Is overriding a method in a subclass considered as a breaking change or not?

06/14/2020 05:52 AM - k0kubun (Takashi Kokubun)

Status: Closed
Priority: Normal
Assignee:

Description

Background

- In [Bug #15589](https://bugs.ruby-lang.org/issues/15589), I'm going to make Integer#zero? faster (comparable to == 0) on JIT by making sure the method is inlinable by MJIT.
  - Currently Integer#zero? is not defined and it's implemented as Numeric#zero?.
  - However, Numeric#zero?'s method definition handles not only Integer but also Rational and Complex. Because rb_equal which calls rb_funcall is used for Rational and Complex, we can't predict what'd happen there and it prevents Numeric#zero? from being inlined by MJIT. Moreover, reducing inlined code size is always good for JIT-ed code. So specializing Numeric#zero? to Integer#zero? will be beneficial for MJIT's inlining regardless of rb_funcall's presence.
  - So I'd like to define Integer#zero? for optimizing JIT-ed code of zero? call against Integer.

Discussion

Theoretically, overriding a method in a subclass in Ruby core might change an existing application's behavior (If a user overwrote Numeric#zero? and then Ruby core overwrote Integer#zero?, the user's definition would get suddenly ignored for Integer).

How should we decide whether the change is acceptable or not? Some possible ideas:

- Search all gems using gem-codesearch and check if there's any code to be broken.
  - In Integer#zero?''s case I found this code but it's not gonna be broken by overriding Integer#zero?. Other zero? method definitions seemed for other classes, but I can double check every occurrence if this will be the measure.
- Discuss and make a decision on case-by-case basis.
  - In Integer#zero?''s case we'll discuss whether overriding Integer#zero? is acceptable or not.

Associated revisions

Revision 95b0fed3 - 06/20/2020 09:55 PM - k0kubun (Takashi Kokubun)
Make Integer#zero? a separated method and builtin (#3226)

A prerequisite to fix https://bugs.ruby-lang.org/issues/15589 with JIT.
This commit alone doesn't make a significant difference yet, but I thought this commit should be committed independently.

This method override was discussed in [Misc #16961].

History

#1 - 06/14/2020 05:58 AM - k0kubun (Takashi Kokubun)
- Description updated

#2 - 06/14/2020 10:32 AM - Eregon (Benoit Daloze)
Could MJIT add its own check if called on a Fixnum, then do the inline fast-path logic, otherwise fallback to a call to the original definition? One would still need to check that Numeric#zero? is not redefined, but in all cases it's needed to check if redefined (whether it's Numeric#zero? or Integer#zero?).
That would work transparently without adding a new method.

Somewhat similar logic in TruffleRuby for +, nil?, etc, handling a subset of inputs, and rewriting as a call if inputs don't match that: https://github.com/oracle/truffleruby/tree/17701a09fca2fa1641353582d8c7be8216345b8a/src/main/java/org/truffleruby/core/inlined

I think we should avoid many new methods on e.g. Integer/Array duplicating Numeric/Enumerable, but rather have generic tricks to be able to specialize on common cases.

#3 - 06/14/2020 09:32 PM - marcandre (Marc-Andre Lafortune)
I imagine that it is fine as long as it's not done in a patch release and it appears in the NEWS.
Array has had some methods specialized for optimization purposes (minmax in 2.7, min/max/sum in 2.4, ...)

I doubt there are many valid reasons to redefine Numeric#zero? such that Integer#zero? wouldn't behave as expected.

#4 - 06/14/2020 09:39 PM - k0kubun (Takashi Kokubun)

Could MJIT add its own check if called on a Fixnum

You're right, that thing is technically not difficult. What I'm trying to do is to avoid adding MJIT-only method definition (like fast-path logic used only for MJIT) or (#ifdef) branches so that we do not cause bugs by discrepancy between VM and JIT and we do not need to maintain lots of copy-pasted code for MJIT.

I think we should avoid many new methods on e.g. Integer/Array duplicating Numeric/Enumerable

My assumption is that we already have many methods on Integer rather than Numeric, and this discussion is for minor leftovers which are inconsistent with other major methods. When such Numeric methods have branches for Integer, Complex, and Rational, removing a branch for Integer from Numeric and adding it as an Integer method don't cause a lot of code duplication at least.

rather have generic tricks to be able to specialize on common cases.

That'd be nice, but because I've already spent a lot of time for considering options I can easily think of, please describe your idea with concrete interface which is used to maintain it.

What kind of code in numeric.c or numeric.rb do you imagine to write for having Integer fast path for num_zero_p? Doesn't it create a path or code duplication which is never used (or tested) on VM? If not, doesn't it complicate the original method definition? (Well, I'm not saying we should never do such things, but those are what I tried to maintain when choosing the current proposal.)

Another idea brought by ko1 in case overriding it is considered a breaking change is that we undef Integer#zero? when Numeric#zero? is redefined. That's another vector for complication, but at least this technique could be also useful for VM and it's not too MJIT-specific.

#5 - 06/18/2020 08:49 AM - matz (Yukihiro Matsumoto)

We already have those overriding methods (e.g. Enumerable and Array) for performance. So I think it's OK if there's a clear benefit.

Matz.

#6 - 06/18/2020 06:07 PM - k0kubun (Takashi Kokubun)

- Status changed from Open to Closed