**Ruby master - Feature #17104**

**Do not freeze interpolated strings when using frozen-string-literal**

08/05/2020 08:06 PM - bughit (bug hit)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Closed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td>Eregon (Benoit Daloze)</td>
</tr>
<tr>
<td>Target version:</td>
<td>3.0</td>
</tr>
</tbody>
</table>

**Description**

I think the point of frozen string literals is to avoid needless allocations. Interpolated strings are allocated each time, so freezing them appears pointless.

```ruby
#frozen_string_literal: true

def foo(str)
  "#{str}
end

fr1 = 'a'
fr2 = 'a'
fr1_1 = foo(fr1)
fr2_1 = foo(fr2)

puts fr1.__id__, fr2.__id__, fr1_1.__id__, fr2_1.__id__
puts fr1_1 << 'b'
```

**Related issues:**

Is duplicate of Ruby master - Misc #16047: Reconsider impact of frozen_string...

Closed

**Associated revisions**

Revision 9b535f3f - 09/15/2020 07:32 PM - Eregon (Benoit Daloze)

Interpolated strings are no longer frozen with frozen-string-literal: true

- Remove freezestring instruction since this was the only usage for it.
- [Feature #17104]

Revision 7b2bea42 - 09/30/2020 01:15 PM - nobu (Nobuyoshi Nakada)

Unfreeze string-literal-only interpolated string-literal

[Feature #17104]

**History**

#1 - 08/05/2020 08:25 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed

Other people have felt the same way, including me (see https://bugs.ruby-lang.org/issues/11473#note-7 and https://bugs.ruby-lang.org/issues/8976#note-67). However, matz (Yukihiro Matsumoto) decided during the November 2015 developer meeting that they should be frozen for simplicity, see https://docs.google.com/document/d/1D0EoSN7NE_ unrlySOKG3IVi_evyXt68BQPM4Pkp7NvMyQ/pub

#2 - 08/05/2020 08:58 PM - bughit (bug hit)

If you want to get a mutable string from an interpolated literal +"_#{method1}"+, 2 allocation will be done instead of 1, if it weren't pointlessly frozen.

In this case a feature designed to reduce allocations is producing more allocations. Behavior that's counter-intuitive and illogical and acting counter to its intent, is not simple.

This happens to be something that can be changed without breaking anything. Can it get a second look?

#3 - 08/05/2020 09:05 PM - jeremyevans0 (Jeremy Evans)

If you would like the behavior changed, you should file a feature request for it, and add it to the agenda of a future developer meeting.
FWIW I'd be +1 to make interpolated Strings non-frozen. It's BTW still the behavior in TruffleRuby to this day, since it seems to cause no incompatibility and is convenient for writing the core library with Ruby code.

Can't we just treat this as I feature request? The reasons are, it will reduce allocations, be more logical, less surprising and produce simpler code (when a mutable string is needed and you don't want extra allocations)

We can treat this as a feature request. I changed it from Misc to Feature and modified the Subject to be the feature you are requesting as opposed to a question. The issue for the next developer meeting is at https://bugs.ruby-lang.org/issues/17041 if you want to add it to the agenda.

If you want to get a mutable string from an interpolated literal "+"#{}", 2 allocation will be done instead of 1

Because I understand the consistency argument. "All string literals are frozen" is much easier to wrap your head around than "All string literals are frozen except the ones that are interpolated".

I'd argue "a#(2)c" is not a string literal ("a2c" is a string literal). It's actually syntactic sugar for mutableEmptyStringOfSourceEncoding << "a" << 2.to_s << "c". (+ .freeze the result in current semantics which feels unneeded)
Same as 42 is an integer literal, but 41 + 1 is not an integer literal.

I understand your point of view. However in my view what defines a literal, is the use of a specific syntax, so "" in this case.

Same for hashes or arrays, [1 + 2] is a literal (to me), it might not be a "static" literal, but it is a literal nonetheless.

There is only one reason for freezing literals, to be able to intern them and reduce allocation. In fact the feature is poorly named, after the consequence(freezing), not the cause (interning).

Freezing strings that are not interned is pointless and counterproductive (it leads to more allocation in the name of less).
A user who does not understand why literals are frozen is unlikely to even notice that interpolated ones are not. And if he does and wonders why, he will, if persistent, arrive at the reason (interning) and be better off for it. The foolish, in this case, consistency in the name of "simplicity" helps no one.

#12 - 08/11/2020 06:22 AM - duerst (Martin Dürst)
bughit (bug hit) wrote in #note-11:

However in my view what defines a literal, is the use of a specific syntax, so ""

There is only one reason for freezing literals, to be able to intern them and reduce allocation. In fact the feature is poorly named, after the consequence(freezing), not the cause (interning).

My understanding is that another reason is avoidance of alias effects. It's easy to write code that when cut down to the essential, does this:

```ruby
a = b = "My string"
a.gsub!(/My/, 'Your')
```

and expects b to still be "My string". Freezing makes sure this throws an error.

(This is not an argument for or against freezing interpolated strings.)

#13 - 08/11/2020 03:57 PM - bughit (bug hit)

another reason is avoidance of alias effects

What you've shown is not another reason for freezing.

```ruby
a = b = "My string"
```

both a and b refer to the same string object regardless of interning/freezing
there's no expectation that mutating it via a will not affect b

the interning scenario is:

```ruby
a = "My string"
b = "My string"
a.gsub!(/My/, 'Your')
```

here there's an appearance of 2 string objects but when they are interned, there's only one, so mutation can not be allowed. As I said, interning is the feature, and it requires freezing.

#14 - 08/15/2020 10:47 AM - sawa (Tsuyoshi Sawada)
- Description updated

#15 - 08/26/2020 07:43 AM - akr (Akira Tanaka)
Non-freezing interpolated strings is good thing to reduce allocations.
I think it is possible if most developers understands difference of interpolated and non-interpolated strings.

#16 - 08/31/2020 08:50 AM - matz (Yukihiro Matsumoto)
OK. Persuaded. Make them unfrozen.

Matz.

#17 - 08/31/2020 07:28 PM - Eregon (Benoit Daloze)
- Assignee set to Eregon (Benoit Daloze)
I'll try to make a PR for this change: https://github.com/ruby/ruby/pull/3488

#18 - 09/01/2020 03:39 PM - mame (Yusuke Endoh)
Eregon (Benoit Daloze) wrote in #note-17:

I'll try to make a PR for this change: https://github.com/ruby/ruby/pull/3488
Thanks, I've given it a try.

I found "foo#{ "foo" }" frozen because it is optimized to "foofoo" at the parser. What do you think?

```
$ ./miniruby --enable-frozen-string-literal -e 'p "foo#{ "foo" }".frozen?'
true
```

#19 - 09/01/2020 06:18 PM - Eregon (Benoit Daloze)
mame (Yusuke Endoh) wrote in #note-18:

I found "foo#{ "foo" }" frozen because it is optimized to "foofoo" at the parser. What do you think?

I guess that's semantically correct (besides the frozen status), since interpolation does not need to call #to_s for a String. Since such code is very unlikely to appear in real code, I think it ultimately does not matter too much.

But if we can easily remove that optimization in the parser, I think it would be better, because this is an inconsistency (it makes it harder to reason about Ruby semantics & it breaks referential transparency) and optimizing "foo#{ "foo" }" seems to have no use in practice. Could you point me to where the optimization is done in the parser if you found it? :)

#20 - 09/02/2020 02:15 PM - nobu (Nobuyoshi Nakada)
https://github.com/nobu/ruby/tree/unfrozen-literal

#21 - 09/03/2020 12:29 AM - ko1 (Koichi Sasada)
I found that freezing interpolated strings are help for Ractor programming with constants.

```
class C
  i = 10
  STR = "foo#{i}"
end
```

#22 - 09/03/2020 02:26 PM - Eregon (Benoit Daloze)
ko1 (Koichi Sasada) wrote in #note-21:

I found that freezing interpolated strings are help for Ractor programming with constants.

Right, anything deeply frozen is helpful for Ractor.
But interpolated Strings are probably not that common.
I see an interpolated String much like an Array literals, and those are not frozen without an explicit .freeze.
Related comment: https://bugs.ruby-lang.org/issues/17100#note-23

#23 - 09/15/2020 07:32 PM - Eregon (Benoit Daloze)
- Status changed from Open to Closed

Applied in changeset git\|9b9b535f3ff7c2148e34dd44564df7ade723881276.

Interpolated strings are no longer frozen with frozen-string-literal: true

- Remove freezestring instruction since this was the only usage for it.
- [Feature #17104]

#24 - 09/15/2020 07:35 PM - Eregon (Benoit Daloze)
I merged https://github.com/ruby/ruby/pull/3468 so it's in time for preview1.
nobu (Nobuyoshi Nakada) Could you decide if you think https://github.com/nobu/ruby/tree/unfrozen-literal is worth it?

#25 - 09/15/2020 07:36 PM - Eregon (Benoit Daloze)
- Target version set to 3.0

#26 - 12/16/2020 05:34 PM - marcandre (Marc-Andre Lafortune)
- Is duplicate of Misc #16047: Reconsider impact of frozen_string_literal on dynamic strings added