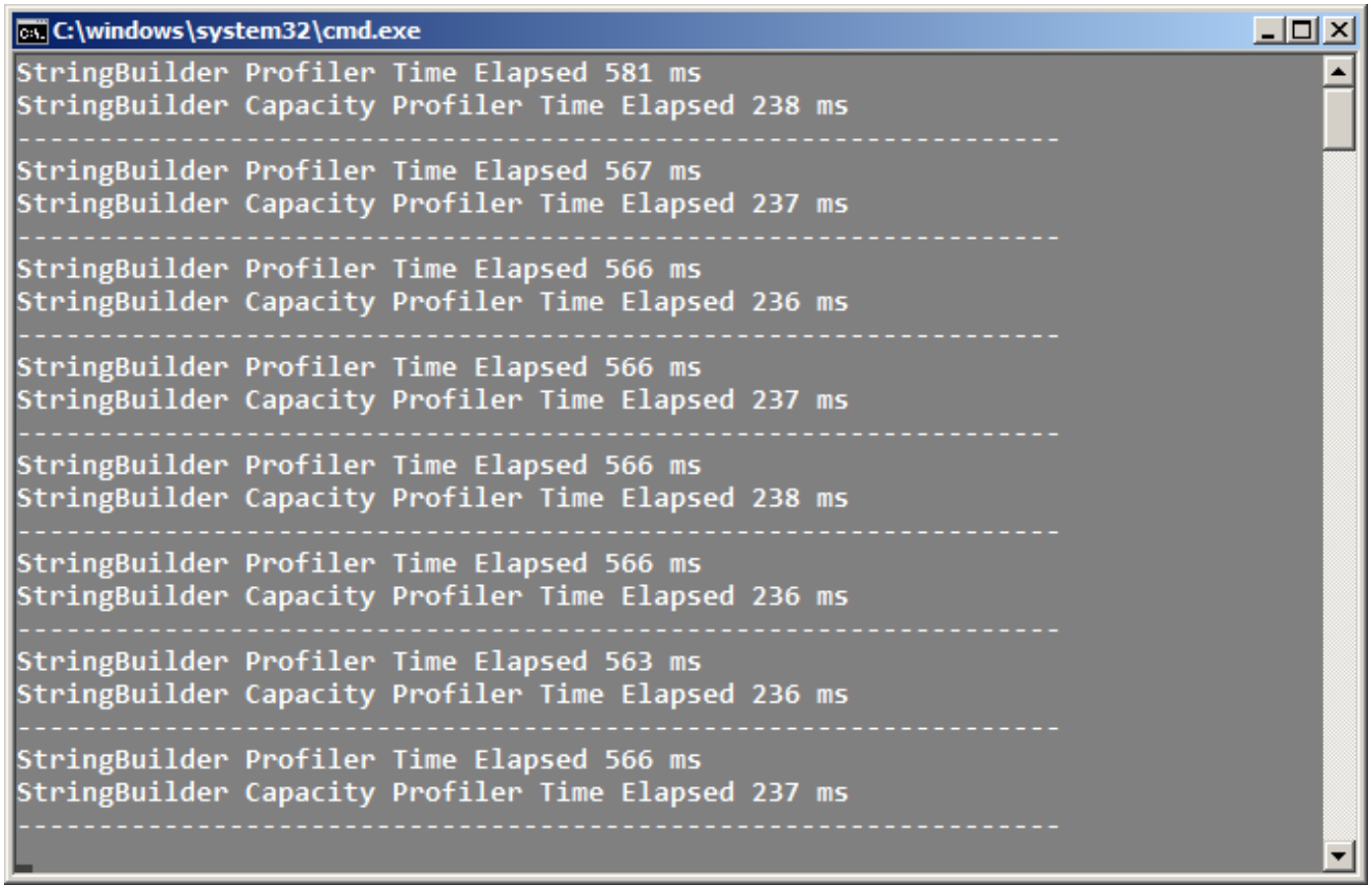
```
Profile("a descriptions", how_many_iterations_to_run, () =>
{
   // ... code being profiled
});
```

و برای استفاده از این متد در آزمایش کارایی کلاس StringBuilder میشه از کدی شبیه به کد زیر استفاده کرد:

```
var iterations = 10000000;
var testString = ".NET Tips is awesme!";
do
{
  var sb1 = new StringBuilder(testString);
  var sb2 = new StringBuilder(testString) { Capacity = testString.Length * iterations };
  try
  {
    Profiler.Profile("StringBuilder Profiler", iterations, () => sb1.Append(testString));
    Profiler.Profile("StringBuilder Capacity Profiler", iterations, () => sb2.Append(testString));
  }
  catch (Exception ex)
  {
    Console.WriteLine(ex.Message);
  }
  finally
```

```
  {
    Console.WriteLine("----------------------------------------------------------------");
    sb1.Clear();
    sb2.Clear();
  }
} while (Console.ReadKey(true).Key == ConsoleKey.C); // C = continue
```

البته برای اینکه عملیات مقدار دهی خاصیت Capacity در قسمت warm up متد profile نتایج رو تحت تاثیر قرار نده برای این تست من اون قسمت رو کامنت کردم (اگر این کار رو نکنین زمانهای بدست اومده برای هر دو مورد یکی خواهد بود). اجرای کد بالا نتایج زیر رو تو سیستم من ارائه داد:



می‌بینین که نتایج استفاده از متد موردبحث کمی فرق داره و افزایش کارایی در حالت استفاده از پراپرتی Capacity دیگه حدود 3 برابر نیست و حدود 2 دو برابره. البته زمان بدست اومده برای هر دو مورد نسبت به قبل کاهش داشته که بیشترش میتونه مربوطه به عدم درنظر گرفتن زمان موردنیاز برای ایجاد کلاس StringBuilder در این تست جدید باشه (چون بعید میدونم عملیات پاکسازی حافظه توسط GC تو این تست تاثیر چندانی داشته باشه). درهر حال نتایج این تست بیشتر به واقعیت نزدیکه!

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۷:۴

مطلب جالبی هست از یکی از اعضای تیم کامپایلر سی شارپ :(^)
بحث محاسبه کارآیی در دات نت شامل زمان صرف شده برای JIT اولیه کدها هم هست. به همین جهت اجرای اولیه اندکی بیشتر زمان می‌بره. همچنین GC هم در اینجا در ترد دیگری به موازات کار شما مشغول به کار است و اگر در یک اجرا زمان خوبی بدست آوردید به این معنا نیست که الزاما در اجرای بعدی هم همان زمان را بدست می‌آورید چون GC موکول شده به بعد. ضمن اینکه این نوع محاسبات چون به صورت ایزوله انجام می‌شود عموما بیانگر شرایط دنیای واقعی که پارامترهای زیادی در آن‌ها دخیل هستند، نیست.
و ... اینکه برای خیلی از برنامه نویس‌ها این نوع مقایسه‌ها بیشتر جذاب هستند:
Head-to-head benchmark: C++ vs .NET

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۱/۰۴/۰۷ ۱۹:۸

مطالب شما کاملا صحیح و صادق هست.
اما هدف اینگونه آزمایشات (Microbenchmark) مقایسه قطعات کد درون یک برنامس و مثلا بدست آوردن بهترین روش برای رسیدن به یک هدف مشخص. مثلا همین مثال کلاس StringBuilder که بین دو روش ذکر شده کدام سریعتره و چقدر بهتره و اینکه درنهایت با استفاده از نتایج این آزمایشات و سایر داده‌های موجود کدام روش به صرفه تره. یا مثلا در دستکاری لیست‌های بزرگ استفاده از آرایه به صرفه‌تره یا مثلا یک کالکشن از انواع موجود. در مورد JIT هم با استفاده از بخش warm up سعی شده اثر منفی کامپایل اولیه کد رو در تست از بین ببره (هرچند اثر منفی اجرای خود کد تستر در بار اول همچنان پابرجاس).
اما در مورد اثر GC با اینکه در ابتدای متد مذکور سعی شده تا با پاکسازی اولیه حافظه، سیستم آماده انجام آزمایش بشه ولی هیچ تضمینی نیست که در میانه تست GC بطور خودکار فعال نشه، که میتونه رو نتایج تاثیر منفی بذاره. تو این موارد دیگه خود برنامه نویس باید با درنظر گرفتن این مسئله در مورد نتایج بدست اومده تصمیم بگیره.