Ruby - Feature #17288

Optimize __send__ call with a literal method name

10/27/2020 08:32 AM - mrkn (Kenta Murata)

Status:	Assigned
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	

Description

I made a patch to optimize a __send__ call with a literal method name. This optimization replaces a __send__ method call with a send instruction. The patch is available in this pull-request.

By this change, the redefined __send__ method is no longer called when it is called by a literal method name. I guess it is no problem because the following warning message is displayed for a long time.

```
$ ruby -e 'def __send__; end'
-e:1: warning: redefining `__send__' may cause serious problems
```

This change makes the optimized case x5~x6 faster. The benchmark result is below:

```
$ make benchmark COMPARE_RUBY="../../ruby/build-o3/ruby" ITEM=vm_send.yml
(snip)
# Iteration per second (i/s)

|             |compare-ruby|built-ruby|
|:------------|-----------:|---------:|
|vm_send      |     18.536M|  113.778M|
|             |           -|     6.14x|
|vm_send_var  |     18.085M|   16.595M|
|             |       1.09x|         -|
```

Related issues:

Is duplicate of Ruby - Feature #17291: Optimize __send__ call	**Assigned**

History

#1 - 10/27/2020 06:35 PM - Eregon (Benoit Daloze)

I think obj.send(:foo) should be optimized too, to not create a an arbitrary performance difference between send and __send__.
send is also used ~15x used more frequently than __send__ in gems, so that would be even more valuable.

Of course, optimizing send means checking that obj.method(:send) corresponds to Kernel#send (somewhat similar to +).

@matz (Yukihiro Matsumoto) suggested an operator for __send__ in https://twitter.com/yukihiro_matz/status/1320496276476063744, that might be a cleaner solution.
OTOH, it would not benefit existing code and not be adopted for a few years.

FWIW in TruffleRuby I've been thinking to optimize all 3 sends (__send__, send, public_send), since they are hidden from backtraces, etc, and optimizing them makes it simpler to remove them from backtraces, etc. It would also reduce the overhead in interpreter for send/__send__(:private_method) calls (in JITed code, those have no overhead). I would still check the correct method in all cases, changing semantics should IMHO be the last resort.

#2 - 10/28/2020 08:25 AM - shyouhei (Shyouhei Urabe)

Hello, I'm against this optimisation. obj.__send__(:method) should just be written as obj.method.

Not against the ability to write obj.__send__(:method), but obj.method must be the preferable way and thus must be the fastest thing.

This patch adds complexity just to encourage people to follow the wrong way. This is -1 to me.

#3 - 10/28/2020 08:34 AM - zverok (Victor Shepelev)

@shyouhei (Shyouhei Urabe) what about private methods?

#4 - 10/28/2020 09:58 AM - shyouhei (Shyouhei Urabe)

zverok (Victor Shepelev) wrote in #note-3:

> @shyouhei (Shyouhei Urabe) what about private methods?

Private methods shall not be called at the first place. Period. That is breaking encapsulation.

Again I'm not against the ability to do such things. But we must not encourage people.

#5 - 10/28/2020 10:09 AM - Eregon (Benoit Daloze)

Here are the first 1000 .send() usages in gems:
https://gist.github.com/eregon/21c8f14c478089c1a9295c21661583a9
420 of them use a literal Symbol for the first argument.

> Private methods shall not be called at the first place. Period.

It's not as simple, there are many cases where it's reasonable to call private methods.
For instance things like Module#{include,prepend,alias_method,define_method} used to be private, and Module#remove_const still is.
Some gems call their own private methods in tests, which seems fair enough.

shyouhei (Shyouhei Urabe) wrote in #note-2:

> Not against the ability to write obj.__send__(:method), but obj.method must be the preferable way and thus must be the fastest thing.

I would think nobody prefers obj.__send__(:some_method) to obj.some_method if some_method is public, so it seems a non-issue to me.
And anyway obj.some_method would always be as fast or faster than obj.__send__(:some_method), never slower (that would be a performance bug).

#6 - 10/28/2020 12:32 PM - shyouhei (Shyouhei Urabe)

Eregon (Benoit Daloze) wrote in #note-5:

> Here are the first 1000 .send() usages in gems:
> https://gist.github.com/eregon/21c8f14c478089c1a9295c21661583a9
> 420 of them use a literal Symbol for the first argument.

So? I don't think we should follow that. If people misunderstand what an OOPL is, we would better not confirm that.

> Private methods shall not be called at the first place. Period.

> It's not as simple, there are many cases where it's reasonable to call private methods.
> For instance things like Module#{include,prepend,alias_method,define_method} used to be private, and Module#remove_const still is.

They are/were private for reasons. Private methods can be made public later, but that must have been done with really careful considerations by the author. Not by callee people.

> Some gems call their own private methods in tests, which seems fair enough.

Testing private methods! That itself has a bunch of discussions.

But even if we put those topics aside, do we want to optimise such tests? I feel that is very low-priority.

> shyouhei (Shyouhei Urabe) wrote in #note-2:
>
> > Not against the ability to write obj.__send__(:method), but obj.method must be the preferable way and thus must be the fastest thing.
>
> I would think nobody prefers obj.__send__(:some_method) to obj.some_method if some_method is public, so it seems a non-issue to me.
> And anyway obj.some_method would always be as fast or faster than obj.__send__(:some_method), never slower (that would be a performance bug).

OK. So the point is wether we want people to call a private method or not. I'm still against that. Encapsulation is a very basic OO principle that Ruby employs. I want that be honoured.

The proposed patch is sending a wrong signal.

#7 - 10/28/2020 08:48 PM - marcandre (Marc-Andre Lafortune)

shyouhei (Shyouhei Urabe) wrote in #note-4:

> Private methods shall not be called at the first place. Period. That is breaking encapsulation.

I wish that was the case, but Ruby access is *not expressive enough* for this to be the case.

```
class Foo
  class << self
    private def special # General users should not call or rely on `special`
    end
  end

  def foo # even from inside our own class...
    self.class.special # this won't work without `send`
  end

  class Bar
    # Bar is a helper class, written by us
    def foo
      Foo.special # we want to call special, we need to use `send`
    end
  end
end
```

Note: using protected changes nothing in this example

In so many gems, you have to either mark method as private and use send, or else use # @api private but that's just a comment.

#8 - 10/28/2020 08:59 PM - shevegen (Robert A. Heiler)

Just a few things:

We need to also remember **.instance_variable_get()** and **.instance_variable_set()**.

Ruby does not quite use a similar "restriction-style" OOP like, say, java. It will
depend a lot on the style and preferences of the ruby user.

Personally I much prefer the more traditional .send() approach that ruby has had, so
zverok's comment about what to do with private stuff, I say .send() it all the way!
Gimme all the goodies; don't restrict me. :D (I tend to use "private" rarely, and
mostly as cue for documentation, rather than as a "I want to restrict this usage 100%").

On the topic of .send(), .public_send() and .**send**(), I have a few gems that use
.send(). I like .send(). I do not use .public_send() or .**send**(), so even a new
operator for .**send**() would not affect me since I would not need it, most likely.

Last but not least, at first I thought .send() will be deprecated, e. g. I misread
this change here by nobu:

https://git.ruby-lang.org/ruby.git/commit/?id=3198e7abd70bd2af977f2bb6c967e9df8f91adb0

Perhaps this one is also related to the issue here? I think some ruby users may
wonder what to use, e. g. .send() or .public_send() or .**send**(). This should also
be kept in mind. Note that I have no strong opinion on the issue here at all, my
only concern would be whether .send() would be changed, but I think I misread that
when I first saw the git-change.

#9 - 10/29/2020 12:11 AM - shyouhei (Shyouhei Urabe)

@marcandre (Marc-Andre Lafortune) Here you are:

```
class Foo
  using Module.new {
    refine Foo.singleton_class do
      def special
      end
    end
  }

  def foo
    self.class.special
  end

  class Bar
    def foo
      Foo.special
```

```
    end
  end
end
```

#10 - 10/29/2020 03:06 AM - mrkn (Kenta Murata)

- Is duplicate of Feature #17291: Optimize __send__ call added

#11 - 10/29/2020 04:06 AM - marcandre (Marc-Andre Lafortune)

shyouhei (Shyouhei Urabe) wrote in #note-9:

> @marcandre (Marc-Andre Lafortune) Here you are:
>
> [snip brilliant code]

1. I was wrong, I retract what I said.
2. My mind is blown. I have always thought of refinements as a way to safely monkey-patch other people's classes, in particular builtin classes. Never as a way to structure access to our own classes. This can also be very fine-grained if desired. This is brilliant @shyouhei (Shyouhei Urabe)!
3. Goto 1

I made a quick POC with RuboCop to isolate methods that I've always wanted to isolate and the only issues was mocking internal methods:

```
Failure/Error: allow(cop).to receive(:complete_investigation).and_return(cop_report)
#<Fake::FakeCop:0x00007fed7e1ce530 ...> does not implement: complete_investigation
```

I'm fine with this, I already try to avoid stub and mocks anyways (10 tests out of 14515 :-) and I'm sure I can find better ways around that.

I also wanted to check any performance impact. I couldn't see any (running tests or running RuboCop). Are there any known circumstances where performance would be affected?

Are there gems using this technique?
Blog posts discussing this?

#12 - 10/29/2020 08:12 PM - Eregon (Benoit Daloze)

How about a private module instead?

```
class Foo
  module Helpers
    def self.special # General users should not call or rely on `special`
      :special
    end
  end
  private_constant :Helpers

  def foo
    Helpers.special
  end

  class Bar
    # Bar is a helper class, written by us
    def foo
      Helpers.special
    end
  end
end

p Foo.new.foo # => :special
p Foo::Bar.new.foo # => :special

p Foo::Helpers.special # =>  private constant Foo::Helpers referenced (NameError)
```

Refinements seems rather heavy to me for this case, notably it creates extra modules, and makes initial lookups slower (once cached it shouldn't matter much).
For an uncached call (e.g. refine Object and many different receivers at some call site), I think the overhead would be noticeable.

Also if the refinements need to be used in multiple files, the module passed to using needs to be named, and stored in a private constant.
If done so, there seem little point to using PrivateHelpers; ...; self.class.foo vs PrivateHelpers.foo, except maybe for instance methods added on existing classes.

But then one could simply use a private method on Foo to begin with.

I'm probably biased against refinements because the semantics around refinements + super or eval are fairly messy.

#13 - 10/30/2020 12:17 AM - shyouhei (Shyouhei Urabe)

JFYI I learned the use of "immediate" refinement from rails.
https://github.com/rails/rails/pull/27363

#14 - 04/03/2024 03:50 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Open to Assigned