عنوان: **intern pool جدول نگهداری رشته‌ها در دات‌نت**
نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۲/۰۲/۲۹ ۲۲:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: C#, Tips, CLR, string, intern, pool

کد زیر را در نظر بگیرید :

```
object text1 = "test";
object text2 = "test";

object num1 = 1;
object num2 = 1;

Console.WriteLine("text1 == text2 : " + (text1 == text2));
Console.WriteLine("num1 == num2 : " + (num1 == num2));
```

به نظر شما چه چیزی در خروجی نمایش داده میشود؟

هر چهار متغییر text1 و text2 و num1 و num2 از نوع object هستند. با اینکه مقدار text1 و text2 یکی و مقدار num1 و num2 هم یکی است، نتیجه text1==text2 برابر true است اما num1==num2 برابر false.

خطی که text2 تعریف شده است را تغییر میدهیم :

```
object text2 = "test".ToLower();
```

اینبار با این که باز مقدار text1 و text2 یکی و هر دو "test" است، اما نتیجه text1==text2 برابر false است. انتظار ما هم همین است. دو object ایجاد شده است و یکی نیستند. تنها در صورتی باید نتیجه == آنها true باشد که هر دو به یک object اشاره کنند.

اما چرا در کد اولی اینگونه نبود؟

دلیل این کار برمیگردد به رفتار دات‌نت نسبت به رشتههایی که به صورت صریح در برنامه تعریف میشوند. CLR یک جدول برای ذخیره رشتهها به نام **intern pool** برای برنامه میسازد. هر رشتهای تعریف میشود، اگر در intern pool رشتهای با همان مقدار وجود نداشته باشد، یک رشته جدید ایجاد و به جدول اضافه میشود، و اگر موجود باشد متغییر جدید فقط به آن اشاره میکند. در واقع اگر 100 جای برنامه حتی در کلاسهای مختلف، رشتههایی با مقادیر یکسان وجود داشته باشند، برای همه آنها یک نمونه وجود دارد.

بنابراین text1 و text2 در کد اولی واقعا یکی هستند و یک نمونه برای آنها ایجاد شده است.

البته چند نکته در اینجا هست :
اگر text1 و text2 به صورت string تعریف شوند، نتیجه text1==text2 در هر دو حالت فوق برابر true است. چون عملگر == در کلاس string یکبار دیگر overload شده است:

```
public sealed class String : ...
{
    ...
    public static bool operator ==(string a, string b)
    {
```

```
      return string.Equals(a, b);
    }
   ...
}
```

این که کدام یک از overloadها اجرا شوند (کلاس پایه، کلاس اصلی، ...) به نوع دو متغییر اطراف == بستگی دارد. مثلا در کد زیر :

```
string text1 = "test";
string text2 = "test".ToLower();

Console.WriteLine("text1 == text2 (string) : " + (text1 == text2));
Console.WriteLine("text1 == text2 (object) : " + ((object)text1 == (object)text2));
```

اولین نتیجه true و دومی false است. چون در اولی عملگر == تعریف شده در کلاس string مورد استفاده قرار می‌گیرد اما در دومی عملگر == تعریف شده در کلاس object.

اگر دقت نشود این رفتار مشکلزا میشود. مثلا حالتی را در نظر بگیرید که text1 ورودی کاربر است و text2 از بانک اطلاعاتی خوانده شده است و با اینکه مقادیر یکسان دارند نتیجه == آنها false است. اگر تعریف عملگرها در کلاس object به صورت virtual بود و در کلاس‌های دیگر override می‌شد، این تغییر نوع‌ها تاثیری نداشت. اما عملگرها به صورت static تعریف می‌شوند و امکان override شدن ندارند. به همین خاطر کلاس object متدی به اسم Equals در اختیار گذاشته که کلاس‌ها آنرا override می‌کنند و معمولا از این متد برای سنجش برابری دو کلاس استفاده می‌شود :

```
object text1 = "test";
object text2 = "test".ToLower();

Console.WriteLine("text1 Equals text2 : " + text1.Equals(text2));
Console.WriteLine("text1 Equals text2 : " + object.Equals(text1, text2));
```

البته یادآور می‌شوم که **فقط رشته‌هایی که به صورت صریح در برنامه تعریف شده‌اند، در intern pool قرار می‌گیرند و این فهرست شامل رشته‌هایی که از فایل یا بانک اطلاعاتی خوانده می‌شوند یا در برنامه تولید می‌شوند، نیست.** این کار منطقی است وگرنه حافظه زیادی مصرف خواهد شد.

با استفاده از متد string.Intern می‌توان یک رشته را که در intern pool وجود ندارد، به فهرست آن افزود. اگر رشته در intern pool وجود داشته باشد، reference آنرا بر می‌گرداند در غیر اینصورت یک reference به رشته جدید به intern pool اضافه می‌کند و آنرا برمی‌گرداند.

یک مورد استفاده آن هنگام lock روی رشته‌هاست. برای مثال در کد زیر DeviceId یک رشته است که از بانک اطلاعاتی خوانده می‌شود و باعث می‌شود که چند job همزمان به یک دستگاه وصل نشوند :

```
lock (job.DeviceId)
{
    job.Execute();
}
```

اگر یک job با DeviceId برابر COM1 در حال اجرا باشد، این lock جلوی اجرای همزمان job دیگری با همین DeviceId را

نمی‌گیرد. زیرا هر چند مقدار DeviceId دو job یکی است ولی به یک نمونه اشاره نمی‌کنند.

می‌توان lock را اینگونه اصلاح کرد :

```
lock (string.Intern(job.DeviceId))
{
    job.Execute();
}
```

نظرات خوانندگان

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۲/۳۰ ۰:۳۵

ممنون. البته شرایط کد خودتون رو کامل اینجا قرار ندادید ولی در حالت کلی توصیه میشه که برای استفاده از lock یک شیء private object در سطح کلاس تعریف بشه و از اون استفاده بشه تا حالت‌های دیگر .

نویسنده: رحمت اله رضایی
تاریخ: ۱۳۹۲/۰۲/۳۱ ۹:۵۶

- البته این فقط یک مثال بود برای درک متد string.Intern .
- چگونگی شی معرفی شده به lock هم بسته به شرایط ممکن است متفاوت باشد. ممکن است یک private object در سطح همان کلاسی که lock در آن استفاده می‌شود، جوابگو باشد. اما در شرایط دیگری ممکن است اینگونه نباشد. مانند مثال فوق.

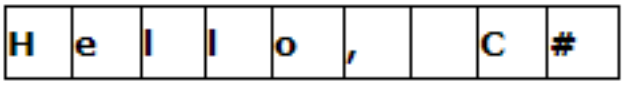
عنوان: **رشته ها و پردازش متن در دات نت به زبان ساده**
نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۲۸ ۱۹:۳۵
آدرس: www.dotnettips.info
گروه‌ها: C#, Regular expressions, .NET, string, Performance

رشته، مجموعه‌ای از کاراکترهاست که پشت سرهم، در مکانی از حافظه قرار گرفته‌اند. هر کاراکتر حاوی یک شماره سریال در جدول یونیکد هست. به طور پیش فرض دات نت برای هر کاراکتر (نوع داده char) شانزده بیت در نظر گرفته است که برای 65536 کاراکتر کافی است.

برای نگهداری از رشته‌ها و انجام عملیات بر روی آنها در دات نت از نوع system.string استفاده می‌کنیم:

```
string greeting = "Hello, C#";
```

که در این حالت مجموعه‌ای از کاراکترها را ایجاد خواهد کرد:



اتفاقاتی که در داخل کلاس string رخ می‌دهد بسیار ساده است و ما را از تعریف []char بی‌نیاز می‌کند تا مجبور نشویم خانه‌های آرایه را به ترتیب پر کنیم. از معایب استفاده از آرایه char میتوان موارد زیر را برشمارد:
خانه‌های آن یک ضرب پر نمیشوند بلکه به ترتیب، خانه به خانه پر می‌شوند.
قبل از انتساب متن باید باید از طول متن مطمئن شویم تا بتوانیم تعداد خانه‌ها را بر اساس آن ایجاد کنیم.
همه عملیات آرایه‌ها از پر کردن ابتدای کار گرفته تا هر عملی، نیاز است به صورت دستی صورت بگیرد و تعداد خطوط کد برای هر کاری هم بالا می‌رود.

البته استفاده از string هم راه حل نهایی برای کار با متون نیست. در انتهای این مطلب مورد دیگری را نیز بررسی خواهیم کرد.
از ویژگی دیگر رشته‌ها این است که آن‌ها شباهت زیادی به آرایه‌ای از کاراکترها دارند؛ ولی اصلا شبیه آن‌ها نیستند و نمی‌توانید به صورت یک آرایه آن‌ها را مقداردهی کنید. البته کلاس string امکاناتی را با استفاده از indexer [] مهیا کرده است که میتوانید بر اساس اندیس‌ها به کاراکترها به صورت جداگانه دسترسی داشته باشید ولی نمی‌توانید آن‌ها را مقدار دهی کنید. این اندیس‌ها از 0 تا طول آن length-1 ادامه دارند.

```
string str = "abcde";
char ch = str[1]; // ch == 'b'
str[1] = 'a'; // Compilation error!
ch = str[50]; // IndexOutOfRangeException
```

همانطور که میدانیم برای مقداردهی رشته‌ها از علامت‌های نقل قول "" استفاده میکنیم که باعث میشود اگر بخواهیم علامت " را در رشته‌ها داشته باشیم نتوانیم. برای حل این مشکل از علامت \ استفاده میکنیم که البته باعث استفاده از بعضی کاراکترهای خاص دیگر هم می‌شود:

```
string a="Hello \"C#\"";
string b="Hello \r\n C#"; //مساوی با اینتر
string c="C:\\a.jpg"; //مسیردهی- \ چاپ خود علامت
```

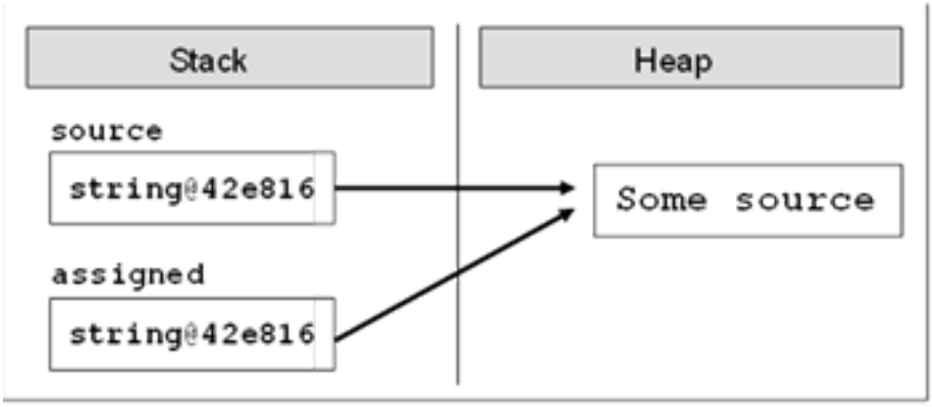
البته اگر از علامت @ در قبل از رشته استفاده شود علامت \ بی اثر خواهد شد.

```
string c=@"C:\a.jpg";// == "C:\\a.jpg"
```

مقداردهی رشته‌ها و پایدار (تغییر ناپذیر) بودن آنها Immutable

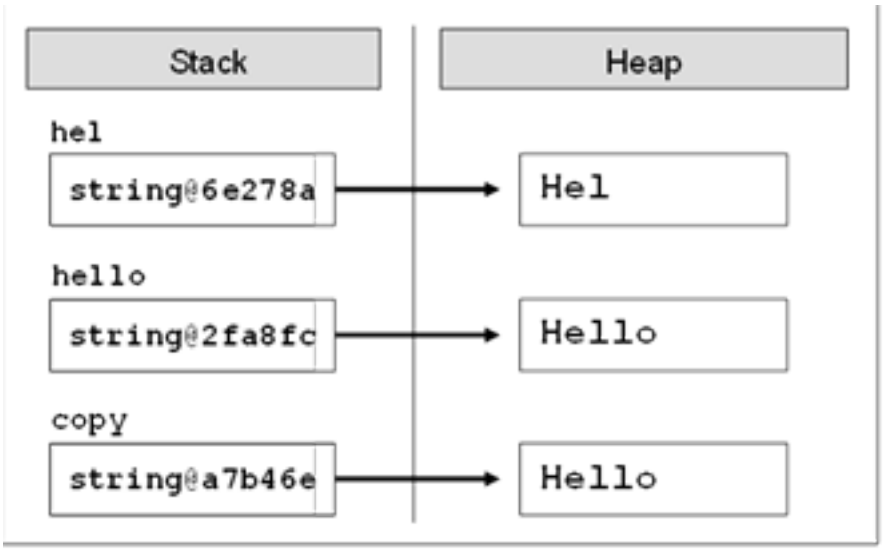
رشته‌ها ساختاری پایدار هستند؛ به این معنی که به صورت reference مقداردهی می‌شوند. موقعی که شما مقداری را به یک رشته انتساب می‌دهید، مقدار متغیر در String pool یا لینک در Heap ذخیره می‌شوند و اگر همین متغیر را به یک متغیر دیگر انتساب دهیم، متغیر جدید مقدار آن را دیگر در حافظه پویا (داینامیک) Heap به عنوان مقدار جدید ذخیره نخواهد کرد؛ بلکه تنها یک pointer خواهد بود که به آدرس حافظه متغیر اولی اشاره می‌کند. به مثال زیر دقت کنید. متغیر source مقدار some source را ذخیره می‌کند و بعد همین متغیر، به متغیر assigned انتساب داده میشود؛ ولی مقداری جابجا نمی‌شود. بلکه متغیر assign به آدرسی در حافظه اشاره می‌کند که متغیر source اشاره می‌کند. هرگاه که در یکی از متغیرها، تغییری رخ دهد، همان متغیری که تغییر کرده است، به آدرس جدید با محتوای تغییر داده شده اشاره می‌کند.

```
string source = "Some source";
string assigned = source;
```

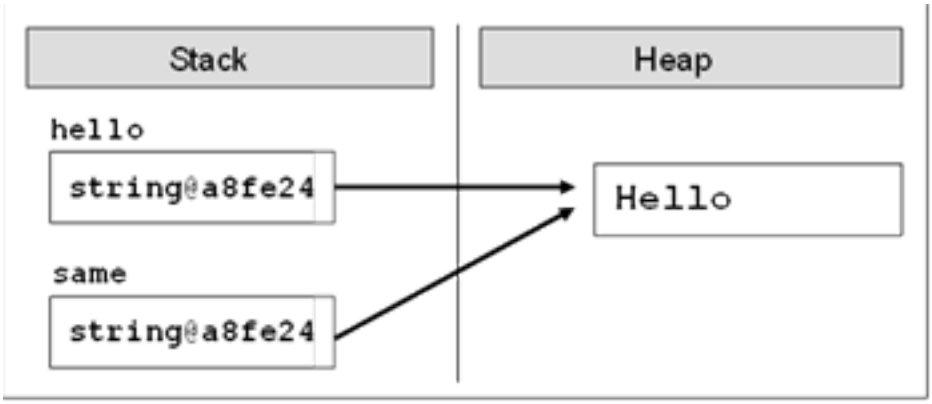


این ویژگی نوع reference فقط برای ساختارهای Immutable به معنی پایدار رخ می‌دهد و نه برای ساختارهای ناپایدار (تغییر پذیر) mutable؛ به این خاطر که آن‌ها مقادیرشان را مستقیما تغییر میدهند و اشاره‌ای در حافظه صورت نمی‌گیرد.

```
string hel = "Hel";
string hello = "Hello";
string copy = hel + "lo";
```



```
string hello = "Hello";
string same = "Hello";
```



برای اطلاعات بیشتر در این زمینه این لینک را مطالعه نمایید.

مقایسه رشته‌ها

برای مقایسه دو رشته میتوان از علامت == یا از متد Equals استفاده نماییم. در این حالت به خاطر اینکه کد حروف کوچک و بزرگ متفاوت است، مقایسه حروف هم متفاوت خواهد بود. برای اینکه حروف کوچک و بزرگ تاثیری بر مقایسه ما نگذارند و c# را با C# برابر بدانند باید از متد Equals به شکل زیر استفاده کنیم:

```
Console.WriteLine(word1.Equals(word2,
    StringComparison.CurrentCultureIgnoreCase));
```

برای اینکه بزرگی و کوچکی اعداد را مشخص کنیم از علامت‌های < و > استفاده میکنیم ولی برای رشته‌ها از متد CompareTo بهره می‌بریم که چینش قرارگیری آن‌ها را بر اساس حروف الفبا مقایسه می‌کند و سه عدد، می‌تواند خروجی آن باشند. اگر 0 باشد یعنی برابر هستند، اگر -1 باشد رشته اولی قبل از رشته دومی است و اگر 1 باشد رشته دومی قبل از رشته اولی است.

```
string score = "sCore";
string scary = "scary";

Console.WriteLine(score.CompareTo(scary));
Console.WriteLine(scary.CompareTo(score));
Console.WriteLine(scary.CompareTo(scary));

// Console output:
// 1
// -1
// 0
```

اینبار هم برای اینکه حروف کوچک و بزرگ، دخالتی در کار نداشته باشند، میتوانید از داده شمارشی StringComparison در متد ایستای string.Compare(s1,s2,StringComparison) استفاده نمایید؛ یا از نوع داده‌ای boolean برای تعیین نوع مقایسه استفاده کنید.

```
string alpha = "alpha";
string score1 = "sCorE";
string score2 = "score";

Console.WriteLine(string.Compare(alpha, score1, false));
Console.WriteLine(string.Compare(score1, score2, false));
Console.WriteLine(string.Compare(score1, score2, true));
```

```
Console.WriteLine(string.Compare(score1, score2,
    StringComparison.CurrentCultureIgnoreCase));
// Console output:
// -1
// 1
// 0
// 0
```

نکته : برای مقایسه برابری دو رشته از متد Equals یا == استفاده کنید و فقط برای تعیین کوچک یا بزرگ بودن از compareها استفاده نمایید. دلیل آن هم این است که برای مقایسه از فرهنگ culture فعلی سیستم استفاده میشود و نظم جدول یونیکد را رعایت نمی‌کنند و ممکن است بعضی رشته‌های نابرابر با یکدیگر برابر باشند. برای مثال در زبان آلمانی دو رشته "SS" و "ß " با یکدیگر برابر هستند.

عبارات با قاعده Regular Expression

این عبارات الگوهایی هستند که قرار است عبارات مشابه الگویی را در رشته‌ها پیدا کنند. برای مثال الگوی +[A-Z0-9] مشخص می‌کند که رشته مورد نظر نباید خالی باشد و حداقل با یکی از حروف بزرگ یا اعداد پرشده باشد. این الگوها میتوانند برای واکشی داده‌ها یا قالب‌های خاص در رشته‌ها به کار بروند. برای مثال شماره تماس‌ها ، پست الکترونیکی و ...

در اینجا میتواند نحوه‌ی الگوسازی را بیاموزید. کد زیر بر اساس یک الگو، شماره تماس‌های مورد نظر را یافته و البته با فیلتر گذاری آن‌ها را نمایش می‌دهد:

```
string doc = "Smith's number: 0898880022\nFranky can be " +
    "found at 0888445566.\nSteven's mobile number: 0887654321";
string replacedDoc = Regex.Replace(
    doc, "(08)[0-9]{8}", "$1********");
Console.WriteLine(replacedDoc);
// Console output:
// Smith's number: 08********
// Franky can be found at 08********.
// Steven' mobile number: 08********
```

سه شماره تماس در رشته‌ی بالا با الگوی ما همخوانی دارند که بعد با استفاده از متد replace در شی Regex عبارات دلخواه خودمان را جایگزین شماره تماس‌ها خواهیم کرد. الگوی بالا شماره تماس‌هایی را میابد که با 08 آغاز شده‌اند و بعد از آن 8 عدد دیگر از 0 تا 9 قرار گرفته‌اند. بعد از اینکه متن مطابق الگو یافت شد، ما آن را با الگوی ********$1 جایگزین می‌کنیم که علامت $ یک placeholder برای یک گروه است. هر عبارت () در عبارات با قاعده یک گروه حساب میشود و اولین پرانتز $1 و دومین پرانتز یا گروه میشود $2 که در عبارت بالا (08) میشود $1 و به جای مابقی الگو، 8 علامت ستاره نمایش داده میشود.

اتصال رشته‌ها در Loop

برای اتصال رشته‌ها ما از علامت + یا متد ایستای string.concat استفاده می‌کنیم ولی استفاده‌ی از آن در داخل یک حلقه باعث کاهش کارآیی برنامه خواهد شد. برای همین بیایید ببینم در حین انتقال رشته‌ها در حافظه چه اتفاقی رخ میدهد. ما در اینجا دو رشته str1 و str2 داریم که عبارات "super" و "star" را نگه داری می‌کنند و در واقع دو متغیر هستند که به حافظه‌ی پویای Heap اشاره می‌کنند. اگر این دو را با هم جمع کنیم و نتیجه را در متغیر result قرار دهیم، سه متغیر میشوند که هر کدام به حافظه‌ای جداگانه در heap اشاره می‌کنند. در واقع برای این اتصال، قسمت جدیدی از حافظه تخصصیص داده شده و مقدار جدید در آن نشسته‌است. در این حالت یک متغیر جدید ساخته شد که به آدرس آن اشاره می‌کند. کل این فرآیند یک فرآیند کاملا زمانبر است که با تکرار این عمل موجب از دست دادن کارآیی برنامه می‌شود؛ به خصوص اگر در یک حلقه این کار صورت بگیرد.

سیستم دات نت همانطور که میدانید شامل GC یا سیستم خودکار پاکسازی حافظه است که برنامه نویس را از dispose کردن بسیاری از اشیاء بی نیاز می‌کند. موقعی‌که متغیری به قسمتی از حافظه اشاره می‌کند که دیگر بلا استفاده است، سیستم GC به صورت خودکار آنها را پاکسازی می‌کند که این عمل زمان بر هم خودش موجب کاهش کارآیی می‌شود. همچنین انتقال رشته‌ها از یک مکان حافظه به مکانی دیگر، باز خودش یک فرآیند زمانبر است؛ به خصوص اگر رشته مورد نظر طولانی هم باشد. **مثال عملی:** در تکه کد زیر قصد داریم اعداد 1 تا 20000 را در یک رشته الحاق کنیم:

```
DateTime dt = DateTime.Now;
            string s = "";
        for (int index = 1; index <= 20000; index++)
        {
            s += index.ToString();
        }
            Console.WriteLine(s);
```

```
            Console.WriteLine(dt);
            Console.WriteLine(DateTime.Now);
            Console.ReadKey();
```

کد بالا تاز زمان نمایش کامل، بسته به قدرت سیستم ممکن است یکی دو ثانیه طول بکشد. حالا عدد را به 200000 تغییر دهید (یک صفر اضافه تر). برنامه را اجرا کنید و مجددا تست بزنید. در این حالت چند دقیقه ای بسته به قدرت سیستم زمان خواهد برد؛ مثلا دو دقیقه یا سه دقیقه یا کمتر و بیشتر.

عملیاتی که در حافظه صورت میگیرد این چند گام را طی میکند:

قسمتی از حافظه به طور موقت برای این دور جدید حلقه، گرفته میشود که به آن بافر میگوییم.

رشته قبلی به بافر انتقال میابد که بسته به مقدار آن زمان بر و کند است؛ 5 کیلو یا 5 مگابایت یا 50 مگابایت و ...

شماره تولید شده جدید به بافر چسبانده میشود.

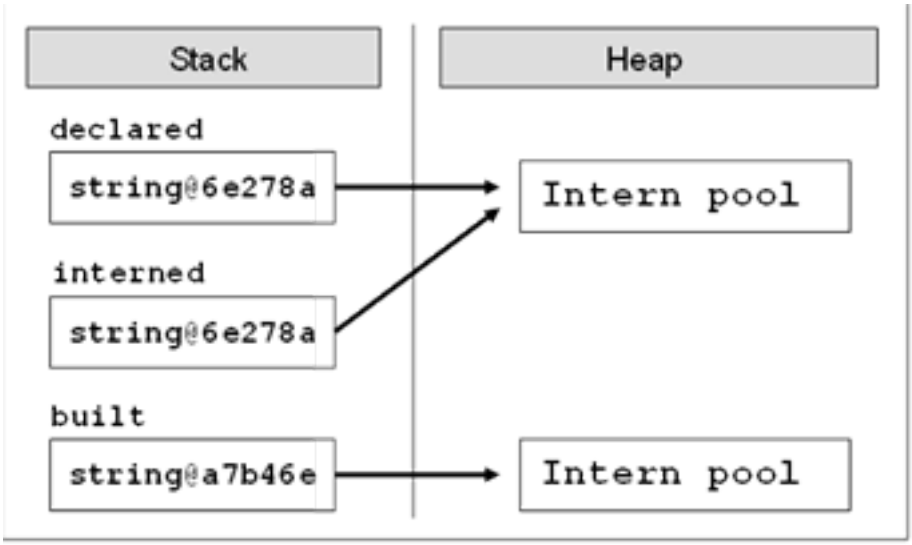
بافر به یک رشته تبدیل میشود وجایی برای خود در حافظه Heap میگیرد.

حافظه رشته قدیمی و بافر دیگر بلا استفاده شده‌اند و توسط GC پاکسازی میشوند که ممکن است عملیاتی زمان بر باشد.

String Builder

این کلاس ناپایدار و تغییر پذیر است. به کد و شکل زیر دقت کنید:

```
string declared = "Intern pool";
string built = new StringBuilder("Intern pool").ToString();
```



این کلاس دیگر مشکل الحاق رشته‌ها یا دیگر عملیات پردازشی را ندارد. بیایید مثال قبل را برای این کلاس هم بررسی نماییم:

```
StringBuilder sb = new StringBuilder();
      sb.Append("Numbers: ");

            DateTime dt = DateTime.Now;
        for (int index = 1; index <= 200000; index++)
        {
            sb.Append(index);
        }
            Console.WriteLine(sb.ToString());
            Console.WriteLine(dt);
            Console.WriteLine(DateTime.Now);
            Console.ReadKey();
```

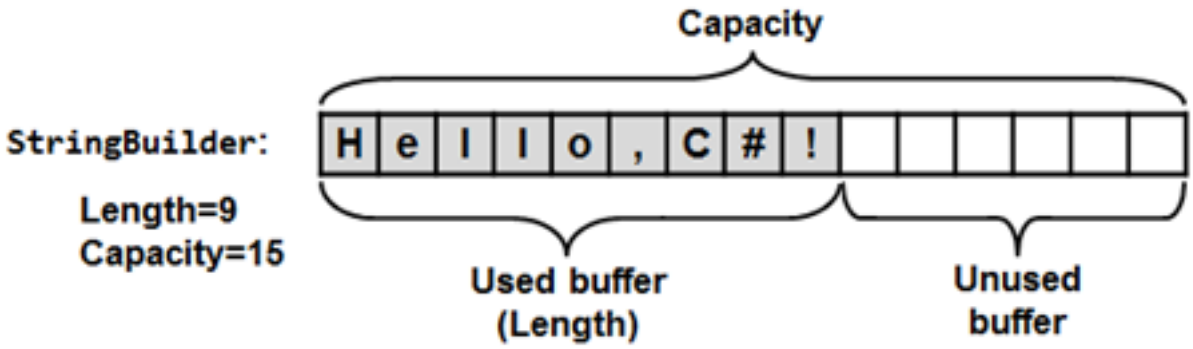
اکنون همین عملیات چند دقیقه‌ای قبل، در زمانی کمتر، مثلا دو ثانیه انجام میشود. *حال این سوال پیش می‌آید مگر کلاس stringbuilder چه میکند که زمان پردازش آن قدر کوتاه است؟*

همانطور که گفتیم این کلاس mutable یا تغییر پذیر است و برای انجام عملیات‌های ویرایشی نیازی به ایجاد شیء جدید در حافظه ندارد؛ در نتیجه باعث کاهش انتقال غیرضروری داده‌ها برای عملیات پایه‌ای چون الحاق رشته‌ها میگردد.

stringbuilder شامل یک بافر با ظرفیتی مشخص است (به طور پیش فرض 16 کاراکتر). این کلاس آرایه‌هایی از کاراکترها را پیاده سازی میکند که برای عملیات و پردازش‌هایش از یک رابط کاربرپسند برای برنامه نویسان استفاده می‌کند. اگر تعداد کاراکترها کمتر از 16 باشد مثلا 5 ، فقط 5 خانه آرایه استفاده میشود و مابقی خانه‌ها خالی میماند و با اضافه شدن یک کاراکتر جدید، دیگر شیء جدیدی در حافظه درست نمی‌شود؛ بلکه در خانه ششم قرار می‌گیرد و اگر تعداد کاراکترهایی که اضافه می‌شوند باعث شود از 16 کاراکتر رد شود، مقدار خانه‌ها دو برابر میشوند؛ هر چند این عملیات دو برابر شدن resizing عملیاتی کند است ولی این اتفاق به ندرت رخ می‌دهد.

کد زیر یک آرایه 15 کاراکتری ایجاد می‌کند و عبارت Hello C# را در آن قرار می‌دهد.

```
StringBuilder sb = new StringBuilder(15);
sb.Append("Hello, C#!");
```



در شکل بالا خانه هایی خالی مانده است Unused و جا برای کاراکترهای جدید به اندازه خانه‌های unused هست و اگر بیشتر شود همانطور که گفتیم تعداد خانه‌ها 2 برابر می‌شوند که در اینجا میشود 30.

استفاده از متد ایستای string.Format

از این متد برای نوشتن یک متن به صورت قالب و سپس جایگزینی مقادیر استفاده می‌شود:

```
DateTime date = DateTime.Now;
string name = "David Scott";
string task = "Introduction to C# book";
string location = "his office";

string formattedText = String.Format(
    "Today is {0:MM/dd/yyyy} and {1} is working on {2} in {3}.",
    date, name, task, location);
Console.WriteLine(formattedText);
```

در کد بالا ابتدا ساختار قرار گرفتن تاریخ را بر اساس الگو بین {} مشخص می‌کنیم و متغیر date در آن قرار می‌گیرد و سپس برای {1},{2},{3} به ترتیب قرار گیری آن‌ها متغیرهای name,last,location قرار میگیرند.

از ()ToString. هم می‌توان برای فرمت بندی خروجی استفاده کرد؛ مثل همین عبارت MM/dd/yyyy در خروجی نوع داده تاریخ و زمان.

نظرات خوانندگان

نویسنده: شهروز جعفری
تاریخ: ۱۳۹۳/۱۱/۲۹ ۱۸:۱۰

یک سوال منظور از Gac اینجا چیه؟

نویسنده: علی یگانه مقدم
تاریخ: ۱۳۹۳/۱۱/۲۹ ۱۸:۴۸

ممنون که گوشزد کردید؛ عذر میخوام. مطلب ویرایش شد. منظور GC بود. بنده اشتباها نوشتم GAC.