

Re: Fast string operations

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2005-06/msg00242.html>

- *From:* "Jonathan Allen" <x@xxx>
 - *Date:* Wed, 1 Jun 2005 13:55:31 -0700
-

> As far as performance, on some of our clients' instances, memory growth is
> rapid. It seems the more memory they have, the faster it grows which leads
> me to believe that the GC is being lax since it has so much free memory
> and doesn't see the need to aggressively collect memory. But it bothers
> our clients and they perceive this to be a memory leak.

If it was an ASP.net application, you could limit the amount of memory used before the application recycles itself. However, I don't know if that is an option for ASP. I think your goal of educating the client is probably the best bet.

May I suggest using the PerfMon tool to show them how often the GC runs and its effect on memory.

--
Jonathan Allen

"Chad Myers" <cmyers@xxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
[news:P3pne.13797\\$PR6.10264@xxxxxxxxxxxxxxxxxxxxxxxx](news:P3pne.13797$PR6.10264@xxxxxxxxxxxxxxxxxxxxxxxx)

> Nicholas,
>
> Thanks for the quick reply. Unfortunately I'm not using .NET 2.0 (yet!),
> so I can't use Generics.
>
> Would looping over chars like that slow things down significantly? Also,
> is the char[] for each string cached with the string, or is a new one
> created when you call things like ToCharArray() or foreach() on the string
> (not every loop iteration, but on the first iteration)? Wouldn't I just be
> replacing a new string instance with a new char[] and not get any net gain
> over just calling .Trim()?
>
> In your opinion, if I weren't against unsafe code, could I make this
> significantly faster, or would it not afford me much difference?
>
> As far as performance, on some of our clients' instances, memory growth is
> rapid. It seems the more memory they have, the faster it grows which leads
> me to believe that the GC is being lax since it has so much free memory

Re: Fast string operations

> and doesn't see the need to aggressively collect memory. But it bothers
> our clients and they perceive this to be a memory leak.
>
> I realize it's an education issue, but I want to make sure that I'm
> educating them correctly, as opposed to just making up a B.S. excuse and
> Jedi hand-waving about the GC stuff.
>
> Also, it's not an ASP.NET application, it's an ASP app that used to call
> into VB6 COM objects. We've replaced the VB6 objects with .NET objects
> exposing a "compatibility layer" that has a ComVisible API that is
> identical (though not binary compatible) with the old VB6 stuff.
> Late-bound clients don't know the difference other than a different ProgID
> for the COM objects.
>
> So we're dealing with the wkst GC, as far as I know (since only ASP.NET
> uses svr unless you host the CLR yourself, from what I understand). I'm
> not sure how I'd even do that in an ASP/COM-interop situation, but,
> assuming it's possible, would writing our own CLR host to use the svr GC
> help matters at all?
>
> Most of our clients' servers are dual-or-more processor boxes.
>
> Thanks again,
> Chad Myers
>
> "Nicholas Paldino [.NET/C# MVP]" <mvp@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote
> in message news:Ozam7ZuZFHA.580@xxxxxxxxxxxxxxxxxxxxxxxx
>> Chad,
>>
>> For the first scenario, your solution should give you an increase.
>>
>> For the second scenario, you should use reflection once to get a
>> reference to the internal static character array WhitespaceChars on the
>> string class. Then, you can write a method which will cycle through a
>> string passed to it, like so:
>>
>> public static bool TrimIsNullOrEmpty(string value)
>> {
>> // If null, then get out.
>> if (value == null)
>> {
>> // Return true.
>> return true;
>> }
>>
>> // Cycle through the characters in the string. If the character is
>> not found
>> // in the whitespace array, return false, otherwise, when done, return
>> true.
>> foreach (char c in value)
>> {

Re: Fast string operations

```
>> // If the character is not found in the WhitespaceArray, then
>> return
>> // false.
>> if (Array.IndexOf<char>(WhitespaceArray, char) == -1)
>> {
>> // Return false.
>> return false;
>> }
>> }
>> }
>>
>> // Return true, the string is full of whitespace.
>> return true;
>> }
>>
>> I used the generic version of the IndexOf method on the Array class in
>> order to eliminate boxing. Also, if you really want to squeeze out every
>> last bit of performance from this, you can take the WhitespaceArray and
>> use the characters as keys in a dictionary. The number of whitespace
>> characters is 25 (right now, that is). However, if your strings
>> typically are padded with spaces, then you could get a big speed boost by
>> copying the array initially, and then placing the space character as the
>> first element in the array (which would cause most of the calls to
>> IndexOf to return very quickly, probably quicker than a lookup in a
>> dictionary).
>>
>> I am curious though, are you seeing a performance issue, or do you
>> just see the numbers and are worried about them? ASP.NET applications
>> tend to get in a nice groove with the GC over time.
>>
>> Hope this helps.
>>
>> --
>> - Nicholas Paldino [.NET/C# MVP]
>> - mvp@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
>>
>> "Chad Myers" <cmyers@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
>> news:2rone.13780\$PR6.9706@xxxxxxxxxxxxxxxxxxxxxxxxxxxx
>>> I've been perf testing an application of mine and I've noticed that
>>> there are a lot (and I mean A LOT -- megabytes and megabytes of 'em)
>>> System.String instances being created.
>>>
>>> I've done some analysis and I'm led to believe (but can't yet
>>> quantitatively establish as fact) that the two basic culprits are a lot
>>> of calls to:
>>>
>>> 1.) if( someString.ToLower() == "somestring" )
>>>
>>> and
>>>
>>> 2.) if( someString != null && someString.Trim().Length > 0 )
>>>
```

Re: Fast string operations

>>>
>>> ToLower() generates a new string instance as does Trim().
>>>
>>> I believe that these are getting called many times and churning up a
>>> bunch of strings faster than the GC can collect them, or perhaps there's
>>> some weird interning/caching thing going on. Regardless, the number of
>>> string instances grows and grows. It gets bumped down occasionally, but
>>> it's basically 5 steps forward, 1 back.
>>>
>>> For reference, this is an ASP application calling into .NET ComVisible
>>> objects. So I assume this uses the workstation GC, right?
>>>
>>>
>>> Anyhow, so I think that I can solve problem (1) with String.Compare()
>>> which can perform in-place case-insensitive comparisons without
>>> generating new string instances.
>>>
>>> Problem (2), however, is more complicated. There doesn't appear to be a
>>> TrimmedLength or any type of method or property that can give me the
>>> length of a string, minus whitespace and without generating a new string
>>> instance, in the BCL.
>>>
>>> I suppose I could do some unsafe, or even unmanaged code (which is what
>>> MSFT did for all their string handling stuff inside System.String and
>>> using the COMString stuff), but I'd like to try to avoid that, or at
>>> least use a library that's already written and well tested.
>>>
>>> Any thoughts?
>>>
>>> Thanks in advance,
>>> Chad Myers
>>>
>>>
>>
>>
>
>

• References:

- ◆ **Fast string operations**
 ◇ From: Chad Myers
- ◆ **Re: Fast string operations**
 ◇ From: Nicholas Paldino [.NET/C# MVP]
- ◆ **Re: Fast string operations**
 ◇ From: Chad Myers

Re: Fast string operations

- Prev by Date: [*Re: Fast string operations*](#)
- Next by Date: [*Re: Free Blog software???*](#)
- Previous by thread: [*Re: Fast string operations*](#)
- Next by thread: [*Re: Fast string operations*](#)
- Index(es):
 - ◆ [*Date*](#)
 - ◆ [*Thread*](#)