Ruby - Feature #20205

Enable `frozen_string_literal` by default

01/23/2024 03:26 PM - byroot (Jean Boussier)

Status:	Assigned
Priority:	Normal
Assignee:	matz (Yukihiro Matsumoto)
Target version:	

Description

Context

The frozen_string_literal: true pragma was introduced in Ruby 2.3, and as far as I'm aware the plan was initially to make it the default for Ruby 3.0, but this plan was abandoned because it would be too much of a breaking change without any real further notice.

According to Matz, he still wishes to enable frozen_string_literal by default in the future, but a reasonable migration plan is required.

The main issue is backward compatibility, flipping the switch immediately would break a lot of code, so there must be some deprecation period.

The usual the path forward for this kind of change is to emit deprecation warnings one of multiple versions in advance.

One example of that was the Ruby 2.7 keyword argument deprecation. It was quite verbose, and some users were initially annoyed, but I think the community pulled through it and I don't seem to hear much about it anymore.

So for frozen string literals, the first step would be to start warning when a string that would be frozen in the future is mutated.

Deprecation Warning Implementation

I implemented a quick proof of concept with @etienne in https://github.com/Shopify/ruby/pull/549

In short:

- Files with # frozen_string_literal: true or # frozen_string_literal: false don't change in behavior at all.
- Files with no # frozen_string_literal comment are compiled to use putchilledstring opcode instead of regular putstring.
- This opcode mark the string with a user flag, when these strings are mutated, a warning is issued.

Currently the proof of concept issue the warning at the mutation location, which in some case can make locating where the string was allocated a bit hard.

But it is possible to improve it so the message also include the location at which the literal string was allocated, and learning from the keyword argument warning experience,
we can record which warnings were already issued to avoid spamming users with duplicated warnings.

As currently implemented, there is almost no overhead. If we modify the implementation to record the literal location,
we'd incur a small memory overhead for each literal string in a file without an explicit frozen_string_literal pragma.

But I believe we could do it in a way that has no overhead if Warning[:deprecated] = false.

Timeline

The migration would happen in 3 steps, each step can potentially last multiple releases. e.g. R0 could be 3.4, R1 be 3.7 and R2 be 4.0.
I don't have a strong opinion on the pace.

- Release R0: introduce the deprecation warning (only if deprecation warnings enabled).
- Release R1: make the deprecation warning show up regardless of verbosity level.
- Release R2: make string literals frozen by default.

Impact

Given that rubocop is quite popular in the community and it has enforced the usage of # frozen_string_literal: true for years now,
I suspect a large part of the actively maintained codebases in the wild wouldn't see any warnings.

And with recent versions of minitest enabling deprecation warnings by default (and potentially RSpec too),
the few that didn't migrate will likely be made compatible quickly.

The real problem of course are the less actively developed libraries and applications. For such cases, any codebase can remain compatible by setting RUBYOPT="--disable=frozen_string_literal",
and so even after R2 release. The flag would never be removed any legacy codebase can continue upgrading Ruby without changing a single line of cod by just flipping this flag.

Workflow for library maintainers

As a library maintainer, fixing the deprecation warnings can be as simple as prepending # frozen_string_literal: false at the top of all their source files, and this will keep working forever.

Alternatively they can of course make their code compatible with frozen string literals.

Code that is frozen string literal compatible doesn't need to explicitly declare it. Only code that need it turned of need to do so.

Workflow for application owners

For application owners, the workflow is the same than for libraries.

However if they depend on a gem that hasn't updated, or that they can't upgrade it, they can run their application with RUBYOPT="--disable=frozen_string_literal" and it will keep working forever.

Any user running into an incompatibility issue can set RUBYOPT="--disable=frozen_string_literal" forever, even in 4.x, the only thing changing is the default value.

And any application for which all dependencies have been made fully frozen string literal compatible can set RUBYOPT="--enable=frozen_string_literal" and start immediately removing magic comment from their codebase.

Related issues:

Related to Ruby - Feature #11473: Immutable String literal in Ruby 3	**Closed**
Related to Ruby - Feature #20390: Issue with StringIO and chilled strings	**Closed**

Associated revisions

Revision 12be40ae6be78ac41e8e3f3c313cc6f63e7fa6c4 - 03/19/2024 08:26 AM - etienne (Étienne Barrié)

Implement chilled strings

[Feature #20205]

As a path toward enabling frozen string literals by default in the future,
this commit introduce "chilled strings". From a user perspective chilled
strings pretend to be frozen, but on the first attempt to mutate them,
they lose their frozen status and emit a warning rather than to raise a
FrozenError.

Implementation wise, rb_compile_option_struct.frozen_string_literal is
no longer a boolean but a tri-state of enabled/disabled/unset.

When code is compiled with frozen string literals neither explictly enabled
or disabled, string literals are compiled with a new putchilledstring
instruction. This instruction is identical to putstring except it marks
the String with the STR_CHILLED (FL_USER3) and FL_FREEZE flags.

Chilled strings have the FL_FREEZE flag as to minimize the need to check
for chilled strings across the codebase, and to improve compatibility with
C extensions.

Notes:

- String#freeze: clears the chilled flag.
- String#-@: acts as if the string was mutable.
- String#+@: acts as if the string was mutable.
- String#clone: copies the chilled flag.

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 12be40ae6be78ac41e8e3f3c313cc6f63e7fa6c4 - 03/19/2024 08:26 AM - etienne (Étienne Barrié)

Implement chilled strings

[Feature #20205]

As a path toward enabling frozen string literals by default in the future,
this commit introduce "chilled strings". From a user perspective chilled
strings pretend to be frozen, but on the first attempt to mutate them,
they lose their frozen status and emit a warning rather than to raise a
FrozenError.

Implementation wise, rb_compile_option_struct.frozen_string_literal is
no longer a boolean but a tri-state of enabled/disabled/unset.

When code is compiled with frozen string literals neither explictly enabled
or disabled, string literals are compiled with a new putchilledstring
instruction. This instruction is identical to putstring except it marks
the String with the STR_CHILLED (FL_USER3) and FL_FREEZE flags.

Chilled strings have the FL_FREEZE flag as to minimize the need to check
for chilled strings across the codebase, and to improve compatibility with
C extensions.

Notes:

- String#freeze: clears the chilled flag.
- String#-@: acts as if the string was mutable.
- String#+@: acts as if the string was mutable.
- String#clone: copies the chilled flag.

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 12be40ae - 03/19/2024 08:26 AM - etienne (Étienne Barrié)

Implement chilled strings

[Feature #20205]

As a path toward enabling frozen string literals by default in the future,
this commit introduce "chilled strings". From a user perspective chilled
strings pretend to be frozen, but on the first attempt to mutate them,
they lose their frozen status and emit a warning rather than to raise a
FrozenError.

Implementation wise, rb_compile_option_struct.frozen_string_literal is
no longer a boolean but a tri-state of enabled/disabled/unset.

When code is compiled with frozen string literals neither explictly enabled
or disabled, string literals are compiled with a new putchilledstring
instruction. This instruction is identical to putstring except it marks
the String with the STR_CHILLED (FL_USER3) and FL_FREEZE flags.

Chilled strings have the FL_FREEZE flag as to minimize the need to check
for chilled strings across the codebase, and to improve compatibility with
C extensions.

Notes:

- String#freeze: clears the chilled flag.
- String#-@: acts as if the string was mutable.
- String#+@: acts as if the string was mutable.
- String#clone: copies the chilled flag.

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 2b08406cd0db0042520fb0346544660e10a4d93c - 03/26/2024 11:54 AM - etienne (Étienne Barrié)

Expose rb_str_chilled_p

Some extensions (like stringio) may need to differentiate between
chilled strings and frozen strings.

They can now use rb_str_chilled_p but must check for its presence since
the function will be removed when chilled strings are removed.

[Bug #20389]

[Feature #20205]

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 2b08406cd0db0042520fb0346544660e10a4d93c - 03/26/2024 11:54 AM - etienne (Étienne Barrié)

Expose rb_str_chilled_p

Some extensions (like stringio) may need to differentiate between
chilled strings and frozen strings.

They can now use rb_str_chilled_p but must check for its presence since
the function will be removed when chilled strings are removed.

[Bug #20389]

[Feature #20205]

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 2b08406c - 03/26/2024 11:54 AM - etienne (Étienne Barrié)

Expose rb_str_chilled_p

Some extensions (like stringio) may need to differentiate between
chilled strings and frozen strings.

They can now use rb_str_chilled_p but must check for its presence since
the function will be removed when chilled strings are removed.

[Bug #20389]

[Feature #20205]

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 49b31c7680a86413853d0c2ce2124d3cba56d334 - 04/08/2024 11:25 AM - etienne (Étienne Barrié)

Document STR_CHILLED flag on RString

[Feature #20205]

Revision 49b31c7680a86413853d0c2ce2124d3cba56d334 - 04/08/2024 11:25 AM - etienne (Étienne Barrié)

Document STR_CHILLED flag on RString

[Feature #20205]

Revision 49b31c76 - 04/08/2024 11:25 AM - etienne (Étienne Barrié)

Document STR_CHILLED flag on RString

[Feature #20205]

Revision 1376881e9afe6ff673f64afa791cf30f57147ee2 - 05/28/2024 05:32 AM - etienne (Étienne Barrié)

Stop marking chilled strings as frozen

They were initially made frozen to avoid false positives for cases such
as:

```
str = str.dup if str.frozen?
```

But this may cause bugs and is generally confusing for users.

[Feature #20205]

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 1376881e9afe6ff673f64afa791cf30f57147ee2 - 05/28/2024 05:32 AM - etienne (Étienne Barrié)

Stop marking chilled strings as frozen

They were initially made frozen to avoid false positives for cases such
as:

```
str = str.dup if str.frozen?
```

But this may cause bugs and is generally confusing for users.

[Feature #20205]

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 1376881e - 05/28/2024 05:32 AM - etienne (Étienne Barrié)

Stop marking chilled strings as frozen

They were initially made frozen to avoid false positives for cases such as:

```
str = str.dup if str.frozen?
```

But this may cause bugs and is generally confusing for users.

[Feature #20205]

Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 15501e13d7588a049437d343548bda76121b92f9 - 05/30/2024 03:11 PM - byroot (Jean Boussier)

[ruby/stringio] Remove special handling of chilled strings

[Feature #20205]

Followup: https://github.com/ruby/stringio/pull/94

They no longer need to be special cases. If StringIO end up mutating a chilled string, a warning will be emitted.

https://github.com/ruby/stringio/commit/dc62d65449

Revision 15501e13d7588a049437d343548bda76121b92f9 - 05/30/2024 03:11 PM - byroot (Jean Boussier)

[ruby/stringio] Remove special handling of chilled strings

[Feature #20205]

Followup: https://github.com/ruby/stringio/pull/94

They no longer need to be special cases. If StringIO end up mutating a chilled string, a warning will be emitted.

https://github.com/ruby/stringio/commit/dc62d65449

Revision 15501e13 - 05/30/2024 03:11 PM - byroot (Jean Boussier)

[ruby/stringio] Remove special handling of chilled strings

[Feature #20205]

Followup: https://github.com/ruby/stringio/pull/94

They no longer need to be special cases. If StringIO end up mutating a chilled string, a warning will be emitted.

https://github.com/ruby/stringio/commit/dc62d65449

Revision 730e3b2ce01915c4a98b79bb281b2c38a9ff1131 - 06/02/2024 11:53 AM - byroot (Jean Boussier)

Stop exposing rb_str_chilled_p

[Feature #20205]

Now that chilled strings no longer appear as frozen, there is no need to offer an API to check for chilled strings.

We however need to change rb_check_frozen_internal to no longer be a macro, as it needs to check for chilled strings.

Revision 730e3b2ce01915c4a98b79bb281b2c38a9ff1131 - 06/02/2024 11:53 AM - byroot (Jean Boussier)

Stop exposing rb_str_chilled_p

[Feature #20205]

Now that chilled strings no longer appear as frozen, there is no
need to offer an API to check for chilled strings.

We however need to change rb_check_frozen_internal to no
longer be a macro, as it needs to check for chilled strings.

Revision 730e3b2c - 06/02/2024 11:53 AM - byroot (Jean Boussier)

Stop exposing rb_str_chilled_p

[Feature #20205]

Now that chilled strings no longer appear as frozen, there is no
need to offer an API to check for chilled strings.

We however need to change rb_check_frozen_internal to no
longer be a macro, as it needs to check for chilled strings.

Revision 95ffcd3f9ff20c3e9b0556672758cf8724542b0c - 06/24/2024 10:43 AM - byroot (Jean Boussier)

Fix --debug-frozen-string-literal to not apply --disable-frozen-string-literal

[Feature #20205]

This was an undesired side effect. Now that this value is a triplet, we can't
assume it's disabled by default.

Revision 95ffcd3f9ff20c3e9b0556672758cf8724542b0c - 06/24/2024 10:43 AM - byroot (Jean Boussier)

Fix --debug-frozen-string-literal to not apply --disable-frozen-string-literal

[Feature #20205]

This was an undesired side effect. Now that this value is a triplet, we can't
assume it's disabled by default.

Revision 95ffcd3f - 06/24/2024 10:43 AM - byroot (Jean Boussier)

Fix --debug-frozen-string-literal to not apply --disable-frozen-string-literal

[Feature #20205]

This was an undesired side effect. Now that this value is a triplet, we can't
assume it's disabled by default.

Revision 257f78fb671151f1db06dcd8e35cf4cc736f735e - 10/21/2024 10:33 AM - etienne (Étienne Barrié)

Show where mutated chilled strings were allocated

[Feature #20205]

The warning now suggests running with --debug-frozen-string-literal:

```
test.rb:3: warning: literal string will be frozen in the future (run with --debug-frozen-string-literal for mo
re information)
```

When using --debug-frozen-string-literal, the location where the string
was created is shown:

```
test.rb:3: warning: literal string will be frozen in the future
test.rb:1: info: the string was created here
```

When resurrecting strings and debug mode is not enabled, the overhead is a simple FL_TEST_RAW.
When mutating chilled strings and deprecation warnings are not enabled,
the overhead is a simple warning category enabled check.

Co-authored-by: Jean Boussier byroot@ruby-lang.org
Co-authored-by: Nobuyoshi Nakada nobu@ruby-lang.org
Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 257f78fb671151f1db06dcd8e35cf4cc736f735e - 10/21/2024 10:33 AM - etienne (Étienne Barrié)

Show where mutated chilled strings were allocated

[Feature #20205]

The warning now suggests running with --debug-frozen-string-literal:

```
test.rb:3: warning: literal string will be frozen in the future (run with --debug-frozen-string-literal for mo
re information)
```

When using --debug-frozen-string-literal, the location where the string
was created is shown:

```
test.rb:3: warning: literal string will be frozen in the future
test.rb:1: info: the string was created here
```

When resurrecting strings and debug mode is not enabled, the overhead is a simple FL_TEST_RAW.
When mutating chilled strings and deprecation warnings are not enabled,
the overhead is a simple warning category enabled check.

Co-authored-by: Jean Boussier byroot@ruby-lang.org
Co-authored-by: Nobuyoshi Nakada nobu@ruby-lang.org
Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision 257f78fb - 10/21/2024 10:33 AM - etienne (Étienne Barrié)

Show where mutated chilled strings were allocated

[Feature #20205]

The warning now suggests running with --debug-frozen-string-literal:

```
test.rb:3: warning: literal string will be frozen in the future (run with --debug-frozen-string-literal for mo
re information)
```

When using --debug-frozen-string-literal, the location where the string
was created is shown:

```
test.rb:3: warning: literal string will be frozen in the future
test.rb:1: info: the string was created here
```

When resurrecting strings and debug mode is not enabled, the overhead is a simple FL_TEST_RAW.
When mutating chilled strings and deprecation warnings are not enabled,
the overhead is a simple warning category enabled check.

Co-authored-by: Jean Boussier byroot@ruby-lang.org
Co-authored-by: Nobuyoshi Nakada nobu@ruby-lang.org
Co-authored-by: Jean Boussier byroot@ruby-lang.org

Revision c3b06792410e99c32ff5c05580ffaef7d5f377c4 - 12/13/2024 07:25 PM - alanwu (Alan Wu)

[DOC] NEWS: Mention String#+@ change with chilled string

From experience, a not insignificant number of people run into this when
trying upgrade. Also it's good to bring up +@ because it's relevant in
general to the topic.

[Feature #20205]

Revision 880a90cf2e56f9782b578fd48088c184bf3952ac - 12/13/2024 07:25 PM - alanwu (Alan Wu)

[DOC] [Feature #20205] Document the new power of String#+@

Revision c3b06792410e99c32ff5c05580ffaef7d5f377c4 - 12/13/2024 07:25 PM - alanwu (Alan Wu)

[DOC] NEWS: Mention String#+@ change with chilled string

From experience, a not insignificant number of people run into this when
trying upgrade. Also it's good to bring up +@ because it's relevant in
general to the topic.

[Feature #20205]

Revision 880a90cf2e56f9782b578fd48088c184bf3952ac - 12/13/2024 07:25 PM - alanwu (Alan Wu)

[DOC] [Feature #20205] Document the new power of String#+@

Revision c3b06792 - 12/13/2024 07:25 PM - alanwu (Alan Wu)

[DOC] NEWS: Mention String#+@ change with chilled string

From experience, a not insignificant number of people run into this when trying upgrade. Also it's good to bring up +@ because it's relevant in general to the topic.

[Feature #20205]

Revision 880a90cf - 12/13/2024 07:25 PM - alanwu (Alan Wu)

[DOC] [Feature #20205] Document the new power of String#+@

History

#1 - 01/23/2024 06:32 PM - matheusrich (Matheus Richard)

> Given that rubocop is quite popular in the community and it has enforced the usage of # frozen_string_literal: true for years now, I suspect a large part of the actively maintained codebases in the wild wouldn't see any warnings.

That's true, but because standardrb doesn't enforce it (and many folks have been defaulting to that), I've seen several codebases not consistently using the pragma.

#2 - 01/24/2024 04:38 AM - mame (Yusuke Endoh)

I think we should evaluate the value of # frozen_string_literal: true before making it the default.

The purpose of # frozen_string_literal: true is to make Ruby code fast and memory-saving. When it was introduced, no quantitative evaluation was available except for micro-benchmarks, because most code did not support frozen_string_literal.

Now that there are many gems supporting # frozen_string_literal: true. So we can evaluate it. For example, how much would the performance degrade if we removed # frozen_string_literal: true from all code used in yjit-bench?

If the degradation is large enough, then making # frozen_string_literal: true the default may be worth serious consideration. If the degradation is negligible, it would be reasonable for Rubocop to stop enforcing # frozen_string_literal: true and remove the magic comments from existing gems as well.

#3 - 01/24/2024 09:34 AM - zverok (Victor Shepelev)

> The purpose of # frozen_string_literal: true is to make Ruby code fast and memory-saving. When it was introduced, no quantitative evaluation was available except for micro-benchmarks, because most code did not support frozen_string_literal.

I believe that whatever the initial intention, the "frozen string literals" concept being adopted by many codebases is also a cultural thing, not only performance-related.

The "string literals are frozen by default" changes the way we program, if even slightly. That's actually a long-standing topic to have more impactful freezing (of the constants, for example), but frozen string literals *at the very least* prevent trivial errors like

```
HEADER = "<html><body>"

def generate
  output = HEADER # no `.dup`
  output << "<p>test</p>" # actually changed HEADER
  # ...
end
```

#4 - 01/24/2024 09:44 AM - byroot (Jean Boussier)

> I think we should evaluate the value of # frozen_string_literal: true before making it the default.

So it's not 100% reliable because I ran it locally rather than on a benchmarking server, as evidenced by some strange effect on benchmarks that don't normally deal with strings (e.g. setivar_object), but here are the results:

```
mutable: ruby 3.4.0dev (2024-01-24T08:24:16Z disable-frozen-str.. a39d5eae1e) [arm64-darwin23]
frozen: ruby 3.4.0dev (2024-01-24T08:24:16Z disable-frozen-str.. bb0cee8dab) [arm64-darwin23]
```

bench	mutable (ms)	stddev (%)	frozen (ms)	stddev (%)	frozen 1st itr	mutable/frozen
activerecord	32.9	4.0	32.7	5.3	1.06	1.01
chunky-png	578.0	1.1	555.2	1.0	1.01	1.04
erubi-rails	1124.2	2.4	1158.7	1.7	0.93	0.97
hexapdf	1703.9	2.7	1670.2	2.2	1.04	1.02
liquid-c	34.3	5.7	34.1	5.0	1.02	1.01
liquid-compile	39.8	4.2	38.0	5.3	0.99	1.05
liquid-render	97.2	3.4	97.7	3.5	1.01	0.99
lobsters	637.8	5.2	614.9	4.2	1.11	1.04
mail	82.7	3.7	82.1	3.7	1.00	1.01
psych-load	1500.8	0.8	1487.6	0.5	0.99	1.01
railsbench	1116.3	2.1	1099.8	1.6	1.02	1.01
rubocop	112.0	3.8	111.2	3.7	1.03	1.01
ruby-lsp	79.5	2.6	80.3	2.4	0.92	0.99
sequel	36.4	4.4	36.5	3.3	0.95	1.00
binarytrees	238.4	1.9	238.8	1.7	0.99	1.00
blurhash	281.9	1.4	282.5	1.7	0.99	1.00
erubi	166.1	2.0	171.5	2.5	0.97	0.97
etanni	198.7	2.7	200.5	1.8	1.04	0.99
fannkuchredux	2065.3	0.7	2078.5	0.5	0.99	0.99
fluentd	1192.0	1.0	1202.7	0.7	1.03	0.99
graphql	2323.6	0.7	2340.5	0.5	0.98	0.99
graphql-native	385.3	1.5	384.2	2.1	1.00	1.00
lee	739.3	1.8	750.1	1.2	0.98	0.99
matmul	1494.2	0.8	1497.8	0.9	1.01	1.00
nbody	71.1	2.3	71.3	5.1	1.00	1.00
nqueens	169.8	1.6	167.6	2.0	1.00	1.01
optcarrot	4202.7	0.6	4216.8	0.5	1.00	1.00
rack	56.0	3.3	53.9	3.9	1.06	1.04
ruby-json	1976.3	0.9	1992.4	0.3	1.00	0.99
rubykon	6971.3	0.8	7053.4	0.4	1.00	0.99
sudoku	1836.1	0.4	1836.3	0.3	1.00	1.00
tinygql	441.4	1.0	447.4	1.2	0.96	0.99
30k_ifelse	1459.0	8.6	1429.1	4.6	0.96	1.02
30k_methods	3331.8	3.7	3264.0	1.1	1.03	1.02
cfunc_itself	81.5	1.8	81.2	2.9	1.02	1.00
fib	187.0	1.6	188.8	1.5	0.94	0.99
getivar	65.1	2.2	65.7	2.4	1.01	0.99
keyword_args	141.6	2.3	140.9	2.1	1.00	1.01
respond_to	184.9	1.9	185.7	1.1	0.96	1.00
setivar	39.2	2.9	39.2	2.4	1.03	1.00
setivar_object	72.7	2.5	77.5	2.1	0.95	0.94
setivar_young	72.5	1.9	77.5	0.9	0.93	0.94
str_concat	69.7	2.5	69.9	1.9	1.02	1.00
throw	14.9	5.0	14.8	4.5	1.04	1.01

Legend:
- frozen 1st itr: ratio of mutable/frozen time for the first benchmarking iteration.
- mutable/frozen: ratio of mutable/frozen time. Higher is better for frozen. Above 1 represents a speedup.

I used the following patch to disable frozen string literals globally: https://github.com/ruby/ruby/compare/master...Shopify:ruby:disable-frozen-string-literal, and ran the suite with ruby run_benchmarks.rb --chruby 'mutable::head-mutable-strings;frozen::head'.

Most of these benchmarks don't really deal with string, but one that I think is the most close to reality and to Ruby bread and butter is lobsters and it seem to be quite positive.

I also didn't enable YJIT, I suspect the difference would be bigger if I did, as the extra GC pressure would be relatively bigger.

I'll see about using the yjit-perf benchmark server to run these more scientifically.

I also want to note that I have to explictly freeze RUBY_DESCRIPTION in my patch, otherwise it would cause a Ractor issue, which is another argument for frozen strings, as they ease the necessary work necessary to make code Ractor compatible.

#5 - 01/24/2024 09:46 AM - byroot (Jean Boussier)

> but because standardrb doesn't enforce it (and many folks have been defaulting to that), I've seen several codebases not consistently using the pragma.

Yes, I didn't extend on my motivations for bringing this now, but It's in big part influenced by standardrb and some other communities starting to discourage the use of frozen string literals. I suspect many people are starting to be fed up with that magic comment and we probably reached peak

usage of it.

#6 - 01/24/2024 11:11 AM - duerst (Martin Dürst)

frozen_string_literal can lead to more efficient code, but it's also part of a programming style (usually called functional programming). Functional programming often leads to cleaner code, but it may require some additional programming effort. There's also a fundamental conflict between object-oriented programming (objects are generally mutable) and functional programming, although Ruby is pretty good at integrating these concepts.

My main issue with this proposal is that I think it's probably the right thing for most big code bases, but it may not be the right thing for quick-and-dirty small scripts. And Ruby is used, and should continue to be usable, for both kinds of code.

Maybe what could help is a declaration on a higher level, e.g. per gem or so rather than per source file. (That's just an idea, with many open questions: Where would the setting go? How would the interpreter pick it up? How would people become aware of it? ...)

#7 - 01/24/2024 11:12 AM - Eregon (Benoit Daloze)

+1 to this issue proposal, it sounds like a good plan.
And agreed with @zverok (Victor Shepelev), having frozen literals by default is not only faster and less wasteful in memory usage, but also a safer model, where "String used like buffers/mutated Strings" are more explicit, which certainly seems good for readability and avoiding to accidentally mutate a String which should not be mutated.

#8 - 01/24/2024 11:14 AM - byroot (Jean Boussier)

> Maybe what could help is a declaration on a higher level, e.g. per gem or so rather than per source file

That's my fallback proposal if this one doesn't go through. Devise a way to set compile options for all files inside a directory, and then integrate with Rubygems to enable frozen string literals on a per gem basis.

But I'd prefer to just flip the default if possible.

#9 - 01/24/2024 02:22 PM - mame (Yusuke Endoh)

@byroot (Jean Boussier) Thanks for the quick benchmark! It depends on further benchmarking, but currently, I don't see enough advantage over compatibility. Do you think it is worth the risk of compatibility?

@zverok (Victor Shepelev) Ah, that's exactly why I was against the introduction of frozen_string_literal, because people misunderstand it like you. This feature must not be mixed up with "immutability". I can understand if all String objects were frozen by default, but freezing only string "literals" makes no sense at all (except in terms of performance). Consider, just adding .upcase or .b or something to a String literal makes it mutable. I don't find "a cultural thing" in such a fragile thing.

#10 - 01/24/2024 03:34 PM - byroot (Jean Boussier)

I'm still trying to get a hold onto the benchmarking server, but the 5% figure on lobsters seems to be quite consistent on my machine, so I think I can use it for my argumentation.

> It depends on further benchmarking, but currently, I don't see enough advantage over compatibility. Do you think it is worth the risk of compatibility?

Alright, so I fear my answer to this will be longer than expected.

So I'm of course biased because the type of workload I work with are very heavily string based (Web), so for me, a 5% improvement (to be confirmed) is quite significant.

And also because while we have many dependencies (over 700 transitive gems in the monolith), we do make sure to prune or replace the abandoned ones, and have the habit to contribute to them regularly. So I'm quite confident we can get all our dependencies ready for this change without much work and in short order, It will certainly be much less work than the Ruby 2.7 keyword argument change was.

Now of course, for Ruby users that don't generally deal with Strings much, this is just yet another annoying change that don't give them anything substantial.

But also retaining compatibility for them is really trivial. If you are running a legacy code base or outdated dependency, but yet are upgrading to a newer Ruby, is it really that much work to just set RUBYOPT="--disable=frozen_string_literal" an move on? Especially after multiple versions warning you that you'll have to do it at some point? Is that legacy code even still working on 3.x after the keyword argument change?

Maybe my bias cause me to downplay the compatibility concern, but it really doesn't seem significant to me assuming a reasonably long timeline so the ecosystem have time to prepare and catch up.

#11 - 01/24/2024 04:28 PM - jeremyevans0 (Jeremy Evans)

byroot (Jean Boussier) wrote in #note-10:

> Is that legacy code even still working on 3.x after the keyword argument change?

It depends on what you consider legacy code. Legacy code designed for Ruby 1.8 or 1.9, which never used keyword arguments, was unaffected by the keyword argument changes in Ruby 3.

#12 - 01/24/2024 05:14 PM - Dan0042 (Daniel DeLorme)

> I think the community pulled through it and I don't seem to hear much about it anymore.

Careful; the community "pulled through" because there was already a lot of accumulated good will, and the 2.7 migration burned through some of that reserve. Yet another migration might result in "not again!" syndrome and the community not pulling through nearly as well. It depends on how annoying the migration is, and the perceived benefit.

I should note that since dynamic string literals are no longer frozen since 3.0, the disruption would be that much smaller.

- Files with no # frozen_string_literal comment are compiled to use putchilledstring opcode instead of regular putstring.
- This opcode mark the string with a user flag, when these strings are mutated, a warning is issued.

That's a lot like #16153 so I like it. I would also like to have # frozen_string_literal: chilled to enable this behavior on my own terms, without impacting gems over which I have no control.

> learning from the keyword argument warning experience

I think another important lesson from that experience is that gems are different from app code. If I want to optimize my app to use frozen string literals, I have to enable this warning at the global level, and then if warnings from gems are mixed in it makes my job a lot more annoying. For all kinds of reasons I do not want to update my apps and gems at the same time.

> Maybe what could help is a declaration on a higher level, e.g. per gem or so rather than per source file

It's a bit similar to #17156 so I like it. Actually I would much prefer this than changing the default; it allows every app and gem to upgrade on their own terms, without enforcing a one-size-fits-all default, and without the noise of a pragma in every file. Especially if combined with frozen_string_literal: chilled it would be very empowering.

> But also retaining compatibility for them is really trivial. If you are running a legacy code base or outdated dependency, but yet are upgrading to a newer Ruby, is it really that much work to just set RUBYOPT="--disable=frozen_string_literal" an move on?

Conversely let me ask: Is it really that much work to just set RUBYOPT="--enable=frozen_string_literal" in your application instead of forcing a new default on everyone else?

#13 - 01/24/2024 05:29 PM - rubyFeedback (robert heiler)

I think this was discussed or suggested before. Personally
I have all my .rb files with one header line being:

```
# frozen_string_literal: true
```

so a change towards frozen string literals would not affect
me, as I already use that by default, since many years
actually.

So the question, then, is, how this may affect other ruby
users and developers.

> One example of that was the Ruby 2.7 keyword argument
> deprecation. It was quite verbose, and some users were
> initially annoyed, but I think the community pulled
> through it and I don't seem to hear much about it anymore.

This is not quite how I remember it; I think there were tons
of warnings initially, in particular from rails code bases,
and some changes were made to the warning situation.

zverok wrote:

> I believe that whatever the initial intention, the
> "frozen string literals" concept being adopted by

> many codebases is also a cultural thing, not only performance-related.

Well, I mostly added it for consistency reasons, and also to make it easier to transition and upgrade to new ruby releases. So for me, it just was easier to add it to all my .rb files, via a standard header I keep on re-using in my .rb files. Not sure if it is a cultural thing, but for me it is more a habit.

zverok wrote:

> The "string literals are frozen by default" changes the way we program, if even slightly.

Yep, that is very true.

In oldschool ruby I could just do:

```
x = 'foo'
x << 'bar'
```

I prefer the oldschool ruby variant. But I understand the performance reasons for frozen Strings, so it's a trade off. That's why I think being able to tell ruby to not use frozen strings would be nice, if only of reason of nostalgia, as I liked the oldschool behaviour more.

The way I solve this is actually solely via .frozen? checks and then .dup. There are other shorter ways (nobu showed that), but for some odd reason I like the more explicit check. Even then, naturally, I prefer the oldschool ruby way without having to check on the frozen status. This is just nostalgia, though - I moved towards every String object being frozen by default in my own ruby code.

> some other communities starting to discourage the use of frozen string literals. I suspect many people are starting to be fed up with that magic comment and we probably reached peak usage of it.

I am not fed up with the comment. I'd actually be more fed up with it if it were removed, and frozen_string_literal: false no longer being possible. Even though I no longer use it myself. :P

> There's also a fundamental conflict between object-oriented programming (objects are generally mutable) and functional programming, although Ruby is pretty good at integrating these concepts.

I don't see that distinction really. Personally I follow more of the object-definition used by Alan Kay, who in turn used a more closer model inspired from molecular biology (and, extending this, Erlang's model could be to have objects everywhere, even though Erlang is not an OOP language).

What is an OOP language? What is a functional language? Ruby kind of is mostly OOP but there-is-more-than-one-way-to-do-it, so ruby is a bit of a hybrid language, even if I'd say it is heavily leaning towards OOP. So what is the ideal functional language? Haskell? Is a monad an object? Can it walk on a moebius strip without falling down?

martin wrote:

> My main issue with this proposal is that I think it's probably the right thing for most big code bases, but it may not be the right thing for quick-and-dirty small scripts. And Ruby is used, and should continue to be usable, for both kinds of code.

Well, I don't disagree necessarily. But if frozen_string_literal is honoured, people can just use frozen_string_literal: false, right? Even some commandline shorthand notation for that could

be used, such as --fsf (frozen string false) or so. Or people can put that frozen string literal: false in the header of the .rb file. I don't think this should be the primary rationale for not making frozen string literals true NOT the default, though.

martin wrote:

> Maybe what could help is a declaration on a higher level, e.g. per gem or so rather than per source file. (That's just an idea, with many open questions: Where would the setting go? How would the interpreter pick it up? How would people become aware of it? ...)

Not a bad idea in itself; I often wanted to have more fine-tuned control over gems I'd publish. More recommendations by default; people can then ignore that if they want to of course. But I think the question about frozen string literals is a language design question, more than one of gem authors or small-scripts use. So matz should decide on that.

jeremy wrote:

> It depends on what you consider legacy code. Legacy code designed for Ruby 1.8 or 1.9, which never used keyword arguments, was unaffected by the keyword argument changes in Ruby 3.

Yeah. I had in the back of my mind remembering more issues than byroot may recall. I am glad I am not the only one; I mix up things a lot these days.

Daniel wrote:

> Careful; the community "pulled through" because there was already a lot of accumulated good will, and the 2.7 migration burned through some of that reserve. Yet another migration might result in "not again!" syndrome and the community not pulling through nearly as well. It depends on how annoying the migration is, and the perceived benefit.

Agreed. Although often the perceived benefit is small for many folks. In my own code, having frozen Strings by default probably led to a noticeable speed-up, but initially it was quite some work to adjust my code base to it. Even then I still prefer the oldschool ruby behaviour :D - imagine if there would be no speed penalty when String objects would be mutable at all times.

Daniel wrote:

> Actually I would much prefer this than changing the default; it allows every app and gem to upgrade on their own terms, without enforcing a one-size-fits-all default, and without the noise of a pragma in every file. Especially if combined with frozen_string_literal: chilled it would be very empowering.

Also understandable. Although, I find "the noise of a pragma in every file" actually less annoying than having to do an object.frozen? check. ;)

Even then I still think this is a language design decision to be made (either way how it goes). The flexibility situation you refer to, be it "chilled" or anything else, as well as more fine-tuned control over gems, and what not, is a secondary question. First should come the "enable frozen strings by default yes/no", as a language design decision, IMO.

Anyway, on the suggestion itself, I am +1, but if I may suggest:

- There should be a clearly documented transition path, e. g. in "2025 this will happen, in 2026 that will happen, in 2027

it is completed" (or something like that) if decided on it.
This may also help for those people who want to adjust towards the frozen string false/true situation, and for whatever reason have not yet. At some point in time even slow movers should be able to adjust (unless they have some reason not to, but in this case they could comment here, if they are made aware of the discussion here).

- The impact on "legacy" ruby code should be evaluated as objectively as possible, and to also get people involved who are using, for instance, frozen string literal: false.

#14 - 01/24/2024 05:46 PM - byroot (Jean Boussier)

> Conversely let me ask: Is it really that much work to just set RUBYOPT="--enable=frozen_string_literal" in your application instead of forcing a new default on everyone else?

The thing is, I can't.

Code that is written for frozen_string_literal: true can generally run with frozen_string_literal: false, but the opposite is not true.

#15 - 01/24/2024 05:46 PM - palkan (Vladimir Dementyev)

byroot (Jean Boussier) wrote in #note-8:

> Devise a way to set compile options for all files inside a directory

Here you go: https://github.com/ruby-next/freezolite 🙂

> integrate with Rubygems to enable frozen string literals on a per gem basis

Smth like spec.frozen_string_literals = true? And during gem registration (setting up a load path), RubyGems can add the path to the *frozen list*, so upon load we can set the compile option. That should work, I think.

#16 - 01/24/2024 06:13 PM - byroot (Jean Boussier)

> Here you go: https://github.com/ruby-next/freezolite 🙂

As mentioned on Reddit when you first published that gem, it's a nice Hack, but I don't think it's quite robust enough. If changing the default isn't accepted and instead we try to make it a per gem configuration, I think Ruby will need to expose a better API to do this in a more reliable and clean way.

#17 - 01/24/2024 06:53 PM - tenderlovemaking (Aaron Patterson)

mame (Yusuke Endoh) wrote in #note-9:

> I can understand if all String objects were frozen by default, but freezing only string "literals" makes no sense at all (except in terms of performance).

For me, freezing string literals is a useful way to catch bugs. When I mutate strings, I am trying to be intentional, purposeful, and isolated.

```
buf = "".b # I intend to mutate this string

name = "Aaron" # I don't intend to mutate this, if I do, it's a bug
```

The places in code where I intend to mutate a string are infrequent and isolated so I am happy to pay a ".dup" tax in order to avoid bugs in other parts of my code. Maybe we could freeze all strings by default in the future, but when we're on the subject of compatibility, I think doing that right now would be far too extreme. Freezing string literals buys us performance (as byroot says, even 5% is great), and *some* safety. The trade is compatibility, but I don't think the trade is worthwhile and not very extreme.

#18 - 01/24/2024 06:56 PM - tenderlovemaking (Aaron Patterson)

> The trade is compatibility, but I don't think the trade is worthwhile and not very extreme.

Sorry, I made a typo. I *do* think the trade is worthwhile, and I *don't* think it's very extreme.

#19 - 01/24/2024 07:42 PM - palkan (Vladimir Dementyev)

byroot (Jean Boussier) wrote in #note-16:

> > Here you go: https://github.com/ruby-next/freezolite □□
>
> As mentioned on Reddit when you first published that gem, it's a nice Hack, but I don't think it's quite robust enough. If changing the default isn't accepted and instead we try to make it a per gem configuration, I think Ruby will need to expose a better API to do this in a more reliable and clean way.

Sure, it must be a part of MRI (and other implementations). Consider it a PoC (though, quite robust and battle-tested in production) and example of how to approach path-based compilation settings. The most important thing here is an API to *wrap* code loading so we can adjust settings on-the-fly (smth like require-hooks).

#20 - 01/24/2024 07:45 PM - palkan (Vladimir Dementyev)

> Files with # frozen_string_literal: true or # frozen_string_literal: false don't change in behavior at all.

There is one use case in which having # frozen_string_literal: true differs from RUBYOPT=--enable=frozen_string_literal today:

```
# frozen_string_literal: true

class B
  attr_reader :name

  def initialize
    @name = "B"
  end

  def eval_name
    instance_eval '@name = "C"'
  end
end

b = B.new

puts b.name.frozen?

b.eval_name

puts b.name.frozen?
```

Results in:

```
$ ruby -v
ruby 3.2.2 (2023-03-30 revision e51014f9c0) [arm64-darwin21]

$ ruby b.rb

true
false

$ RUBYOPT='--enable=frozen_string_literal' ruby b.rb

true
true
```

#21 - 01/24/2024 07:48 PM - kddnewton (Kevin Newton)

> There is one use case in which having # frozen_string_literal: true differs from RUBYOPT=--enable=frozen_string_literal today

I don't think that's a difference, you don't have the magic comment in the eval. Changing it to:

```
instance_eval "# frozen_string_literal: true\n@name = \"C\""
```

makes them both true.

#22 - 01/24/2024 07:49 PM - byroot (Jean Boussier)

That's by design, each eval call is its own file, and it doesn't inherit the pragma from the file where it's invoked, which is logical if you think about it.

But you are right that it is uncommon for users to put # frozen_string_literal: pragmas in evaled code, so this could indeed require some small adjustment for codebases that already define a pragma, but it should be very rare.

#23 - 01/24/2024 07:57 PM - palkan (Vladimir Dementyev)

byroot (Jean Boussier) wrote in #note-22:

> But you are right that it is uncommon for users to put # frozen_string_literal: pragmas in evaled code, so this could indeed require some small adjustment for codebases that already define a pragma, but it should be very rare.

Yeah, that's what I meant. Even if users put # frozen_string_literal: in every file (but not within eval) they still might need to adjust their code. So, I'd suggest covering this edge case in the migration guide (or whatever).

#24 - 01/24/2024 09:36 PM - Dan0042 (Daniel DeLorme)

byroot (Jean Boussier) wrote in #note-14:

> > Conversely let me ask: Is it really that much work to just set RUBYOPT="--enable=frozen_string_literal" in your application instead of forcing a new default on everyone else?
>
> The thing is, I can't.

Sorry, that didn't make sense to me... with a new ruby version having frozen_string_literal enabled by default you are "quite confident we can get all our dependencies ready", and yet with --enable=frozen_string_literal which has exactly the same effect, you "can't" ??? It seems to me the ideal course here is you test your 700 gems with --enable=frozen_string_literal then submit any fixes to the gems' maintainers, and possiblity convince them to switch to frozen_string_literal: false style, and then you can run your app with --enable=frozen_string_literal and everyone's happy. Doesn't seem to be any need to change the default of the ruby interpreter for that.

> Code that is written for frozen_string_literal: true can generally run with frozen_string_literal: false, but the opposite is not true.

Indeed, and that's why changing the current default of false to true runs the risk of incompatibility. If you consider that a problem that prevents you from using --enable=frozen_string_literal, why would it not be a problem when changing the interpreter default?

I mean, of course what's missing in the above is the "chilled string" of the proposal, which provides a feasible migration path. But having these chilled strings is orthogonal to changing the default. You could run your app with e.g. --enable=CHILLED_string_literal and achieve your goal without having to change the default.

#25 - 01/24/2024 09:58 PM - byroot (Jean Boussier)

> and possiblity convince them to switch to frozen_string_literal: false style

You are missing the social aspect of it.

As of today running ruby with --enable=frozen_string_literal is a little known feature that basically no-one is doing. It's hard, if not impossible to convince maintainers to take in such changes. Since it's not the default, it basically comes down to personal preference (cf standardrb).

And any new gem created from today will likely not test with --enable=frozen_string_literal, so when they get added on our apps, they won't work. So it will be a never ending task.

Now if the Ruby project states that in the future the default will flip, even if it's a long time from now, it becomes easy to convince maintainers.

I'm not asking to change the default because I'm too lazy to fix some gems. I'm more than happy to help gems upgrade, I fixed over a hundred gems to be compatible with Ruby 3.0..., and will likely fix many more to be compatible with frozen string literals whenever Ruby decide to make the switch.

This is just me recognizing I can't realistically make this change alone in my corner of the ecosystem, no matter how much effort I put in it.

But also, I'm not getting this out of thin air, unless I misunderstood Matz, he stated he wishes to enable frozen string literals by default at some point, and that the only reason it wasn't done yet is the lack of a proper migration plan. So I'm not the one to suggest to flip the default in the first place, I'm merely proposing a migration plan.

#26 - 01/24/2024 10:25 PM - jeremyevans0 (Jeremy Evans)

byroot (Jean Boussier) wrote in #note-25:

> As of today running ruby with --enable=frozen_string_literal is a little known feature that basically no-one is doing. It's hard, if not impossible to convince maintainers to take in such changes.

As a counterpoint, I've been testing Sequel, Roda, and Rodauth since the release of Ruby 2.3 with --enable-frozen-string-literal. When starting out, many of the dependencies (direct and transitive) broke internally, and I had to submit pull requests to fix them. It took a few years, but I eventually got all related pull requests merged upstream (or alternative fixes implemented), and for years the tests have been clean with --enable-frozen-string-literal.

In most cases, it was easy to convince maintainers to fix any breakage encountered when using --enable-frozen-string-literal, and in many cases, maintainers took it upon themselves to fix other issues I didn't encounter.

> And any new gem created from today will likely not test with --enable=frozen_string_literal, so when they get added on our apps, they won't work. So it will be a never ending task.

It's been years since I've had to submit a pull request upstream related to --enable-frozen-string-literal, so I don't think it necessarily has to be a never ending task.

However, I'm not dealing with 700 transitive dependencies, maybe not even 70. And when you are using --enable-frozen-string-literal, you need to have everything fixed for things to work. So maybe at Shopify scale, the problem really is intractable.

> Now if the Ruby project states that in the future the default will flip, even if it's a long time from now, it becomes easy to convince maintainers.

Certainly it becomes easier, but my experience is that it is already easy. I would expect you are more likely to run into a dependency that isn't maintained, versus a dependency that is maintained but the maintainer is against fixing --enable-frozen-string-literal issues.

> But also, I'm not getting this out of thin air, unless I misunderstood Matz, he stated he wishes to enable frozen string literals by default at some point, and that the only reason it wasn't done yet is the lack of a proper migration plan. So I'm not the one to suggest to flip the default in the first place, I'm merely proposing a migration plan.

I am in favor of switching to frozen static string literals by default with the migration plan proposed.

#27 - 01/24/2024 10:32 PM - byroot (Jean Boussier)

> when you are using --enable-frozen-string-literal, you need to have everything fixed for things to work. So maybe at Shopify scale, the problem really is intractable.

Yes you are right, I should have said, it's hard if not impossible to convince **all** the maintainers of my transitive dependencies.

#28 - 01/25/2024 11:30 AM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote in #note-9:

> @zverok (Victor Shepelev) Ah, that's exactly why I was against the introduction of frozen_string_literal, because people misunderstand it like you. This feature must not be mixed up with "immutability". I can understand if all String objects were frozen by default, but freezing only string "literals" makes no sense at all (except in terms of performance). Consider, just adding .upcase or .b or something to a String literal makes it mutable. I don't find "a cultural thing" in such a fragile thing.

It is not full immutability of all Strings, true.
But it is full immutability of all (static) String literals. And that is valuable. With the new default it becomes impossible to accidentally mutate a String literal, which is a nice error category to remove.
It also saves some memory because the same String literal in different places is the same String object.
If not frozen, the bytes can be shared but not the String object itself.

I would say 5% on lobsters is a huge gain.
Very few optimizations can give that much (and as an extra it's fairly simple and well understood semantically).

My impression is most Rubyists are aware that basically all new code should use # frozen_string_literal: true semantics.
The default of false is basically deprecated in practice and has almost no value, except compatibility for existing code.

As an example very very few files use # frozen_string_literal: false.
And those that do typically only do it because they have no been migrated to # frozen_string_literal: true and to not break under --enable-frozen-string-literal.

#29 - 01/25/2024 12:18 PM - byroot (Jean Boussier)

> I would say 5% on lobsters is a huge gain.

Still trying to get hold on our benchmarking server...

But just to note, lobsters itself doesn't use frozen_string_literal: true, and it's probably the case of at least some of its dependencies too.

When I have some extra time I'd also like to make it --enable=frozen_string_liteal compatible to see if a couple more % could be squeezed out of it.

#30 - 01/25/2024 01:53 PM - Dan0042 (Daniel DeLorme)

byroot (Jean Boussier) wrote in #note-25:

> You are missing the social aspect of it.

Thank you for the explanation. I'm not sure I fully agree but it makes a lot more sense now.

Eregon (Benoit Daloze) wrote in #note-28:

> I would say 5% on lobsters is a huge gain.

Let's not cherry-pick here. 5% is the best case. The worst case is -6% on setivar_object. And something more relevant to a rails application: erubi has -3%. The average of all tests is roughly 0%. So it's not like the performance benefit is clearly compelling. At least based on these preliminary benchmarks.

#31 - 01/25/2024 02:33 PM - byroot (Jean Boussier)

> The worst case is -6% on setivar_object

This is a fluke caused by random slowdown on my development machine. Here's the benchmark source:
https://github.com/Shopify/yjit-bench/blob/3774b4bc320519f8b560eb23bdea48f549cf8b30/benchmarks/setivar_object.rb

There is absolutely nothing in there influenced by frozen strings. I'm somewhat confident on the 5% figure for lobsters because I ran it alone about 10 times and always got 5%. But running the full suite on my machine takes way too long to do that for all benchmarks.

I shouldn't have ran the micro-benchmarks anyway, only the headline benchmarks make sense.

What is certain is that turning on frozen_string_literal cannot possibly have a negative performance impact. Only null or positive. So don't lean to much on these preliminary results.

#32 - 01/25/2024 03:32 PM - Dan0042 (Daniel DeLorme)

byroot (Jean Boussier) wrote in #note-31:

> I shouldn't have ran the micro-benchmarks anyway, only the headline benchmarks make sense.

I'll freely admit I have no idea which benchmarks are micro and which are headline. What about erubi and erubi-rails?

> What is certain is that turning on frozen_string_literal cannot possibly have a negative performance impact.

I wouldn't affirm "cannot possibly", but I tend to agree; though deduplication has an overhead, I'd be surprised if it was measurable. So the erubi -3% result stands out. Was that a benchmark glitch?

#33 - 01/25/2024 03:43 PM - byroot (Jean Boussier)

> I have no idea which benchmarks are micro and which are headline. What about erubi and erubi-rails?

Each yjit-bench suite has a category: https://github.com/Shopify/yjit-bench/blob/3774b4bc320519f8b560eb23bdea48f549cf8b30/benchmarks.yml#L21

The more relevant ones are "headline", they generally are more sizeable and varied, so mor erepresentative of actually production workloads.

erubi and erubi-rails are both in the headline category. But frozen_string_literal doesn't matter one bit for eruby because eruby already compile into code that use frozen literals regardless.

> though deduplication has an overhead

frozen_string_literal: true doesn't incur any overhead even if it means you sometimes need to dup.

With frozen_string_literal: false, "foo" is strictly equivalent to "foo".dup except that the dup is done as part of putstring instead of a second instruction. But we could even eliminate that if we wanted by compiling "literal".dup into putstring. It is very very unlikely to make any measurable difference though.

So yes, if we want to be extremely pedantic it's possible to generate a micro-benchmark that would suffer from frozen_string_literal: true, but in reality the impact can't reasonably be negative.

#34 - 01/25/2024 04:49 PM - byroot (Jean Boussier)

Alright, I finally got hold of the benchmarking server. For the record it's an AWS c5n.metal, with various tuning like disabling frequency scaling etc to hopefully get more stable results.

- yjit-bench revision: https://github.com/Shopify/yjit-bench/commit/95e1a3caddc7281fbaf5bd0f20b197561453993f
- ruby revision: https://github.com/Shopify/ruby/commit/bb0cee8daba4b70cedb40b36af05d796956475d6

Both rubyes are ran with YJIT enabled. mutable has MUTABLE_STRINGS=1 which makes the frozen_string_literal: true comment become essentially a noop.

I ran headline benchmark twice for good measures, most of them seem to be consistent with 1% between the two runs, except for ruby-lsp which has widely inconsistent results (+/-12%??), and hexapdf too to some extent (+/-4%).

```
mutable: ruby 3.4.0dev (2024-01-24T08:24:16Z mutable-strings bb0cee8dab) +YJIT dev [x86_64-linux]
frozen: ruby 3.4.0dev (2024-01-24T08:24:16Z mutable-strings bb0cee8dab) +YJIT dev [x86_64-linux]

--------------  ------------  ----------  -----------  ----------  --------------  --------------
bench           mutable (ms)  stddev (%)  frozen (ms)  stddev (%)  frozen 1st itr  mutable/frozen
activerecord    72.8          1.2         70.8         1.2         1.03            1.03
chunky-png      1666.3        0.1         1672.7       0.2         1.00            1.00
erubi-rails     2330.1        0.3         2312.2       0.2         1.01            1.01
hexapdf         3960.1        8.9         3614.7       7.3         1.04            1.10
liquid-c        107.1         1.3         101.3        1.4         0.95            1.06
liquid-compile  100.0         2.8         98.5         3.1         1.00            1.02
liquid-render   153.0         1.1         137.8        1.1         0.97            1.11
lobsters        1346.6        8.5         1239.2       7.6         1.00            1.09
mail            202.8         0.8         198.9        1.0         1.02            1.02
psych-load      3723.2        0.1         3661.3       0.1         1.01            1.02
railsbench      2669.8        0.1         2519.6       0.2         1.02            1.06
rubocop         278.6         5.9         272.5        6.1         1.00            1.02
ruby-lsp        217.5         8.7         250.7        10.4        0.98            0.87
sequel          111.8         0.7         111.8        0.8         1.01            1.00
--------------  ------------  ----------  -----------  ----------  --------------  --------------
Legend:
- frozen 1st itr: ratio of mutable/frozen time for the first benchmarking iteration.
- mutable/frozen: ratio of mutable/frozen time. Higher is better for frozen. Above 1 represents a speedup.
```

```
mutable: ruby 3.4.0dev (2024-01-24T08:24:16Z mutable-strings bb0cee8dab) +YJIT dev [x86_64-linux]
frozen: ruby 3.4.0dev (2024-01-24T08:24:16Z mutable-strings bb0cee8dab) +YJIT dev [x86_64-linux]

--------------  ------------  ----------  -----------  ----------  --------------  --------------
bench           mutable (ms)  stddev (%)  frozen (ms)  stddev (%)  frozen 1st itr  mutable/frozen
activerecord    72.8          1.2         70.8         1.3         1.00            1.03
chunky-png      1664.1        0.2         1672.5       0.2         1.01            0.99
erubi-rails     2351.9        0.2         2311.4       0.2         1.01            1.02
hexapdf         3759.9        7.0         3696.5       5.8         1.04            1.02
liquid-c        108.0         1.1         101.3        1.4         0.97            1.07
liquid-compile  100.5         3.1         98.5         3.1         0.99            1.02
liquid-render   152.8         1.0         138.0        1.1         0.97            1.11
lobsters        1339.0        9.8         1244.2       7.9         1.00            1.08
mail            202.6         0.7         198.9        1.0         1.00            1.02
psych-load      3627.0        0.1         3659.6       0.0         0.99            0.99
railsbench      2670.7        0.1         2560.3       1.7         1.02            1.04
rubocop         279.5         6.3         271.4        6.1         1.00            1.03
ruby-lsp        274.2         9.5         245.0        9.8         0.98            1.12
sequel          111.9         0.7         112.0        1.3         1.01            1.00
--------------  ------------  ----------  -----------  ----------  --------------  --------------
```

```
Legend:
- frozen 1st itr: ratio of mutable/frozen time for the first benchmarking iteration.
- mutable/frozen: ratio of mutable/frozen time. Higher is better for frozen. Above 1 represents a speedup.
```

Overall the performance benefit seem much more important than on my initial benchmark, likely in part because YJIT is enabled and also likely in part because of the different architecture (X86_64 vs ARM64).

#35 - 02/14/2024 07:30 AM - matz (Yukihiro Matsumoto)

I agree with the proposal. It seems a well-thought process to migrate. The performance improvement was not as great as I had hoped for. But since I feel that the style of individually freezing strings when setting them to constants is not beautiful, and since I feel that magic comment is not a good style. I feel that making string literals frozen is the right direction to go in the long run.

Matz.

#36 - 02/14/2024 09:08 AM - byroot (Jean Boussier)

Thank you Matz.

In that case I'll work with @etienne into getting the proof of concept into a mergeable feature over the next few weeks.

#37 - 02/14/2024 02:48 PM - Dan0042 (Daniel DeLorme)

Question: what is the effect of "chilled string" on #frozen?
Does chilled_string.frozen? return true?
Personally I think it should, as I remarked in #16153#note-11

#38 - 02/14/2024 04:28 PM - byroot (Jean Boussier)

> Question: what is the effect of "chilled string" on #frozen?
> Does chilled_string.frozen? return true?

That's the current idea yes.

```
"chilled string".frozen? # => true
+"chilled string" # returns a new, mutable string (without chilled flag)
-"chilled string" # returns a different, frozen string
"chilled string".freeze # the chilled string become frozen for real.
"chilled string" << "foo" # emit a warning and clear the "chilled" status.
```

This way as you point out in the other issues, the common dup if frozen? idiom works as expected.

#39 - 03/19/2024 08:27 AM - etienne (Étienne Barrié)

- Status changed from Open to Closed

Applied in changeset git|12be40ae6be78ac41e8e3f3c313cc6f63e7fa6c4.

Implement chilled strings

[Feature #20205]

As a path toward enabling frozen string literals by default in the future,
this commit introduce "chilled strings". From a user perspective chilled
strings pretend to be frozen, but on the first attempt to mutate them,
they lose their frozen status and emit a warning rather than to raise a
FrozenError.

Implementation wise, rb_compile_option_struct.frozen_string_literal is
no longer a boolean but a tri-state of enabled/disabled/unset.

When code is compiled with frozen string literals neither explictly enabled
or disabled, string literals are compiled with a new putchilledstring
instruction. This instruction is identical to putstring except it marks
the String with the STR_CHILLED (FL_USER3) and FL_FREEZE flags.

Chilled strings have the FL_FREEZE flag as to minimize the need to check
for chilled strings across the codebase, and to improve compatibility with
C extensions.

Notes:

- String#freeze: clears the chilled flag.
- String#-@: acts as if the string was mutable.
- String#+@: acts as if the string was mutable.
- String#clone: copies the chilled flag.

Co-authored-by: Jean Boussier byroot@ruby-lang.org

#40 - 03/19/2024 05:17 PM - Dan0042 (Daniel DeLorme)

Thank you for this great feature!

Would it be possible to have a ruby API for chilled strings? Something like str.chill or String.chill(str) ...

> - String#+@: acts as if the string was mutable.

Nitpick: actually, String#+@ creates a dup as if the string was immutable.

#41 - 03/19/2024 07:00 PM - byroot (Jean Boussier)

> Would it be possible to have a ruby API for chilled strings? Something like str.chill or String.chill(str)

It's technically very easy to implement, yes. Would need to be a separate feature request though.

Also one drawback is that right now chilled string are an internal concept that we'll be able to get rid of and cleanup in the future. If we expose it to users, we'll have to keep it forever. So it's debatable whether the benefit outweigh the maintenance burden long term.

#42 - 03/19/2024 08:37 PM - Dan0042 (Daniel DeLorme)

> it's debatable whether the benefit outweigh the maintenance burden long term.

Yeah, good point. And if truly necessary we could get a chilled string with eval(str.inspect) anyway.

#43 - 05/06/2024 06:00 PM - headius (Charles Nutter)

I am a bit late to the party but nobody seems to have raised a concern I have.

If a chilled string appears to be frozen? then a consumer may proceed to use the string expecting it to remain frozen, such as for a cache key. If that string can later become unfrozen and be modified, warning or not, they may now have a broken cache with an unexpected mutable key.

I don't know of a specific case for this, but the fact that chilled strings masquerade as frozen when they are not really frozen seems like a major issue to me.

#44 - 05/06/2024 06:58 PM - byroot (Jean Boussier)

While your concern is absolutely valid, I don't think it's much of a problem in practice. There is pretty much an infinite amount of code out there, so I don't think there is any way really to discount it, but after having migrated a gigantic codebase to 3.4-dev no such issue appeared.

And generally speaking, code that want to hold on hold on a frozen string like you mention tend to use either str.dup.freeze, or str.freeze or -str. str.frozen? ? str : str.dup.freeze isn't a very common pattern.

So yes it could happen, but you'd really need many stars to align for that.

#45 - 05/20/2024 01:37 AM - byroot (Jean Boussier)

I thought about this more at Kaigi, maybe avoiding the false positive on the str.dup if str.frozen? pattern isn't worth the possible confusion.

I'll experiment with chilled strings frozen? method returning false.

#46 - 05/22/2024 12:51 AM - byroot (Jean Boussier)

- Related to Feature #11473: Immutable String literal in Ruby 3 added

#47 - 05/23/2024 05:53 PM - Dan0042 (Daniel DeLorme)

@byroot (Jean Boussier) How is that experiment going? I'm all for experimenting, but just as in #15554 I believe we should reduce false positives to a minimum.

Also I spent some time on this, but I'm having a hard time coming up with a non-contrived example case where returning false for #frozen? is

beneficial.

#48 - 05/23/2024 06:00 PM - byroot (Jean Boussier)

I'm still on my way back from Kaigi, so I haven't started working on this. But I had a quick chat with Matz, and it wasn't clear to him that we went with frozen? -> true, and he was clear he expects frozen? -> false.

Also based on the reception I saw of the release notes in various places, it seems to have created quite a bit of confusion.

So yes, this will introduce some false positive, which is unfortunate, but I'll definitely change the behavior soon.

#49 - 05/24/2024 12:27 PM - Dan0042 (Daniel DeLorme)

This is quite unfortunate, as there is not a single useful case for frozen? -> false
Oh well :-/

#50 - 05/24/2024 12:42 PM - Eregon (Benoit Daloze)

> avoiding the false positive on the str.dup if str.frozen? pattern

+str seems a good replacement for that pattern.
Besides, I would think it's pretty rare that it's OK to use that pattern, because it mutates a String that is not "owned" if it happens to not be frozen (e.g. it's typically not OK if it's an argument, and if intended then no need to check if frozen).

IOW, returning false for frozen? for chilled strings seems safer and I think will cause very few false positive warnings (in comparison to true positive chilled strings warnings).

#51 - 05/24/2024 12:46 PM - Eregon (Benoit Daloze)

@Dan0042 @headius (Charles Nutter) mentioned a few above.
It also seems pretty bad that an object could frozen? => true and then become unfrozen, I think that alone could cause very tricky and serious bugs (e.g. it would break custom hash tables).

#52 - 05/24/2024 01:19 PM - Eregon (Benoit Daloze)

To give a concrete example, Hash would be broken if it calls frozen? for string keys. That's the case on Rubinius:
https://github.com/rubinius/rubinius/blob/84368419a49767ef9549a5778812e5f54b6c6223/core/hash.rb#L54-L56
So the pattern of "safe frozen copy" str = str.dup.freeze unless str.frozen? would be broken if frozen? would return true for chilled strings (str would still be mutable).

#53 - 05/24/2024 09:04 PM - Dan0042 (Daniel DeLorme)

I was going to say this is MRI, not Rubinius, but it turns out chilled strings actually have a bug in MRI when used as Hash keys

```
k = "key"
h = {}
h[k] = 42
k << "!"  #warning: literal string will be frozen in the future
p h       #{"key!"=>42}
```

The hash key is definitely supposed to be frozen here, not just chilled.

#54 - 05/30/2024 12:43 PM - byroot (Jean Boussier)

- Related to Feature #20390: Issue with StringIO and chilled strings added

#55 - 06/03/2024 10:30 PM - hartator (Julien Khaleghy)

I think making # frozen_string_literal: true the default is a bad idea.

If the main point is to make Ruby faster, IRL benchmarks so far have shown the reverse. byroot's own initial benchmarks show regular strings being faster or as fast as frozen strings in 62.2% of libs that has been tested. I would be interested to see if *removing* # frozen_string_literal: true will actually make some libs faster (edit: super tiny benchmarks on pry repo running its tests on 3 runs after 1 warmup and can be just noise, but 35.052s, 34.469s, 34.948s *with* # frozen_string_literal: true vs. 34.525s, 34.577s, 34.714s *without* # frozen_string_literal: true) .

If the main point is to avoid some kind of bugs, the reverse is also true. # frozen_string_literal: true can be a misdirection and introduces its own kind of bugs. IRL this has introduces stubble bugs in our SerpApi codebase, consider this file:

```
# frozen_string_literal: true

class Converter
  LANGUADE_CODES = {
```

```
    en: "English",
    es: "Spanish",
    jp: "Japanese"
  }
  def lower_case_version
    LANGUADE_CODES.transform_values!(&:downcase)
  end
end
```

The initial quick assessment is to feel safe that LANGUADE_CODES can't be modified and lower_case_version would have one of these behaviors:

- Raise an error
- Log a warning
- Code works, but LANGUADE_CODES is unmodified.
- Code works, but LANGUADE_CODES is modified for only this object instance.

However, none of the above is true.

Surprisingly for both junior and senior engineers, a #lower_case_version method call will silently modify LANGUADE_CODES for all Ruby threads:

```
[7] pry(main)> Converter::LANGUADE_CODES
=> {:en=>"english", :es=>"spanish", :jp=>"japanese"}
```

I am afraid we are making a more complex and inflexible Ruby (I do agree that string << "a story," appending style is lovely and super useful) for unfortunately no tangible gain in code explicitness and correctness, and now even have introducing surprising new behaviors. I think it's easier to just assume all String objects are mutable and just allows flexible coding.

#56 - 10/09/2024 03:18 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Closed to Open

- Assignee set to matz (Yukihiro Matsumoto)

I and @yhonda try to fix this warnings at httpclient.

https://github.com/nahi/httpclient/pull/462

I surprisedly faced stdlib have this warning like:

```
ruby-dev/lib/ruby/3.4.0+0/open-uri.rb:455: warning: literal string will be frozen in the future
ruby-dev/lib/ruby/3.4.0+0/logger/log_device.rb:45: warning: literal string will be frozen in the future
```

But above warnings caused by httpclient usage like:

```
def set_body_encoding
  if type = self.content_type
    OpenURI::Meta.init(o = '')
    o.meta_add_field('content-type', type)
    @body_encoding = o.encoding
  end
end
```

It's hard to found above code from open-uri warning.

#57 - 10/09/2024 04:12 AM - mame (Yusuke Endoh)

I did quick investigation on the warnings that @hsbt (Hiroshi SHIBATA) said.

```
ruby-dev/lib/ruby/3.4.0+0/open-uri.rb:455: warning: literal string will be frozen in the future
```

This warning is due to the fact that httpclient calls OpenURI::Meta.init(o = '') and OpenURI::Meta.init destructively modifies the argument by Object#extend.

```
ruby-dev/lib/ruby/3.4.0+0/logger/log_device.rb:45: warning: literal string will be frozen in the future
```

This is probably a warning in StringIO#write. Passing a frozen string literal like StringIO.new("foo") is not warned, but warned after StringIO#write.

```
$ ruby -rstringio -w -e 'StringIO.new("")'

$ ruby -rstringio -w -e 'StringIO.new("").write("foo")'
-e:1: warning: literal string will be frozen in the future
```

Note that we can't issue a warning at StringIO.new("") because it could be a false positive if you use the StringIO read-only.

#58 - 10/09/2024 05:37 AM - mame (Yusuke Endoh)

--debug-frozen-string-literal does not work well for StringIO.new("") case.

```
$ ruby --enable-frozen-string-literal --debug-frozen-string-literal -rstringio -w -e 'io = StringIO.new("")
io.write("foo")'
-e:2:in 'StringIO#write': not opened for writing (IOError)
        from -e:2:in '<main>'
```

The message failed to spot Line 1 where the string is allocated.

#59 - 10/09/2024 12:54 PM - byroot (Jean Boussier)

> --debug-frozen-string-literal does not work well for StringIO.new("") case.

Yes, for code that have a String#frozen? conditional, the source of the issue can be harder to find, that is why the initial implementation had String#frozen? return true.

@hsbt (Hiroshi SHIBATA) since you re-opened, what would you consider a condition for closing again?

#60 - 10/11/2024 05:10 AM - mame (Yusuke Endoh)

byroot (Jean Boussier) wrote in #note-59:

> Yes, for code that have a String#frozen? conditional, the source of the issue can be harder to find, that is why the initial implementation had String#frozen? return true.

The problem is that the warning does not point where you are doing StringIO.new(""). Even if #frozen? method returns true on a chilled string, I don't think that problem will be solved at all.

In this case, I think --debug-frozen-string-literal could work by stopping the #frozen? check and making StringIO#write attempt to actually modify the string unless read-only mode is explicitly specified.

But what can I say, do we have to go so far to change it? I totally agree with hartator #note-55.

#61 - 10/11/2024 10:12 AM - byroot (Jean Boussier)

> The problem is that the warning does not point where you are doing StringIO.new("")

We can improve that by displaying where the String was allocated when you are running with --debug-frozen-string-literal. I'll work on that Monday with @etienne.

I get that in a few cases it can be hard to track down, but having done the work to get Shopify monolith run with --enable-frozen-string-literal, hence having fixed issue in a bunch of our dependencies, it really wasn't that much work.

And in the rare case of old gems like httpclient that are no longer actively developed, just slapping # frozen_string_literal: false is a simple way out: https://github.com/Shopify/httpclient/commit/19790ac5bef02613b368ad7f3443767c8d481ec4

If any gem is causing you trouble, let me know and I'll happily fix it for you.

#62 - 10/14/2024 09:54 AM - byroot (Jean Boussier)

We just submitted a PR with @etienne: https://github.com/ruby/ruby/pull/11893

Now the default warning is: warning: literal string will be frozen in the future (run with --debug-frozen-string-literal for more information)

And if you run with --debug-frozen-string-literal, the warning will be:

```
test.rb:3: warning: literal string will be frozen in the future
test.rb:1: the string was created here
```

Which should make it much easier to find the source of the issue in cases like: https://github.com/nahi/httpclient/pull/462

#63 - 10/21/2024 10:05 AM - pdfrod (Pedro Rodrigues)

hartator (Julien Khaleghy) wrote in #note-55:

> If the main point is to avoid some kind of bugs, the reverse is also true. # frozen_string_literal: true can be a misdirection and introduces its own kind of bugs. IRL this has introduces stubble bugs in our SerpApi codebase, consider this file:

```
    # frozen_string_literal: true

    class Converter
      LANGUADE_CODES = {
        en: "English",
        es: "Spanish",
        jp: "Japanese"
      }
      def lower_case_version
        LANGUADE_CODES.transform_values!(&:downcase)
      end
    end
```

I don't think this example can be used to argue for or against frozen string literals, because there's no attempt to mutate strings here at all (downcase never modifies the original string; it always returns a copy). The real problem in this code is that it's using transform_values!, which mutates the hash. The code would have issues even if the values were integers. For example:

```
class Converter
  LANGUADE_CODES = {
    en: 1,
    es: 2,
    jp: 3
  }
  def lower_case_version
    LANGUADE_CODES.transform_values! { _1 * 100 }
  end
end

irb> Converter::LANGUADE_CODES
=> {:en=>1, :es=>2, :jp=>3}

irb> Converter.new.lower_case_version
=> {:en=>100, :es=>200, :jp=>300}

irb> Converter::LANGUADE_CODES
=> {:en=>100, :es=>200, :jp=>300}  # Oops, Converter::LANGUADE_CODES got mutated
```

#64 - 10/21/2024 10:34 AM - byroot (Jean Boussier)

- Status changed from Open to Closed

I merged https://github.com/ruby/ruby/pull/11893 which I believe addresses https://bugs.ruby-lang.org/issues/20205#note-56

#65 - 12/17/2024 04:14 AM - mame (Yusuke Endoh)

I checked the usage of # frozen-string-literal: true among the public gems. (@ko1 (Koichi Sasada) gave me this idea)

```
Number of gems that have `# frozen-string-literal: true` in all .rb files

  among all public gems: 14254 / 175170 (8.14%)
  per-file basis: 445132 / 3051922 (14.59%)

  among public gems with the latest version released in 2015-present: 14208 / 101904 (13.94%)
  per-file basis: 443460 / 1952225 (22.72%)

  among public gems with the latest version released in 2020-present: 11974 / 41205 (29.06%)
  per-file basis: 389559 / 1136445 (34.28%)

  among public gems with the latest version released in 2022-present: 9121 / 26721 (34.13%)
  per-file basis: 329742 / 848061 (38.88%)
```

I used these scripts: https://gist.github.com/mame/d4a9be41acf3d25cc2667a6959c4d37d

Less than 10% of all gems are fully frozen-string-literal: true compliant. Even among active gems released in the last three years, only 34% are fully frozen-string-literal: true.

Incidentally, when ko1 asked matz about his prediction before showing these results, he replied, “About 90% of files are using frozen-string-literal: true". However, the actual rate was about 39% even on a per-file basis.

This suggests that the decision was based on a major misunderstanding of the current situation. @matz (Yukihiro Matsumoto), are you sure you don't want to rethink this?

#66 - 12/17/2024 08:45 AM - byroot (Jean Boussier)

Note that you don't necessarily need to set # frozen_string_literal: true to be compatible.

Lots of code is compatible without the comment just by the virtue of not mutating literals, and some other code is compatible because it sets #frozen_string_literal: false.

#67 - 12/17/2024 01:17 PM - kddnewton (Kevin Newton)

I think to be more accurate your script would probably want to check for string literals that will or will not be frozen.

#68 - 12/19/2024 01:53 AM - ko1 (Koichi Sasada)

> some other code is compatible because it sets #frozen_string_literal: false.

FYI

only a few. survey which pragma is used on my collected ruby scripts from rubygems:

```
[[:FILES, 2945839],
 [["frozen_string_literal", "true"], 372224],
 [["frozen_string_literal", "false"], 4108],
```

all usages (it contains not pragma, just a comment):
https://gist.github.com/ko1/801a55069673aa3b45bf5c9c9be26e42

#69 - 12/19/2024 02:39 AM - hsbt (Hiroshi SHIBATA)

- Status changed from Closed to Assigned

I re-opened this ticket for further discussion with Matz.

#70 - 12/19/2024 04:48 AM - austin (Austin Ziegler)

byroot (Jean Boussier) wrote in #note-66:

> Note that you don't necessarily need to set # frozen_string_literal: true to be compatible.
>
> Lots of code is compatible without the comment just by the virtue of not mutating literals, and some other code is compatible because it sets #frozen_string_literal: false.

It is worth noting that standardrb (the variant of Rubocop that I use) turns off # frozen_string_literal: true checking (in part because of hints that frozen strings would never be enforced, which RuboCop has on by default. I believe that I have deleted a whole bunch of cases where I used it in some of my gems (mime-types, diff-lcs), but I try to be very careful about allocations &c in all of my gems.

#71 - 12/19/2024 07:50 AM - byroot (Jean Boussier)

> be very careful about allocations &c in all of my gems.

If you wish not to include the comment, but remain compatible a good way is to add --enable-frozen-string-literal in your CI matrix.

#72 - 12/19/2024 08:09 AM - byroot (Jean Boussier)

To put the number in perspective, I adapted @mame's script to count the same thing but for a specific application and all its dependencies:
https://gist.github.com/byroot/87235e9c2553aacd36c31b1f16e81b84

For https://github.com/lobsters/lobsters

```
$ ruby /tmp/frozen.rb . $(bundle show --paths)
total files: 8037
files with comment: 5424
rate 67.5%
```

And yet, lobsters is compatible:
https://github.com/lobsters/lobsters/blob/96cf0b32ee81bb1bd7e6e9cb8c7030631145f902/.github/workflows/check.yml#L46

#73 - 12/19/2024 08:25 AM - byroot (Jean Boussier)

Also it probably doesn't make a huge difference, but your regexp is a bit too strict:

```
/^# frozen[-_]string[-_]literal: true$/
```

Should be something like:

```
/^#\s*frozen[-_]string[-_]literal:\s*true\s*$/
```

#74 - 12/19/2024 03:22 PM - Dan0042 (Daniel DeLorme)

I watched this Ruby Core Developers Q&A video recently, and at 20:30 on the topic of frozen string literals by default everyone was complaining about the annoyance of having to put the magic comment at the top of every file. But if that's the main justification, there's a very easy fix that doesn't involve changing the global default: just set a default frozen_string_literal **per directory**, so that all files loaded afterward within a gem or app directory will use that default.

And even if the global default is changed to frozen_string_literal: true (which I am in favor of), having the ability to set a per-gem default would make it MUCH easier to add frozen_string_literals: false compatibility to older gems.

#75 - 07/10/2025 08:51 AM - matz (Yukihiro Matsumoto)

Probably we would have Ruby4.0 in 2025, but I am not going to make frozen-string-literal default this year.

Matz.

#76 - 08/10/2025 08:57 PM - janosch-x (Janosch Müller)

byroot (Jean Boussier) wrote in #note-73:

> it probably doesn't make a huge difference, but your regexp is a bit too strict

It probably makes even less of a difference, but magic comments are also indentable, case-insensitive, and can be grouped in one line, so *maybe*

```
/^\s*#\s*(-\*-(?:.*;)*\s*)?frozen[-_]string[-_]literal:\s*true(?(1)\s*(?:;.*)*-\*-)\s*$/i
```

#77 - 10/28/2025 06:44 PM - Eregon (Benoit Daloze)

matz (Yukihiro Matsumoto) wrote in #note-75:

> Probably we would have Ruby4.0 in 2025, but I am not going to make frozen-string-literal default this year.

That's unfortunate, but OTOH it makes sense because first we need to make the chilled string deprecation warning show up regardless of verbosity level (R1, as explained in the issue description).
@matz (Yukihiro Matsumoto) Could you decide for what release R1 will be done?
I think that could make a lot of sense for Ruby 4.0: to warn regardless of verbosity level when mutating a chilled string, while preserving the semantics as-is.
And then in some Ruby 4.X or 5.0 we'd finally switch the default.

FWIW, Jean wrote a nice blog post about this: https://byroot.github.io/ruby/performance/2025/10/28/string-literals.html
I feel sad when reading it, much effort has been spent to make Ruby nicer and have less boilerplate (the frozen_string_literal: true magic comment), and yet it's still has not happened and so far there isn't even a plan of when it would happen.
In hindsight, I find it clear that "foo".freeze was just a workaround and wouldn't scale (too ugly / too much boilerplate) and that frozen_string_literal: true is much better but still boilerplate for every file (and if missing that's a performance trap) and feels like a workaround.
Let's make Ruby beautiful and boilerplate-free again!

#78 - 10/28/2025 06:51 PM - Anonymous