C# compiler and caching of local variables

Asked 15 years, 11 months ago Modified 3 months ago Viewed 2k times

2

EDIT: Oops - as rightly pointed out, there'd be no way to know whether the constructor for the class in question is sensitive to when or how many times it is called, or whether the object's state is changed during the method, so it would have to be created from scratch each time. Ignore the Dictionary and just consider delegates created in-line during the course of a method :-)

Say I have the following method with Dictionary of Type to Action local variable.

```
void TakeAction(Type type)
{
    // Random types chosen for example.
    var actions = new Dictionary<Type, Action>()
    {
        {typeof(StringBuilder), () =>
            {
                // ..
            }},
         {typeof(DateTime), () =>
            {
                // ..
            }}
    };

    actions[type].Invoke();
}
```

The Dictionary will always be the same when the method is called. Can the C# compiler notice this, only create it once and cache it somewhere for use in future calls to the method? Or will it simply be created from scratch each time? I know it could be a field of the containing class, but it seems neater to me for a thing like this to be contained in the method that uses it.

c# optimization

Share

Improve this question

Follow

edited Aug 25 at 1:58

Jonas

128k ● 100 ● 326 ● 405

asked Jan 15, 2009 at 10:43

xyz

27.8k ● 30 ● 106 ● 126

4 Answers

Sorted by: Highest score (default)

7

How should the C# compiler know that it's "the same" dictionary every time? You explicitly create a new dictionary every time. C# does not support static local variables, so you have to use a field. There's nothing wrong with that, even if no other method uses the field.

It would be bad if the C# compiler did things like that. What if the constructor of the variable uses random input? :)

Share Improve this answer Follow

answered Jan 15, 2009 at 10:50

OregonGhost
23.7k ● 7 ● 75 ● 109

7

Short answer: no.

Slightly longer answer: I believe it will cache the result of creating a delegate from a lambda expression which doesn't capture anything (including "this") but that's a pretty special case.

Correct way to change your code: declare a private static readonly variable for the dictionary.

```
private static readonly Dictionary<Type,Action> Actions =
    new Dictionary<Type, Action>()
{
    { typeof(StringBuilder), () => ... },
    { typeof(DateTime), () => ... },
}

void TakeAction(Type type)
{
    Actions[type].Invoke();
}
```

Share Improve this answer Follow

answered Jan 15, 2009 at 10:56

Jon Skeet
1.5m ● 889 ● 9.3k ● 9.3k

Yep - I knew that was the alternative. The caching of delegates dependant on whether they capture anything makes a lot of sense - thanks. – xyz Jan 15, 2009 at 11:10

I wouldn't rely on my memory for this one - I haven't checked the spec. – Jon Skeet Jan 15, 2009 at 11:13

For any compiler to be able to do this, it would have to have some way to have guarantees for the following issues:

2

- Constructing two objects the exact same way produces identical objects, in any way, except for their location in memory. *This would mean that the object constructed the second time would be no different from the first one, as opposed to say, caching an object of the Random typ.e*
- Interacting with the object does not change its state. *This would mean that caching the object would be safe and would not change the behavior of subsequent calls. This would for instance rule out modifying the dictionary in any way.*

The reason for this is that the compiler would have to be able to guarantee that the object it constructed the first time would be equally usable the next time around without having to recreate it.

Now, that C# and .NET does not have mechanisms for making these guarantees is probably not the reason why support for this kind of optimization isn't done by the compiler, but these would have to be implemented first. There could also be other such guarantees the compiler would need to have before it could do it that I don't know of.

The change that Jon Skeet has suggested is basically the way to say that *I know that these two guarantees hold for my code*, and thus you take control over the situation yourself.

Share Improve this answer Follow

answered Jan 15, 2009 at 11:36

Lasse V. Karlsen
391k ● 106 ● 646 ● 844

That dictionary will be created anew each time; otherwise, for example, you could put other things into the dictionary, and the intent would be lost.

1

Share Improve this answer Follow

answered Jan 15, 2009 at 10:56

Marc Gravell
1.1m ● 273 ● 2.6k ● 3k

I'm ashamed that I overlooked that! I imagine it would be difficult for the compiler to know whether or not subsequent uses of an object changed its state. – xyz Jan 15, 2009 at 11:20