What sorts of things should I do to make a performant and robust reflection cache?

Asked 16 years, 2 months ago Modified 16 years, 2 months ago

Viewed 234 times

2

In .NET 3.5, I'm going to be working with System.Reflection to use AOP (probably in the context of Castle's Windsor Interceptors) to do things like define which security actions need to be performed at the method level, etc. I have heard that some parts of Reflection are slow (I've read the MSDN article around it), and would like to cache these parts (when I get closer to production code, at any rate). I would like to validate my approach:

- cache key is {type} + {case sensitive method name} + {list of parameter types}
- cache key objects can be compared via an Equals operation
- cache payload is a {MethodInfo} + {list of custom-attributes defined on the method}
- cache is injected to my interceptors via constructor injection
- cache can be maintained for a long time (based on the assumption that I'm not going to be writing self-

modifying code ;-))

Update:

I'm not intending to call methods via Reflection something I'm writing myself; just (at the moment) look up attributes on the ones I want to inject functionality into, where the attributes define the behaviour to inject. My interceptors at the moment will be using Castle's Windsor IInterceptor mechanism until I notice a reason to change it.

c# .net reflection caching

Share

Improve this question

Follow

edited Oct 12, 2008 at 11:59

asked Oct 11, 2008 at 8:56

Peter Mounce

4,225 ● 4 ● 36 ● 67

2 Answers

Sorted by: Highest score (default)

3

Explicitly calling a MethodInfo is indeed slow - but you can make it much, *much* faster if you convert it into a delegate. See this blog post for example. That doesn't help in terms of finding methods etc of course, but if you're going to call the method repeatedly it's worth bearing in mind.

The cache key sounds easy enough to build - types and strings compare nice and easily. Values are always relatively simple :)

Once built, is the cache going to be read-only? Can you separate out the phases so that you can guarantee it won't be read before being fully built? If so, you should be able to get away without any explicit locking - basically a dictionary from your custom key type to your custom value type.

Share Improve this answer

Follow

answered Oct 11, 2008 at 9:07

Jon Skeet

1.5m ● 889 ● 9.3k ● 9.3k

> At the moment I'm thinking yes, the cache will be read-only and forever-persistent - I don't think I'll be changing the "extension-point is here" attributes at run-time (though the *actions* that are provoked may be more dynamic).
> – Peter Mounce Oct 12, 2008 at 12:01

1

I'd agree with most of Jon's post - a small note re the dictionary: from a performance perspective, you might want to benchmark this vs just a flat list. Last time I did a benchmark (dictionary vs flat list, checking each item until match found), the dividing point (for read access) was around 150 items; below that, the list was quicker (just through simplicity). But do your own tests... (I don't have the numbers to hand to prove one way or another).

Depending on the code, you might be able to use generics to further split the data - i.e. a Cache so that all the info for type T is in one place, populated in the static ctor for Cache. This may or may not be possible, depending on the architecture.

Finally, it might or might not fit, but there are existing AOP frameworks like PostSharp that might help simplify the injection points.

Re the generic point - it would be fairly easy to (in your init code) create a typed delegate to a method on the Cache method, to reduce the amount of data it needs to scan through - just a bit of Type.MakeGenericType and and Delegate.CreateDelegate - after this point, the code just knows about your Func<...> delegate, and doesn't need to care about the implementation.

answered Oct 11, 2008 at 14:44

Marc Gravell

1.1m ● 272 ● 2.6k ● 3k