.Net 2.0 - How efficient are Generic Lists?

Asked 16 years, 3 months ago Modified 13 years, 3 months ago

Viewed 2k times

9

I'm creating an app that holds loads of loads of user data in memory, and it's mostly keeping it all in List<T> structures (and some Dictionary<T,T> when I need lookup).

And I'm wondering...

How efficient are Lists? How much memory overhead do I get for each of them? (that is, memory space in addition to what the objects they contain would take) How much of a penalty do I pay every time I instance a new one?

Is there a more efficient way?

Dictionaries are just HashTables, right? Or are them a less efficient data structure?

I'd like to use Arrays, but I have the typical problem of adding and removing things all the time from them, so having to grow / shrink them would be a pain.

Any ideas/suggestions?

Edit: I know my basic data structures 101, and why a Linked List is better for adding/removing, and a HashTable is better for Random Access.

I'm mostly concerned about .Net's idionsyncracies. How much memory each of these structure wastes, for example. And time wasted on initializing / killing them.

Things like, for example, if it takes a lot of time to instance/GC a List, but not much to clear it, maybe I should keep a little pool of Lists waiting for me, and clear them and send them back to the pool when done, instead of simply dereferencing them.

Or, if Hashtables are faster for access but waste a lot of memory, I might prefer to use Lists and traverse them, for small item counts.

And I'd also really like to focus on memory usage, since my app is hediously memory intensive (think memcached like)... Does anyone know where I can find such info?

.net performance generics list memory

Share

Improve this question

Follow

edited Oct 2, 2010 at 20:50

community wiki
4 revs
Daniel Magliola

Why are you resurrecting this topic now, over two years after you initially posted it? Note that by editing it, you bring it to the front page. Unless you want to get renewed interest in your question leave it as-is, warts and all. – Lasse V. Karlsen Oct 2, 2010 at 21:07

10 Answers

Sorted by: Highest score (default)

4

Maybe you should consider using some type of in-memory database if you have that much data that has to be held in the memory,

Share Improve this answer Follow

answered Aug 28, 2008 at 21:14

Vaibhav
11.4k ● 11 ● 53 ● 71

What kind of in-memory database are you thinking of? Datasets? My understanding is that they are as slow as hell... Or are you thinking about some kinf of out-of-process database, like a MySQL with a table in memory? (or memcached?) – Daniel Magliola Aug 28, 2008 at 21:16

First, if you are going to comment on an answer, use the "add comment" feature. Second, I suspect he's thinking about something like SQLite (sqlite.org). – chyne Jun 3, 2009 at 12:01

Lists are arrays underneath, so the performance hit of adding an item, unless it is at the end, will be very costly.

2

Otherwise they will be basically as fast as an array.

Share Improve this answer Follow

answered Aug 28, 2008 at 21:08

Nick
13.4k ● 17 ● 66 ● 100

2

List uses an array internally and Dictionary uses an hash table.

They are faster then the older non-generic classes ArrayList and HashTable because you don't have the cost of converting everything to/from object (boxing, unboxing and type checking) and because MS optimized them better then the old classes.

Share Improve this answer Follow

answered Aug 28, 2008 at 21:14

Nir
29.6k ● 11 ● 68 ● 104

2

If you need efficiency in inserting or removing at random places in the list there is a LinkedList data structure - the MSDN Article gives details. Obviously being a linked list random access isn't efficient.

Share Improve this answer Follow

answered Aug 28, 2008 at 21:18

ljs
37.8k ● 36 ● 109 ● 124

I'm always adding to the end of the list. Many times, I'm removing from the middle of some of the biggest lists. Besides insertion/removal times, how do Linked Lists compared to regular Lists? (memory, time to traverse, etc) – Daniel Magliola Aug 28, 2008 at 21:21

2

The LinkedList object would take less time to add to and remove from because of the nature of linked lists. When you add an element it does not have to resize an array like a normal list does. Other than that improvement I would suspect that the LinkedList would perform about the same as a normal List.

See this on Wikipedia: [Linked Lists vs. Arrays]

Share Improve this answer Follow

answered Aug 28, 2008 at 21:29

Jesse Dearing
2,261 ● 18 ● 20

But doesn't .Net's LinkedList wrap each of my objects in a new object? Won't that waste a lot of memory? I'm really concerned about the memory needs this app will have, I would like to keep it as low as possible. – Daniel Magliola Aug 28, 2008 at 21:31

@Daniel: Being linked lists they are efficient at random insertions and deletions, either lack or are inefficient at random access (I've not played with them so I don't know which) and are fine being traversed front to back. If you require random access then I reckon List<T> or Dictionary<T, T> will be fine, depending on whether you want to access memebers by index or a value. – ljs Aug 28, 2008 at 21:37

It does wrap the object in a LinkedListNode object, but that object consists of 4 properties but 3 of those are just references to other objects taking up a very small amount of memory and the 4th is your actual object. You can always write your own linked list to reduce any overhead added by the .NET type. I originally said to use a struct, but that probably work as well in C#. – Jesse Dearing Aug 28, 2008 at 21:39

2

If you really want to see all the gory details of how List<> and Dictionary<,> are implemented, use the wonderfully useful .NET Reflector.

See also the documentation for the excellent C5 Generic Collection Library, which has very good implementations of a number of collection types missing from the BCL.

Share Improve this answer Follow

answered Aug 31, 2008 at 2:42

McKenzieG1
14.2k ● 7 ● 38 ● 42

1

If you are concerned about memory usage, the real key is to store your array on disk and map just the parts you need into memory at that time.

The key is to use FILE_FLAG_NO_BUFFERING and always read/write exactly one sector's worth of data.

Share Improve this answer

Follow

answered Aug 28, 2008 at 21:39

Nick

13.4k ● 17 ● 66 ● 100

Unfortunately, I do need to keep everything in memory, I think... Most of it for sure... But your answer did open me to a lot of interesting ideas. Maybe I can keep some of the stuff I use less frequently on disk. Any ideas on letting Windows PAGE that automatically to HD? For example, could I keep my less-frequently-used data in a separate process, and somehow give that other process less "memory-priority" than the main one? That way, when the system runs out of memory, it can page THAT less priority stuff first, and keep my most important things in RAM? Am I daydreaming? – Daniel Magliola Aug 28, 2008 at 21:53

You can have your less frequently used data be more likely to be paged, by using it less frequently. – Jon Hanna Nov 11, 2010 at 14:35

1

I think the two-process thing might be overkill; plus the interprocess communication will likely have some slowness (although I've never tried such a thing so take my opinion of it as a grain of salt). I work on a data-driven application where each data unit is tiny, but we may have upwards of a billion data units at any given time. The method we use is basically:

- Everything resides on disk, no matter what
- Data is blocked into "chunks"; each chunk knows when it was last accessed

- Chunks are dragged up from disk into memory when they are needed
- A low-priority thread monitors memory usage and deletes the least recently used stuff

In other words, it's a homebrew caching scheme. The benefit is you can control with pinpoint accuracy what data is in memory, which you cannot if you rely on the OS paging scheme. If some commonly used variable ends up mixed in with your data on a page, that page will be repeatedly hit and prevent it from going to disk. If you design into your application an accommodation that some data requests will take longer than others, then this will work pretty well. Particularly if you know what chunks you will need ahead of time (we don't).

Keep in mind that everything in a .NET app has to fit within 2 GB of memory, and because of the way the GC works and the overhead of you app, you actually probably have somewhat less than that to work with.

To spy on exactly what your heap looks like and who is allocating, use the **CLR profiler**: http://www.microsoft.com/downloads/details.aspx?familyid=86ce6052-d7f4-4aeb-9b7a-94635beebdda&displaylang=en

Share Improve this answer Follow

answered Aug 28, 2008 at 22:18

Nick
13.4k ● 17 ● 66 ● 100

Are .Net processes also limited by 2Gb in Windows x64? Uh... Oh... I was counting on the opposite :-S
– Daniel Magliola Aug 28, 2008 at 22:31

I think the x64 will let you address 4 GB, I did not consider that. However I would not count on avoiding OutOfMemory all the way up to that limit, since the GC will not perfectly "pack" your objects into that space (heap fragmentation). – Nick Aug 28, 2008 at 23:01

To answer my own question: No, they're not.
– Daniel Magliola Jun 3, 2009 at 15:33

0

I wouldn't move a finger until there's some performance problem and a profiler showed you were it is. Then you'll have a definitive problem to solve and it'll be much easier.

Share Improve this answer Follow

answered Jun 3, 2009 at 5:55

Pablo Fernandez
287k ● 139 ● 401 ● 641

0

The .Net List doesn't use a linked list. It is an array, it starts with 4 positions by default and I think it doubles in size as you add things. So performance can vary a bit depending on how you use it.

If your using VS 2008 run the profiler before you get too far down this rat hole. When we started actually looking at where we were losing time it didn't take long for use to figure out that debating the finer points of linked lists just really didn't matter.

Share Improve this answer Follow

edited Sep 15, 2011 at 16:57

Gumbo
655k ● 112 ● 790 ● 851

answered Aug 28, 2008 at 21:46

Dan Blair
2,381 ● 3 ● 21 ● 32

Good idea about the profiler. Can I get away with running that against the live process in the server, without installing the whole of VS 2008 in it? Maybe a small program I can put in in there that'll give me a log? Any tools similar to the profiler that'll let me see what is my memory used on? (for example, how many instances of each Class, or how many bytes in instances of each Class) – Daniel Magliola Aug 28, 2008 at 21:56

Regarding tools: see stackoverflow.com/questions/134086. I personally had success with WinDbg + SOS. – Constantin Sep 27, 2008 at 15:25