Ruby master - Misc #16047

Reconsider impact of frozen_string_literal on dynamic strings

08/05/2019 05:03 PM - Dan0042 (Daniel DeLorme)

Status:	Open
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)

Description

The rationale for introducing frozen_string_literal was because rubyists were starting to litter their code with "".freeze for optimization, and it's ugly.

But by using frozen_string_literal we introduce the opposite problem: we must litter the code with "".dup in order to have mutable strings, and it's ugly.

The rationale for freezing all strings including dynamic was because it's easy to explain
This may be true, but at the expense of making it cumbersome to use. And freezing dynamic strings is useless (no-op) for memory optimization, but making it mutable again via "foo #{bar}".dup means we allocate two strings where only one was needed before.

In my personal experience using frozen_string_literal, I find that static strings are usually ok to freeze without changing anything else, but that freezing dynamic strings often create bugs that require +"" or "".dup boilerplate to circumvent. So in the end I found myself stopping regular use of that feature, since it's more trouble than it's worth.

As such I'd like to ask other rubyists how has been their experience with **actually using** frozen_string_literal on a day-to-day basis; if my experience is unique or common. Thank you for sharing your thoughts.

History

#1 - 08/05/2019 06:16 PM - shevegen (Robert A. Heiler)

I'll comment on this, but allow me to make a few partially related/unrelated comments first.

Benoit has in another discussion made a comment about having asked xyz folks about a feature/change, in regards introducing (and accessing) numbered parameters in additio to denoting the name of the variable at hand. Here in this issue, I do not want to go into the pros/cons of that other feature, but purely refer to the "popularity" part.

I understand that people will be very biased in favour of what they like, and what they dislike; I am not different to this. :) (Although, perhaps it may be culture-related; japanese folks seem to not have the same "pattern" or perhaps I may not see it if they write in japanese, here on the bug tracker, I don't know - I can only see the pattern on the ruby tracker and mostly non-japanese like to complain more than japanese. At the least on the bug tracker ... :P)

Anyway - what I want to say with this is that matz is the one making the decisions in regards to ruby. So ultimately this is more a discussion that has to go talk about pros/cons, in regards to features/changes of ruby, and perhaps convince matz in one way or another, rather than any "convince others"; or ask others about their opinion per se. Which is of course perfectly fine too - just that matz is the language designer of ruby, not random people all over the world.

However had, you have also asked specifically about opinions, and I will provide my opinion. I noted this down before over the years, in regards to frozen strings.

Personally, I find:

```
"abc def".freeze
```

AND

```
"abc def".dup
```

to be ugly in general.

I think it is better to never have to use .freeze or .dup explicitely. And still have everything be as fast as possible ... :P

We do, however had, have to admit that lots of people were using .freeze in the past, in particular with style such as this:

```
FOOBAR = 'abc def ghi'.freeze
```

I saw that in many different projects; often combined with freezing Arrays as well.

I find that style very ugly, but I believe a reason why this was done is because people are so eager to want to squeeze out the maximum of ruby in regards to its performance. And this is understandable to some extent.

Matz tends to say that nobody minds if ruby (or any programming language) becomes faster. :)

That is also partially a goal of the ruby 3.0 3x3 approach, in regards to "make ruby 3.0 3 times as fast as ruby 2.0", at the least in regards to benchmarks listed by the optcarrot test (I think). No idea if the goal has already been reached or not, but the last time I checked it was close even without mjit; and with mjit we already know we are above the goal (if I remember correctly; but I don't want to talk about the mjit now, that is only partially related to the topic).

I should like to correct you on one matter, though.

You wrote:

> The rationale for introducing frozen_string_literal was because
> rubyists were starting to litter their code with """.freeze
> for optimization, and it's ugly.

Now - if I remember correctly then this was not the rationale.

I think matz actually had a presentation back when frozen string literals were added. The primary reason, or at the least from what I remember, was that it leads to more efficient ruby code, e. g. in particular in regards to the typical string allocation situation - and it leads to faster execution as a consequence.

Again - my memory is not great, so I may misremember, but for the most part I believe that was the main rationale; I do not remember that the rationale ever was because people would use lots of .freeze in their code base. For example, I never did use .freeze in my own code bases, mostly because I hate how ugly it looks. It distracts me from other things too. :P

Immutable strings are faster simply because less can be done with them.

I also always said that having to use .dup and checking for as to whether a string is frozen or not, leads to more code, and also uglier code as a consequence. The various .dup additions are not very pretty to see in code.

There are other ways to de-dup, such as String#- and what not, but I also don't think they are very pretty. They are mostly to be able to write less code, yet still work with immutable strings.

Example:

https://ruby-doc.org/core-2.6.3/String.html#method-i-2D-40

I actually use the more verbose variant here, e. g.

```
if string.frozen?
  string = string.dup
```

Not very pretty, I know, but I actually like this more than -str and +str, oddly enough.

Anyway.

To give you my personal opinion here: both .dup and .freeze are ugly. ;)

There is, however had, one undeniable adantage - it leads to ruby scripts being executed faster, in particular for large projects.
I have seen this in my own projects; although I can not give any specific benchmarks or speed improvements here, I am using ruby just about daily from the commandline, and the speed improvement IS noticable to me.

So I am actually using frozen strings, even though I like oldschool string behaviour more. :P

I used to add:

```
frozen_string_literal: false
```

to all my .rb files, but at some point I started to change it to frozen_string_literal: true.

If the magic frozen comment is kept then I think it is no problem - people can just use whatever they prefer.

If I remember correctly then the idea was to default to frozen strings in general for 3.0, but some time ago matz dropped this and postponed it to after ruby 3.0, due to the compatibility problem (feel free to correct me if my description of this is not accurate).

> But by using frozen_string_literal we introduce the opposite
> problem: we must litter the code with """.dup in order to
> have mutable strings, and it's ugly.

You could use frozen_string_literal: false; and you could decide to NOT use .freeze either.

You are only focusing on some parts here, but not all of them. ;)

There are ALWAYS trade-offs, of course. Oldschool ruby will be slower. People have to decide what they want to have; but I also don't think that inertia should affect OTHER ruby users, if they want to see changes or improvements, such as ruby scripts being interpreted in a faster way.

Part of the more-than-one-way philosophy is to pick the strategy that best works for you and your use cases.

By the way - you haven't yet suggested what you'd like to change specifically. :)

I am also sure it would lead to some other trade-offs again.

> The rationale for freezing all strings including dynamic was
> because it's easy to explain

I don't know from where you get these explanations of the rationales. Has matz actually said this? I am quite sure that it was not in any of his presentations, since I tend to watch these presentations. Often simpler to keep track of what is changing in ruby than reading through lots of bug tracker issues. :D

I think one big reason for the magic comment was simply for compatibility, because with a hard change to defaulting to frozen strings, it would mean that lots of ruby code out there in the world would not work for new ruby releases, without them being changed. And changing all these .rb files will take ruby users quite some time. You can look at the transition phase from ruby 1.8 to ruby 2.0 and beyond. Some people are fast, others are slow.

I used to be very slow in the past, and used multiple ruby versions too, but since some time I only use the latest stable (xmas release) and just keep on changing local code in .rb files whenever something changes. (I default to latest stable mostly because my heavy tinker days are over - I can always wait for the next xmas. I do, however had, also sometimes try new things in ruby-dev releases, but most of the time I just default to the latest stable.)

So to come back to the transition period - having a long transition period can be good. You give people time to adjust and adapt at their own pace. IMO, this is a good thing.

I still have not changed to frozen string literals to "true" for all my .rb files; only in my more important projects have I done so.

For new .rb files I use ruby to autogenerate these .rb files and these files then have the proper encoding line/format, and a frozen-string true comment on top, too.

> This may be true, but at the expense of making it cumbersome
> to use.

Yes, it is more cumbersome to use. I don't think anyone disputes this.

> And also completely useless for memory optimization.

I don't understand you - if we have to allocate less RAM etc... for String objects, the programs will be interpreated/evaluated more quickly, yes? To me this is a speed-related optimization.

Note that I am not necessarily advocating the change as such in totality; in the past I have also commented that I don't think speed-related considerations should be the primary impetus for change regarding to ruby. I never used ruby due to speed; but there are also people who complain about ruby being "slow", so I can understand the objective to make ruby more efficient, faster etc...

It's simply a trade-off situation.

In the context of frozen strings, although I agree that it is uglier than in oldschool ruby, I think that it is actually worth it, due to the speed gain, which is noticable (really; someone should add some benchmarks perhaps so that people can objectively compare it, but I am sure that there is a big difference).

Since the old behaviour is still there, and nobody is forced to HAVE to use .dup and .freeze in their own code base, I don't fully understand the claim that people are required to have to use either.

> In my personal experience using frozen_string_literal, I find
> that static strings are usually ok to freeze without changing
> anything else, but that freezing dynamic strings often
> create bugs that require +"" or "".dup boilerplate to
> circumvent. So in the end I found myself stopping regular
> use of that feature, since it's more trouble than it's worth.

Yes, working with frozen strings is more annoying.

Oldschool ruby were just like this:

```
object << 'hello ' << 'world'
```

So I do not disagree with you completely; oldschool ruby is easier to work with.

But we can still control that behaviour; and I am perfectly fine using frozen strings in my .rb files, in the comment section. I don't see any problem with this. If you want the oldschool behaviour, you can just use frozen-strings to false; which should be the default right now, too. So I don't see your point, or perhaps I am not understanding you completely either.

More boilerplate code is added, yes; this is not ideal either. I agree on that part.

It really is a trade-off. You want faster strings? Well, then they may have to be immutable. I think python does something similar or has had a similar reasoning for having Strings be immutable by default.

It's simply a trade-off.

Personally I use frozen strings a lot. I found that it is perfectly possible to work with them in general and I don't have any big problem with them (but they are uglier, yes).

> As such I'd like to ask other rubyists how has been their experience with actually using frozen_string_literal on a day-to-day basis; if my experience is unique or common. Thank you for sharing your thoughts.

I use them on a daily basis (if I write ruby code on that day). I actually almost never use .freeze, because I dislike it for some reason (hard to explain why).

I have to use .dup a lot in my code, yes. Not perfect, but not the end of the world either.

I guess the ideal situation would be if we could have frozen strings AND never have to use .dup or .frozen? or #- or #+.

No idea if that would be even possible, though.

But to conclude - I think it is ultimately a trade off that can be managed.

This is just my opinion though; matz is listening to feedback from folks who use (real world use cases that is, not purely abstract ones).

#2 - 08/05/2019 08:20 PM - Eregon (Benoit Daloze)

- Assignee set to matz (Yukihiro Matsumoto)

I was also surprised that dynamic strings are also frozen, since the gain there seems minimal.
"a#{b}c" is conceptually the same as "a".dup << b.to_s << "c" and so freezing that seems artificial and not needed since there was mutation in the first place.
Also, it's basically one more mutation to the resulting String by changing its frozen flag, that would otherwise not be needed.

Interestingly enough, dynamic strings are not frozen currently in TruffleRuby (not intentional), and that seems to have caused no compatibility issue so far.
So if we went with this, I think it would be very compatible.

We use # frozen_string_literal: true for all core library Ruby files in TruffleRuby (18k SLOC, not typical application code though).
The majority of dynamic strings seem to be used for exception messages and #inspect.
Exception messages are mutable in MRI but we decided +"" or "".dup are too ugly and so we keep them frozen when they are static string literals, which seems fine for compatibility so far.
OTOH, results of #inspect and other core methods are more commonly mutated by the user, and so there we would have to use .dup or +" #{...} " in quite a few places.
I agree with you, I intuitively expect that dynamic strings are not frozen.

#3 - 08/05/2019 10:18 PM - ioquatix (Samuel Williams)

I started using frozen_string_literal and I also wish frozen_literal (generally applies to Array [], Hash {}, etc).

The model I use:

Mutable string: String.new

Constant string: "CONSTANT"

I've experienced bugs because strings (and other things) were mutated after being passed as an argument to an object constructor.

I feel immutable by default is the safest and most efficient option, and users should opt into mutability when required, using constructors, e.g. as shown above for string, or [] (immutable) vs Array.new (mutable). We can establish some standard pattern.

With good CoW implementation, calling .dup should almost be a no-op for most data structures, so we should encourage that in object constructors, e.g.

```
class Box
  def initialize(items)
    @items = items # risky
```

```
    @items = items.dup # safe but maybe expensive?
  end
end
```

This is also made more efficient by immutable by default.

Finally, we should provide better performance guarantees to users and give some general purpose rules like using #dup when you are receiving arguments and saving into instance variables. Right now, I see many code doing different things.

#4 - 08/06/2019 01:59 AM - Dan0042 (Daniel DeLorme)

shevegen (Robert A. Heiler), I didn't suggest what I would like to change because I was mostly looking for insight on people's real-world usage of frozen_string_literal and how it might compare to mine. That's why I put this under the Misc tracker and not Feature. But since you took the time to write such an extra-complete response I will try my best to answer your questions back.

> The rationale for introducing frozen_string_literal

Before frozen_string_literal, "".freeze was introduced as a memory optimization. Prior to ruby 2.1 it would just create a new string and freeze it, but from 2.1 on it would deduplicate it also. So everybody started using this for memory performance. This was seen as injecting code ugliness in exchange for speed and not the ruby way. At least as explained here by matsuda (Akira Matsuda)

I am interested in the claimed/perceived speedup. I had seen some benchmarks that claimed no difference in speed. And that made sense to me since deduplication is a memory optimization, not a significant CPU optimization.

Of course I can also choose to use frozen_string_literal: false (and I do), but the point that I was trying to make is that it doesn't have to be an either-or situation; we could have the **best of both worlds** if dynamic string literals were not frozen by the magic comment.

> The rationale for freezing all strings including dynamic was because it's easy to explain

I got that rationale from here

> And also completely useless for memory optimization.

What I meant here is that "foo #{bar}" allocates a string so there's no deduplication and no advantage to freezing; and "foo #{bar}".dup allocates two strings where only one was previously needed. Of course everything else is a worthy optimization so saying "useless" was perhaps a bit too strong.

#5 - 08/06/2019 02:15 PM - byroot (Jean Boussier)

> I am interested in the claimed/perceived speedup. I had seen some benchmarks that claimed no difference in speed. And that made sense to me since deduplication is a memory optimization, not a significant CPU optimization.

The speedup mostly come from reduced GC pressure. Consider the following snippet:

```
def call(env)
  env["HTTP_PATH_INFO"]
end
```

Without frozen_string_literal: true it's pretty much equivalent to:

```
HTTP_PATH_INFO = "HTTP_PATH_INFO".freeze
def call(env)
  env[HTTP_PATH_INFO.dup]
end
```

That's a totally useless allocation. In itself it doesn't matter much, but if it's in a tight loop it can really participate in trashing the GC, and overall, it adds up.

Also, if you profile a Ruby application you'll very often see 20-30% of the time spent in the GC, so reducing useless allocations as well as the size of the heap is a fairly good medium for increasing CPU performance.

As for .dup/+, maybe that's me but I extremely rarely need it.

> we could have the best of both worlds if dynamic string literals were not frozen by the magic comment

I don't really see it as a upside. IMO these are literals before being "dynamic", and as such it seems much more consistent to me that they'd be frozen as well.

#6 - 08/06/2019 02:21 PM - Dan0042 (Daniel DeLorme)

- Description updated

#7 - 08/06/2019 02:41 PM - jeremyevans0 (Jeremy Evans)

byroot (Jean Boussier) wrote:

> The speedup mostly come from reduced GC pressure. Consider the following snippet:
>
> ```
> def call(env)
>   env["HTTP_PATH_INFO"]
> end
> ```
>
> Without frozen_string_literal: true it's pretty much equivalent to:
>
> ```
> HTTP_PATH_INFO = "HTTP_PATH_INFO".freeze
> def call(env)
>   env[HTTP_PATH_INFO.dup]
> end
> ```

Unfortunately, this is not a good example, because even without frozen_string_literal: true, Ruby already optimizes this case. Try the following:

```
GC.disable
ObjectSpace.count_objects[:T_STRING]
x = ObjectSpace.count_objects[:T_STRING]
h = {'a'=>1}
100000.times{h['a']}
p(ObjectSpace.count_objects[:T_STRING] - x)
```

> > we could have the best of both worlds if dynamic string literals were not frozen by the magic comment
>
> I don't really see it as a upside. IMO these are literals before being "dynamic", and as such it seems much more consistent to me that they'd be frozen as well.

I agree it makes more consistent to freeze both static and dynamic strings literals. Personally, I think only freezing static strings is more useful behavior, even if less consistent, but that ship has sailed and it would be worse to try to change the behavior now.

#8 - 08/06/2019 05:43 PM - byroot (Jean Boussier)

> Ruby already optimizes this case

TIL indeed. I should have used a random method call instead.

#9 - 04/14/2020 04:37 PM - ngan (Ngan Pham)

I find myself only applying ".freeze" to all-cap constants:

RED = "red".freeze

I think it would nice if Ruby 3 only (automatically) froze string that are declared/assigned to constants.