Ruby master - Feature #17171

Why is the visibility of constants not affected by `private`?

09/15/2020 08:34 PM - marcandre (Marc-Andre Lafortune)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

```
class Foo
  def call_me
    # ...
  end

  private

  SOME_DATA = %i[...].freeze  # is public, why not private?

  def calc_stuff  # is private, ok.
    # ...
  end
end
```

It's probably a naive question, but why shouldn't SOME_DATA's visibility be private?

When writing gems, more often than not the constants that I write are not meant for public consumption. I find it redundant (and tiresome) to explicitly write private_constant :SOME_DATA.

Related issues:

Related to Ruby master - Feature #16752: :private param for const_set	**Open**

History

#1 - 09/15/2020 08:50 PM - marcandre (Marc-Andre Lafortune)

- Related to Feature #16752: :private param for const_set added

#2 - 09/15/2020 09:04 PM - marcandre (Marc-Andre Lafortune)

I'd like us to consider changing this so that private also changes the visibility of future constants being set.

If that is deemed too risky for compatibility, could private_constant with no receiver do this?

#3 - 09/15/2020 11:18 PM - mame (Yusuke Endoh)

I think your expectation is reasonable. If we change it, we need much work to estimate its compatibility impact carefully and to design a migration path, of course.

I proposed and implemented private_constant ten years ago. [Feature #2366] was written in Japanese (sorry!), but as far as I see, no one discussed changing private, maybe because of compatibility. I did not introduce private_constant with no receiver because I wanted to avoid a new module state (as I recall, matz now dislikes the module state), but I agree that it is redundant.

#4 - 09/16/2020 09:13 PM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote in #note-3:

> I did not introduce private_constant with no receiver because I wanted to avoid a new module state (as I recall, matz now dislikes the module state), but I agree that it is redundant.

Interesting, is private implemented as state on the module? (but then would need to be thread-local when concurrently loading or so)
I always thought it's part of the state stored in the frame.

Regarding the issue, I agree it would be nice.
Maybe we can make an experiment and gather feedback from gems testing against ruby-head?

#5 - 09/17/2020 07:36 AM - byroot (Jean Boussier)

I agree too that it would be much cleaner, I keep seeing newer Ruby developers expecting it to work that way. That and def self.something.

> Maybe we can make an experiment and gather feedback from gems testing against ruby-head?

I can already tell you without a doubt it will break lots of code. What scares me is that since it would need to go through a deprecation cycle, and would be quite noisy, it has the potential to create a backlash like the keyword args change did. But maybe it wouldn't be as verbose?

#6 - 09/29/2020 05:08 PM - shevegen (Robert A. Heiler)

I am not quite as active on the bug-tracker as I used to be in the past, but I will comment on this particular issue since the reasoning given makes sense to me.

I will not comment on the backwards-incompatibility issue much at all, but let me state that I feel that the backwards-incompatibility is a separate consideration to make. For example, matz could approve of the idea here (if he agrees on it), but decide to delay changes here to some later stage for some ruby 3.x release in the future, or perhaps 4.0. There are quite a few changes that may come past ruby 3.0 anyway, but I feel that for this particular discussion here, it may be better to separate between the issue itself ("make constants private too"), and any incompatibility issue as a secondary consideration IF the proposed change would receive the "go-ahead" signal. Otherwise we may have problems discussing and changing anything, because the fear of incompatibility prevents any change, which would be bad too, in particular past ruby 3.0.

I also think that a deprecation period is perfectly fine, so I disagree with byroot; I also don't think it has the same impact as keyword arguments or frozen strings. Do note that I do not disagree with his comment that this may lead to broken code - yep, this is most likely true. I just think that a deprecation time is perfectly viable IF matz decides to approve this proposal. For example, when frozen strings were introduced, just about none of my ruby code would have worked. Nowadays, all my ruby code works 100% with frozen strings (or, say, 99.5%; I have not ported ALL my old ruby code, but I can easily do so step by step, so it really is a non-issue to me if it were the default. I also understand that larger code bases are harder to change of course). And here, in this particular case of frozen strings, a deprecation time did help. So I disagree with byroot on that particular note - I think a deprecation period would be perfectly fine. Just give people some time to adjust their code, it will slowly happen. **Make sure to communicate this too** , because not everyone reads the bugtracker. :) (This is sometimes a problem by the way; perhaps it is easier for japanese developers, but I think the communication part could be better on the english side. I think quite a few ruby users read updates to ruby mostly via blogs, and may never look at the bugtracker. I still read a lot of content on the bugtracker, but I am not commenting as much as I once did.)

So, next, I will **focus only on the suggestion itself** , without consideration for backwards incompatibility.

Let me "preface" this by saying that I don't often use "private". I also don't use .public_send(). To me .send() is more logical, and "private" is also not quite as logical - but this may depend on the OOP style that one uses. For example, Java users may perhaps prefer a stricter distinction whereas smalltalk users may prefer more unrestricted objects; I can understand that.

At any rate, while I am not really the target audience for the proposed change, and while I am not a big fan of "private" in general (by the way, nobody mentions **protected** :D), what marcandre writes still **makes a LOT of sense** to me.

Why is that so?

Well, simple.

Consider a .rb file where the ruby developer writes "private" in the middle of that .rb file, just as shown by marcandre in the example above (we can ignore the CONSTANT for the moment). Ok, we can expect that all methods past that point are now "private" too, thus more restricted than by default. That is the natural way to use private, yes?

IF if you come from THAT point of view, where the simple use of "private" switches the other methods that follow towards the state of "more restrictions" than the default "public" does, then actually what marcandre writes, makes a LOT of sense to me as well.

- If the methods are private, why not the constants too? Makes sense what is written there really. The intent seems to be that the ruby user wanted the following code to be more restricted after all.

You can of course say that constants and methods are not directly linked, yes; perhaps "private" implies more to mean "private_method" only. I am not so sure, but let's just assume this for the moment, that a ruby user could specifically want ONLY the methods to be private, and NOT the constants at the same time.

We could have longer APIs, such as "private_method" and "private_constant" perhaps (not that I propose this, it is just a thought experiment) - I feel this is a bit separate to the issue at hand, though. Aka, what "private" really means in this context.

So, the key question then would be:

- What does "private" entail, and, following on this
- How much control should ruby users have over ruby code in this regard?

The second part in particular feels 100% like a language design choice, so for matz to decide.

To me, then, "private" really means more restrictions past that point.

Even then, though, I think it makes a lot of sense to me here what marcandre wrote. I think this will apply by far to most ruby code as well out there, if they use "private" in this way - at the least as an intention. The intention seems to clearly be to put all methods into a "more restricted" state. (Yes, some older ruby code may have problems, as byroot pointed out, but that is a separate consideration to make, in my opinion.)

IF the ruby user uses "private" in the way shown by marcandre, to me it also means that the user is saying something like this:

```
"Ruby, please, I would like to have all the code that follows there to be more restricted, to avoid funky outs
ide use of this code".
```

And to me, if this instruction given to ruby is true, then it would also make a lot of sense to apply this restriction to constants just as well, if you come from that point of view or a similar point of view.

I can not say whether that was the same point of view marcandre used, but I come to the same conclusion in the end, so +1 from my side on that proposed change. To me it seems "logical" (even if it may not be logical per se, but I think marcandre is right; most people may probably assume that a restriction should occur, IF private is used in that manner. And that is coming from someone who rarely uses private; I just feel it really makes sense, since it will capture the intention of most ruby users IMO.)

So, TL;DR, +1 from me.

#7 - 10/16/2020 03:55 AM - marcandre (Marc-Andre Lafortune)

So can we envision issuing a warning for two release cycles?

Is it feasible to issue a warning for constants that are currently public that would be made private this way?

Difficult being that call to private_constant happens after point of definition:

```
class Foo
  private
  X = 42
end # => Warn about privacy change

class Bar
  private
  X = 42
  private_constant :X
end # => No warning
```

#8 - 10/16/2020 04:19 AM - jeremyevans0 (Jeremy Evans)

marcandre (Marc-Andre Lafortune) wrote in #note-7:

> So can we envision issuing a warning for two release cycles?
>
> Is it feasible to issue a warning for constants that are currently public that would be made private this way?
>
> Difficult being that call to private_constant happens after point of definition:
>
> ```
> class Foo
>   private
>   X = 42
> end # => Warn about privacy change
>
> class Bar
>   private
>   X = 42
>   private_constant :X
> end # => No warning
> ```

I think if we do this, we should only warn on access (i.e. Foo::X), not definition. This would require internal changes to add a sort of deprecated-public visibility.

#9 - 10/16/2020 05:26 AM - marcandre (Marc-Andre Lafortune)

Deprecating on access (first time only) would be less noisy and sounds easier to implement indeed □□

#10 - 10/16/2020 02:49 PM - znz (Kazuhiro NISHIYAMA)

If private affects constants, are constants private after protected?

```
class Foo
  protected
  P1 = 1 # private constant or not?
  private
  protected
```

```
  P2 = 2 # private constant or not?
end
```

#11 - 10/16/2020 03:28 PM - Dan0042 (Daniel DeLorme)

At the same time it would be worth reconsidering #16276, which would allow private{ X = 42 } right now without a deprecation warning or having to use private_constant

#12 - 10/26/2020 05:44 AM - nobu (Nobuyoshi Nakada)

- Description updated

#13 - 10/26/2020 07:52 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

This looks very interesting, but it would introduce quite big incompatibility. I don't want to break existing code.

Matz.