Ruby - Feature #7816

Don't invalidate method caches when defining a new method on a class without subclasses

02/10/2013 02:13 AM - Anonymous

Status:	Closed	
Priority:	Normal	
Assignee:	ko1 (Koichi Sasada)	
Target version:	2.6	

Description

=begin
Attached is a patch that avoids incrementing the VM state version when defining a method and these conditions are true:

- The method is not a redefinition of an existing method in the same class or any superclass
- The class has no subclasses
- The class is not a module

This means that defining singleton methods on objects no longer invalidates every method cache. This will significantly improve performance of code that defines singleton methods at runtime (eg. when using OpenStruct)

In my testing, a fresh Rails app boots about 15% faster (~1.7 sec down to ~1.4 sec).

This controller action can do ~440 requests per second with my patch, compared to ~320 requests per second without my patch.

class HomeController < ApplicationController
def index
OpenStruct.new a: 1, b: 2
render text: "home"
end
end

Of course these numbers will vary between apps, but I think this is a good start in improving the performance of a very common use case in Ruby.
=end

History

#1 - 02/10/2013 02:53 AM - ko1 (Koichi Sasada)

(2013/02/10 2:13), charliesome (Charlie Somerville) wrote:

> In my testing, a fresh Rails app boots about 15% faster (~1.7 sec down to ~1.4 sec).
>
> This controller action can do ~440 requests per second with my patch, compared to ~320 requests per second without my patch.

charliesome++

--
// SASADA Koichi at atdot dot net

#2 - 02/10/2013 09:46 AM - marcandre (Marc-Andre Lafortune)

- Subject changed from Don't invalidate method caches when defining a new method on a class without subclasses to Don't invalidate method caches when defining a new method on a class without subclasses

Speed boosts sounds awesome, especially the 15% rails bootup time, since the example is a bit contrived.

It would have been great to get this in 2.0.0

I'd split the new API from the patch; personally I'm not convinced of the usefulness of Class#has_subclass? or of RubyVM.state_version

#3 - 02/10/2013 10:19 AM - mame (Yusuke Endoh)

marcandre (Marc-Andre Lafortune) wrote:

> It would have been great to get this in 2.0.0

Do not kamikaze!

--
Yusuke Endoh mame@tsg.ne.jp

#4 - 02/10/2013 10:33 AM - Anonymous

marcandre (Marc-Andre Lafortune) wrote:

> I'd split the new API from the patch; personally I'm not convinced of the usefulness of Class#has_subclass? or of RubyVM.state_version

I'm not particularly attached to Class#has_subclass? - I don't care if that stays or goes. I figured that if we have the information, we may as well let the user access it.

I think RubyVM.state_version could be very useful for performance profiling. I work on a Rails app that invalidates the method cache ~200 times per request (mainly due to OpenStruct). I think this patch will lower that number somewhat, but I want to keep this API in so I (and others) can monitor cache invalidations and try to eliminate the remaining invalidations.

#5 - 02/10/2013 09:34 PM - nobu (Nobuyoshi Nakada)

=begin
Nice.

My thoughts are:

- RCLASS_INHERITED flag should go to internal.h.
- Class#has_subclass? is not only useless but harmful, it mimics users when subclasses are removed.
- RubyVM.state_version seems useless also, and should be hidden.
- why rb_method_entry() ignores the cache for an undefined method?
- what are extra parens around RB_TYPE_P() in rb_method_entry_make().
- what's inst in .gitignore.
  =end

#6 - 02/10/2013 09:58 PM - Anonymous

Thanks for the feedback nobu.

> - RCLASS_INHERITED flag should go to internal.h.

I think it should stay in include/ruby/ruby.h where all the other flags are defined. This way if someone adds a new class flag, they do not accidentally also pick FL_USER5 because they did not see it already in use.

> - Class#has_subclass? is not only useless but harmful, it mimics users when subclasses are removed.

Fair enough, I'll update my patch and remove it.

> - RubyVM.state_version seems useless also, and should be hidden.

I don't think it is useless. It will be useful for performance tuning. Right now there is no way to tell if the method caches have been cleared, which makes profiling and tuning harder.

> - why rb_method_entry() ignores the cache for an undefined method?

Here is an example:

```
class Foo
unless method_defined?(:bar)
def bar
end
end
end
```

If cache entries for undefined methods were not ignored, 'bar' would be defined, but calling it would raise NoMethodError, because the global cache thinks it is not defined.

> - what are extra parens around RB_TYPE_P() in rb_method_entry_make().

Whoops, this is my mistake.

- what's inst in .gitignore.

Ditto, I use './configure --prefix=pwd/inst', but I forgot to stash before creating the patch.

#7 - 02/10/2013 10:13 PM - Anonymous

- File feature-7816-v2.patch added

#8 - 02/10/2013 10:34 PM - nobu (Nobuyoshi Nakada)

charliesome (Charlie Somerville) wrote:

> I think it should stay in include/ruby/ruby.h where all the other
> flags are defined. This way if someone adds a new class flag, they
> do not accidentally also pick FL_USER5 because they did not see it
> already in use.

Rather I think other RMODULE_* flags also should go to internal.h.

- Class#has_subclass? is not only useless but harmful, it mimics
  users when subclasses are removed.

> Fair enough, I'll update my patch and remove it.

Or, add subclass count in rb_classext_t.

- what's inst in .gitignore.

> Ditto, I use './configure --prefix=pwd/inst', but I forgot to
> stash before creating the patch.

I use dotted directories, e.g., .x86_64-linux.

#9 - 02/16/2013 06:31 PM - Anonymous

- File feature-7816-v3.patch added

- Subject changed from Don't invalidate method caches when defining a new method on a class without subclasses to Don't invalidate method caches when defining a new method on a class without subclasses

I have updated my patch to not clear the cache when a class is garbage collected.

My reasoning is that if a class is garbage collected, then no objects of that class could possibly exist, so the method cache would never be hit.

#10 - 02/16/2013 07:32 PM - funny_falcon (Yura Sokolov)

charliesome (Charlie Somerville) wrote:

> I have updated my patch to not clear the cache when a class is garbage collected.
>
> My reasoning is that if a class is garbage collected, then no objects of that class could possibly exist, so the method cache would never be hit.

That is not the true: occasionally a new class could be placed at the same object slot. Then method cache item will be considered as cache-hit, but will point to wrong direction.

#11 - 02/18/2013 09:18 AM - ko1 (Koichi Sasada)

- Subject changed from Don't invalidate method caches when defining a new method on a class without subclasses to Don't invalidate method caches when defining a new method on a class without subclasses

- Assignee set to ko1 (Koichi Sasada)

#12 - 12/07/2013 03:05 PM - Anonymous

- Status changed from Open to Closed

Superseded by klasscache

#13 - 12/09/2013 03:18 PM - headius (Charles Nutter)

Is "klasscache" a reference to some other patch/issue?

Am I understanding correctly when I say that JRuby would not benefit from this sort of fix? In JRuby there is no global cache, so the only cache damage caused by a singleton class is that call sites encountering it will have to cache for the first time or fail a type test if they already cached a method. That's not great, but it's not a global invalidation event either.

#14 - 12/09/2013 03:21 PM - Anonymous

@headius (Charles Nutter): klasscache refers to #8426/r42822 which implements JRuby style method cache invalidation.

#15 - 12/09/2013 09:43 PM - headius (Charles Nutter)

Righto, thanks!

Files

dont-invalidate-method-cache-when-defining-new-singleton-methods.patch	4.55 KB	02/10/2013	Anonymous
feature-7816-v2.patch	3.41 KB	02/10/2013	Anonymous
feature-7816-v3.patch	3.72 KB	02/16/2013	Anonymous