Optimization techniques in C# [closed]

Asked 15 years, 11 months ago Modified 10 years, 9 months ago

Viewed 15k times

12

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 7 years ago.

Improve this question

I am wondering what kind of optimization techniques people often use nowadays. I have seen people do caching all the time with dictionary and all. Is the trading space for speed the only way to go?

c# .net optimization

Share

Improve this question

Follow

asked Jan 28, 2009 at 0:50

Tom

948 ● 2 ● 7 ● 14

Can you be any more specific about the scenario you have in mind? "Optimization" is a terribly broad category. – Larsenal Jan 28, 2009 at 0:53

Yeah, way to broad. Do you have a specific question? – Ed Swangren Jan 28, 2009 at 0:56

Just wondering if there is any way to optimize the code other than using caching with dictionaries. – Tom Jan 28, 2009 at 1:19

6 Answers

Sorted by: Highest score (default)

5

Really it's about your choice in algorithms. Usually there is no "silver bullet" for optimization.

For example, using a `StringBuilder` instead of concatenation can make your code significantly faster, but there is a tradeoff. If you aren't concatenating huge sets of strings, the memory and time it takes to initialize `StringBuilder` is worse than just using regular concatenation. There are a lot of examples of this throughout the framework, such as dictionary caching as you mentioned in your question.

The only general optimization you can really *learn* and apply to your coding throughout your day is the performance hit from boxing/unboxing (heap vs. stack). To do this you need to learn what it's about and how to avoid, or reduce the need to do it.

Microsoft's MSDN documentation has 2 articles on performance that give a lot of good general purpose techniques to use (they're really just different versions of the same article).

- http://msdn.microsoft.com/en-us/library/ms173196.aspx
- http://msdn.microsoft.com/en-us/library/ms173196(VS.80).aspx

Share Improve this answer

Follow

answered Jan 28, 2009 at 2:25

Dan Herbert

103k ● 51 ● 192 ● 221

StringBuilder is an awesome easy to implement optimizer when working with iterative building. – Pat Jan 28, 2009 at 3:17

5

I will suggest below

1. Knowing when to use StringBuilder

You must have heard before that a `StringBuilder` object is much faster at appending strings together than normal string types.

```
The thing is StringBuilder is faster mostly with big strings. This means if you have a loop that will add to a single string for many iterations then a StringBuilder class is definitely much faster than a string type.
```

`However if you just want to append something to a string a single time then a StringBuilder class is overkill. A simple string type variable in this case improves on resources use and readability of the C# source code.`

Simply choosing correctly between StringBuilder objects and string types you can optimize your code.

2. Comparing Non-Case-Sensitive Strings

In an application sometimes it is necessary to compare two string variables, ignoring the cases. The tempting and traditionally approach is to convert both strings to all lower case or all upper case and then compare them, like such:

```
str1.ToLower() == str2.ToLower()
```

However repetitively calling the function ToLower() is a bottleneck in performace. By instead using the built-in string.Compare() function you can increase the speed of your applications.

To check if two strings are equal ignoring case would look like this:

```
string.Compare(str1, str2, true) == 0 //Ignoring cases
```

The C# string.Compare function returns an integer that is equal to 0 when the two strings are equal.

3. Use string.Empty

This is not so much a performance improvement as it is a readability improvement, but it still counts as code optimization. Try to replace lines like:

```
if (str == "")
```

with:

```
if (str == string.Empty)
```

This is simply better programming practice and has no negative impact on performance.

Note, there is a popular practice that checking a string's length to be 0 is faster than comparing it to an empty string. While that might have been true once it is no longer a significant performance improvement. Instead stick with string.Empty.

4. Replace ArrayList with List<>

ArrayList are useful when storing multiple types of objects within the same list. However if you are keeping the same type of variables in one ArrayList, you can gain a performance boost by using List<> objects instead.

Take the following ArrayList:

```
ArrayList intList = new ArrayList();
intList.add(10);
return (int)intList[0] + 20;
```

Notice it only contains intergers. Using the List<> class is a lot better. To convert it to a typed List, only the variable types need to be changed:

```
List<int> intList = new List<int>();

intList.add(10)

return intList[0] + 20;
```

There is no need to cast types with List<>. The performance increase can be especially significant with primitive data types like integers.

5. Use && and || operators

When building if statements, simply make sure to use the double-and notation (&&) and/or the double-or notation (||), (in Visual Basic they are AndAlso and OrElse).

If statements that use & and | must check every part of the statement and then apply the "and" or "or". On the other hand, && and || go thourgh the statements one at a time and stop as soon as the condition has either been met or not met.

Executing less code is always a performace benefit but it also can avoid run-time errors, consider the following C# code:

```
if (object1 != null && object1.runMethod())
```

If object1 is null, with the && operator, object1.runMethod()will not execute. If the && operator is replaced with &, object1.runMethod() will run even if object1 is already known to be null, causing an exception.

6. Smart Try-Catch

Try-Catch statements are meant to catch exceptions that are beyond the programmers control, such as connecting to the web or a device for example. Using a try statement to keep code "simple" instead of using if statements to avoid error-prone calls makes code incredibly slower. Restructure your source code to require less try statements.

7. Replace Divisions

C# is relatively slow when it comes to division operations. One alternative is to replace divisions with a multiplication-shift operation to further optimize C#. The article explains in detail how to make the conversion.

REFERENCE

Share Improve this answer Follow

answered Mar 10, 2014 at 6:03

Amit

15.4k ● 8 ● 49 ● 69

4

There are often problems with algorithms as well, usually when something expensive is done inside of a loop. Generally, the first thing you do is profile your application,

which will tell you the slowest part(s) of the application. Generally, what you do to speed up your application depends upon what you find. For example, if your application mimics a file system, it may be that you're calling the database recursively to travel up the tree (for instance). You may optimise that case by changing those recursive calls into one flattened database call that returns all of the data in one call.

Again, the answer is, as always, 'it depends'. However, more examples and advice can be found in Rico Mariani's blog (browse back a few years, as his focus has shifted):

Share Improve this answer Follow

edited Jan 29, 2009 at 3:14

answered Jan 28, 2009 at 2:13

tsimon

8,010 ● 2 ● 33 ● 44

Profiling you specific application is probably the only good advice. Profile, see what's slow (or memory hogging), and try to cut it down. – Kibbee Jan 28, 2009 at 2:15

I am actually looking for good examples. People use memoization (caching again) for recursions. +1 for database example. – Tom Jan 28, 2009 at 16:18

2

Depends on a lot of things, really.

As an example, when memory becomes an issue and a lot of temporary objects are being created I tend to use object pools. (Having a garbage-collector is not a reason to not take care of memory allocation). If speed is what matters then I might use unsafe pointers to work with arrays.

Either way, if you find yourself struggling too much with optimization techniques in a c#/.net application you probably chose the wrong language/platform.

Share Improve this answer

Follow

answered Jan 28, 2009 at 1:40

Trap

12.4k ● 15 ● 57 ● 69

1

In general, make sure you understand the time complexity of different algorithms, and use that knowledge to choose your implementations wisely.

For .NET in particular, this article goes into great detail about optimizing code deployed to the CLR (though it's also relevant for Java, or any other modern platform), and is one of the best guides I've ever read:

http://msdn.microsoft.com/en-us/library/ms973852.aspx

To distill the article into one sentence: Nothing affects the speed of a .NET application (with sensible algorithms)

more than the memory-footprint of its objects. Be very careful to minimize your memory consumption.

Share Improve this answer

Follow

answered Jan 28, 2009 at 2:02

benjismith

16.7k ● 9 ● 61 ● 83

1

I would recommend Effective C# by Bill Wagner (first edition and second edition). He goes through a number of language constructs and techniques and explains which ones are faster and why. He touches on a lot of best practices as well.

More often than not, however, optimizing your algorithm will give you far better results than using any kind of language / optimization technique.

Share Improve this answer

Follow

answered Jan 28, 2009 at 4:28

pbz

9,075 ● 15 ● 59 ● 73

Exactly what I would have said-- this is where I'd go on my first pass. – mmr Jan 28, 2009 at 4:40