Freezing strings can give us a nice performance boost, but freezing previously non-frozen strings is a backward-incompatible change which is hard to handle because the place where the string is mutated can be far from where it was frozen, and tests might not cover the cases of frozen input vs non-frozen input.

I propose adding a flag which gives us a migration path for freezing strings. For purposes of discussion I will call this flag "eventually_frozen". It would act as a pseudo-frozen flag where mutating the object would result in a warning instead of an error. It would also change the return value of Object#frozen? so code like obj = obj.dup if obj.frozen? would work as expected to remove the warning. Note that eventually_frozen strings cannot be deduplicated, as they are in reality mutable.

This way it would be possible for Symbol#to_s (and many others) to return an eventually_frozen string in 2.7 which gives apps and gems time to migrate, before finally becoming a frozen deduplicated string in 3.0. This might even open up a migration path for eventually using frozen_string_literal:true as default. For example if it was possible to add frozen_string_literal:eventual to all files in a project (or as a global switch), we could run that in production to discover where to fix things, and then change it to frozen_string_literal:true for a bug-free performance boost.

Proposed changes

- Object#freeze(immediately:true)
  - if immediately keyword is true, set frozen=true and eventually_frozen=false
  - if immediately keyword is false, set eventually_frozen=true UNLESS frozen flag is already true
- String#+@
  - if eventually_frozen is true, create a duplicate string with eventually_frozen=false
- Object#frozen?(immediately:false)
  - return true if immediately keyword is false and eventually_frozen flag is true
- rb_check_frozen
  - output warning if eventually_frozen flag is true

Alternatively: the eventually_frozen flag is an internal detail only

- OBJ_EVENTUAL_FREEZE
  - used instead of OBJ_FREEZE in rb_sym_to_s and others to set eventually_frozen=true
- Object#freeze
  - set frozen=true and eventually_frozen=false
- String#+@
  - if eventually_frozen is true, create a duplicate string with eventually_frozen=false
- Object#frozen?
  - return true (or maybe :eventually) if eventually_frozen flag is true
- rb_check_frozen
  - output warning if eventually_frozen flag is true

Related issues:
Related to Ruby master - Feature #16150: Add a way to request a frozen string... Open

History

#1 - 09/18/2019 01:27 PM - Dan0042 (Daniel DeLorme)
- Description updated

#2 - 09/19/2019 09:11 AM - duerst (Martin Dürst)
freeze(eventual: false) gives the impression that it will never be frozen, but the proposal is to have this mean that it is frozen immediately.

So it would be better to say freeze(immediately: true) for what happens now, and frozen(immediately: false) for the new behavior.
The naming is a bit wonky, I'll freely admit. I'll change the description to your suggestion.

Does this mean you agree with the basic idea?

Dan0042 (Daniel DeLorme) wrote:

The naming is a bit wonky, I'll freely admit. I'll change the description to your suggestion.

Does this mean you agree with the basic idea?

The details and pros and cons were discussed at yesterday's Ruby committer meeting, with Matz present. I mostly commented on the naming, and I'm not sure I remember all the other details of the discussion correctly, sorry.

Shevegen (Robert A. Heiler)

Hmm. I can somewhat understand the proposal, but I believe you would also add a bit more complexity here; you need to remember that ruby users may have to understand the idea behind this too, and that adds a bit of complexity. A bit like the Maybe monad in Haskell ... :P

Personally I think it would be much simpler to not introduce this, and to instead have matz decide WHEN frozen strings will become the default (you may refer to all frozen objects but I think strings are the big ones, due to them being used so much in ruby programs out there).

The main question or issue that matz mentioned was in regards to backwards compatibility, e.g. for people to transition from ruby 2.0 to ruby 3.0 without much problem, which is understandable in my opinion (see the migration from ruby 1.8.x to ruby 2.0 which was not trivial for everyone).

On the other hand, I actually think it would be better to, rather than add more complexity here, actually do the switch even if it may be painful for people, just as headius proposed to then have frozen strings by default for ruby 3.0. You can probably handle some of the problems e.g. make some exception for old gems so that they work as-is or something. I would prefer that much more compared to a maybe-frozen situation; although I am also perfectly fine with frozen strings not being the default in 3.0 either. At this point all my current and actively maintained code has a frozen-string comment so the change would not affect my own code anyway; may be harder for cod that is not maintained though.

I think the naming is not necessarily the biggest issue here; even with a better name it may add complexity. Remember that in ruby 1.8 people did not have to think about it much at all since there were no frozen strings; or at the least I can barely remember much code with "abc".freeze all over. That somehow was found in later ruby code, where some people literally applied .freeze to just about everything they could get their hands on. I always found this sort of strange, too.

Matz didn't determine this proposal at the meeting. There were four points discussed:

- Before matz accepts this proposal, he must decide the grand design change: Ruby should be immutable by default?
- If we want to make Ruby gradually immutable, this proposal is feasible.
- However, it is arguable if it is worthwhile consuming one bit for each object for this feature.
- Even after Ruby is immutable, it is doubtful if Ruby becomes so faster.

I'm against making Ruby immutable by default. One of the most important properties of Ruby is dynamics. Ruby has allowed users to change almost anything in run time: dynamically (re)defining classes and methods (including core builtin ones), manipulating instance variables and local variables (via Binding) through encapsulation, etc. These features are not recommended to abuse, but they actually bring big flexibility: monkey-patching,
DRY, flexible operation, etc. However, blindly freezing objects may spoil this usefulness.

It is a bit arguable if this flexibility limits performance. Some people say that it is possible to make Ruby fast with the flexibility kept (TruffleRuby proves it). That being said, I admit that the flexibility actually limits performance in the current MRI, and that we have no development resource to improve MRI so much in near future. I think that my proposal #11934 was one possible way to balance the flexibility and performance. Anyway, we need to estimate how much Ruby can be faster if the flexibility is dropped. If it becomes 10 times faster, it is worth considering of course. If it becomes 10 percent faster, it does not pay (in my opinion).

#7 - 09/23/2019 08:43 PM - Dan0042 (Daniel DeLorme)
I think it's important to make a distinction between "immutable" and "frozen".

Some programming languages have immutable data structures, and some programmers find that concept really cool, functional, powerful and whatnot. This is a "grand design" kind of thing. Like mame, I am very much against ruby going in that direction. It would only result in ruby becoming a half-assed functional language and no one would be happy.

On the other hand, freezing certain strings is merely an optimization concern. Not even all strings, just Symbol#to_s, Module#name and literals. Heck, just static string literals if I had anything to say about it. Certainly in many cases it's a premature optimization, but I still feel weirdly "uncomfortable" with all these extra strings objects being allocated when 99.9% of the time they're not needed.

In #16150 matz says "For frozen Symbol#to_s, I see a clear benefit. But I worry a little bit for incompatibility."
In #11473 he says "I REALLY like the idea but I am sure introducing this could cause HUGE compatibility issue"

This is not about some grand design to make ruby immutable, this is just about facilitating some specific desirable cases that are otherwise too hard because of incompatibility.

My proposal was to have the eventually_frozen flag visible and modifiable via regular ruby methods (because why not make it available to everyone?)
But if there's a concern about overuse spoiling the flexibility and usefulness of ruby, I think it would also be fine to limit this to the C API and have the core team decide where is best to use it.

#8 - 11/01/2019 01:56 PM - Dan0042 (Daniel DeLorme)
- Description updated

Since returning a frozen string from Symbol#to_s was reverted, it looks like this proposal is still relevant.

Is it realistic to wish for this in 2.7? The sooner the better imho.

#9 - 11/03/2019 04:52 PM - Eregon (Benoit Daloze)
Dan0042 (Daniel DeLorme) wrote:

- Since returning a frozen string from Symbol#to_s was reverted, it looks like this proposal is still relevant.

It's not reverted (yet at least).

#10 - 11/03/2019 04:58 PM - Eregon (Benoit Daloze)
For this discussion, I would focus on having this for Strings only initially.
I don't think it's about the "grand design of immutability", but rather a tool to ease migration towards frozen Strings for some APIs.

My main concern about it would be that such strings are reported as Kernel#frozen? but actually are not. That is some inconsistency.
I think it would be more logical to return false for Kernel#frozen? and just have String#+@/String#dup make a copy without the eventually_frozen flag.

#11 - 11/04/2019 04:17 PM - Dan0042 (Daniel DeLorme)
- Description updated

Eregon (Benoit Daloze) wrote:

- For this discussion, I would focus on having this for Strings only initially.

Fair enough. I don't see much point in having this for objects other than strings. Focusing on strings might simplify the implementation.

- My main concern about it would be that such strings are reported as Kernel#frozen? but actually are not. That is some inconsistency.
I think it would be more logical to return false for Kernel#frozen? and just have String#+@/String#dup make a copy without the eventually_frozen flag.

Let's keep in mind this is just a temporary bridge to reach frozen strings. As such I wouldn't worry too much about inconsistency. Sure, consistency is important, but it's far more important that the bridge does a good job of facilitating migration. And for that I think it's necessary to handle the situation of str = str.dup if str.frozen? which is a pretty common pattern in ruby code. Right now I can't think of an example where returning false would be
beneficial.

#12 - 11/06/2019 10:49 AM - Eregon (Benoit Daloze)

Dan0042 (Daniel DeLorme) wrote:

And for that I think it's necessary to handle the situation of str = str.dup if str.frozen? which is a pretty common pattern in ruby code.

Right, it is useful for that pattern at least.
I think String#+@ would be a better way to write that, but it's only available since Ruby 2.3.

I would also recommend an unconditional .dup to avoid mutating arguments, the fact a String is mutable doesn't mean there is no other reference to it which expects the String to not change.
That depends on the specific code around of course.

I think we all agree this is a good way to "deprecated" methods returning mutable strings and let them return frozen strings in the future, isn't it?
This could be useful for instance for making Symbol#to_s return a frozen String after deprecation (#16150)

#13 - 11/06/2019 11:09 AM - byroot (Jean Boussier)

I think we all agree this is a good way to "deprecated" methods returning mutable strings and let them return frozen strings in the future, isn't it?
This could be useful for instance for making Symbol#to_s return a frozen String after deprecation (#16150)

Agreed, not sure if we can still hope for the Symbol#to_s to make it to 2.7, but regardless having such way for marking strings (or other objects ?) as frozen in the next version would be convenient.

#14 - 02/10/2020 07:39 PM - Eregon (Benoit Daloze)

Here is a POC PR by byroot (Jean Boussier) to implement this: https://github.com/ruby/ruby/pull/2872

#15 - 02/10/2020 07:40 PM - Eregon (Benoit Daloze)

- Related to Feature #16150: Add a way to request a frozen string from to_s added