Yesterday, we had a face-to-face developer meeting.
https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20131001Japan
Several committers attended.
matz didn't attend, though. (This means this issue is not concluded.)

We believe we found a better way to freeze static string literals for
less GC pressure.
“static string literal” is a string literal without dynamic expression.

Currently, f-suffix, "...f, is used to freeze a string literal to avoid
String object allocation.

There are several problems for f-suffix:

- The notation is ugly.
- Syntax error on Ruby 2.0.
  We cannot use the feature in version independent libraries.
  So, it is difficult to deploy.
- Need to modify for each string literal.
  This is cumbersome.

The new way we found is a file-scope directive as follows

freeze_string: true

The above comment at top of a file changes semantics of
static string literals in the file.
The static string literals will be frozen and always returns same object.
(The semantics of dynamic string literals is not changed.)

This way has following benefits:

- No ugly f-suffix.
- No syntax error on older Ruby.
- We need only a line for each file.

We can write version independent library using frozen static string literals as follows.

- Use the directive at top of the file: # freeze_string: true
  Older Ruby ignore this as a comment.
- Use "...".dup for strings to be modified.
  Older Ruby has small disadvantage: useless dup is called.

Note that the directive effects all static string literals regardless of
single quotes, double quotes, %q-string, %qq-string and here documents.
The reason that the directive is effective not only single quotes is
we want to use escape sequences such as \n in frozen string literals.

Also note that similar directive is already exist:

```ruby
% ruby -w -e '
def m
  end
```
The directive, `warn_indent: false`, disables "mismatched indentations" warning.

nobu implemented this feature in the meeting. Please attach the patch, nobu.

**Related issues:**
- Related to Ruby master - Feature #8977: String#frozen that takes advantage of...
- Related to Ruby master - Feature #8579: Frozen string syntax
- Related to Ruby master - Feature #11473: Immutable String literal in Ruby 3
- Has duplicate Ruby master - Feature #9278: Magic comment "immutable: string"...

**Closed**

**06/29/2013**

**Associated revisions**

Revision 859337b1 - 09/27/2015 06:44 AM - nobu (Nobuyoshi Nakada)
```
frozen-string-literal pragma
```
- compile.c (iseq_compile_each): override compile option by option given by pragma.
- iseq.c (rb_iseq_make_compile_option): extract a function to overwrite rb_compile_option_t.
- parse.y (parser_set_compile_option_flag): introduce pragma to override compile options.
- parse.y (magic_comments): new pragma "frozen-string-literal".
  [Feature #8976]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@51953 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 51953 - 09/27/2015 06:44 AM - nobu (Nobuyoshi Nakada)
```
frozen-string-literal pragma
```
- compile.c (iseq_compile_each): override compile option by option given by pragma.
- iseq.c (rb_iseq_make_compile_option): extract a function to overwrite rb_compile_option_t.
- parse.y (parser_set_compile_option_flag): introduce pragma to override compile options.
- parse.y (magic_comments): new pragma "frozen-string-literal".
  [Feature #8976]

Revision 51953 - 09/27/2015 06:44 AM - nobu (Nobuyoshi Nakada)
```
frozen-string-literal pragma
```
- compile.c (iseq_compile_each): override compile option by option given by pragma.
- iseq.c (rb_iseq_make_compile_option): extract a function to overwrite rb_compile_option_t.
- parse.y (parser_set_compile_option_flag): introduce pragma to override compile options.
- parse.y (magic_comments): new pragma "frozen-string-literal".
  [Feature #8976]

Revision 51953 - 09/27/2015 06:44 AM - nobu (Nobuyoshi Nakada)
```
frozen-string-literal pragma
```
- compile.c (iseq_compile_each): override compile option by option given by pragma.
- iseq.c (rb_iseq_make_compile_option): extract a function to overwrite rb_compile_option_t.
- parse.y (parser_set_compile_option_flag): introduce pragma to override compile options.
- parse.y (magic_comments): new pragma "frozen-string-literal".
  [Feature #8976]
Revision 51953 - 09/27/2015 06:44 AM - nobu (Nobuyoshi Nakada)
fronzen-string-literal pragma

- compile.c (iseq_compile_each): override compile option by option given by pragma.
- iseq.c (rb_iseq_make_compile_option): extract a function to overwrite rb_compile_option_t.
- parse.y (parser_set_compile_option_flag): introduce pragma to override compile options.
- parse.y (magic_comments): new pragma "fronzen-string-literal". [Feature #8976]

Revision 51953 - 09/27/2015 06:44 AM - nobu (Nobuyoshi Nakada)
fronzen-string-literal pragma

- compile.c (iseq_compile_each): override compile option by option given by pragma.
- iseq.c (rb_iseq_make_compile_option): extract a function to overwrite rb_compile_option_t.
- parse.y (parser_set_compile_option_flag): introduce pragma to override compile options.
- parse.y (magic_comments): new pragma "fronzen-string-literal". [Feature #8976]

Revision 832c74f4 - 09/27/2015 06:47 AM - nobu (Nobuyoshi Nakada)
ruby.c: frozen-string-literal option

- ruby.c (process_options): add an option to enable/disable frozen-string-literal. [Feature #8976]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@51954 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 51954 - 09/27/2015 06:47 AM - nobu (Nobuyoshi Nakada)
ruby.c: frozen-string-literal option

- ruby.c (process_options): add an option to enable/disable frozen-string-literal. [Feature #8976]

Revision 51954 - 09/27/2015 06:47 AM - nobu (Nobuyoshi Nakada)
ruby.c: frozen-string-literal option

- ruby.c (process_options): add an option to enable/disable frozen-string-literal. [Feature #8976]

Revision 51954 - 09/27/2015 06:47 AM - nobu (Nobuyoshi Nakada)
ruby.c: frozen-string-literal option

- ruby.c (process_options): add an option to enable/disable frozen-string-literal. [Feature #8976]

Revision 51954 - 09/27/2015 06:47 AM - nobu (Nobuyoshi Nakada)
ruby.c: frozen-string-literal option

- ruby.c (process_options): add an option to enable/disable frozen-string-literal. [Feature #8976]

Revision 51954 - 09/27/2015 06:47 AM - nobu (Nobuyoshi Nakada)
ruby.c: frozen-string-literal option

- ruby.c (process_options): add an option to enable/disable frozen-string-literal. [Feature #8976]

Revision a76fdea9 - 10/21/2015 09:34 AM - nobu (Nobuyoshi Nakada)
parse.y: magic comment w/o indicators

- parse.y (parser_magic_comment): allow a sole magic comment without indicators, neither other non-space comments. [Feature #8976]
parse.y: magic comment w/o indicators

- parse.y (parser_magic_comment): allow a sole magic comment without indicators, neither other non-space comments. [Feature #8976]

compile.c (iseq_compile_each): Dynamic string literals should be frozen.

- compile.c (iseq_compile_each): Dynamic string literals should be frozen. [ruby-core:57574] [Feature #8976]

Revision 52208 - 10/21/2015 09:34 AM - nobu (Nobuyoshi Nakada)
parse.y: magic comment w/o indicators

- parse.y (parser_magic_comment): allow a sole magic comment without indicators, neither other non-space comments. [Feature #8976]

Revision 52208 - 10/21/2015 09:34 AM - nobu (Nobuyoshi Nakada)
parse.y: magic comment w/o indicators

- parse.y (parser_magic_comment): allow a sole magic comment without indicators, neither other non-space comments. [Feature #8976]

Revision 52208 - 10/21/2015 09:34 AM - nobu (Nobuyoshi Nakada)
parse.y: magic comment w/o indicators

- parse.y (parser_magic_comment): allow a sole magic comment without indicators, neither other non-space comments. [Feature #8976]

Revision 52208 - 10/21/2015 09:34 AM - nobu (Nobuyoshi Nakada)
parse.y: magic comment w/o indicators

- parse.y (parser_magic_comment): allow a sole magic comment without indicators, neither other non-space comments. [Feature #8976]

Revision 52208 - 10/21/2015 09:34 AM - nobu (Nobuyoshi Nakada)
parse.y: magic comment w/o indicators

- parse.y (parser_magic_comment): allow a sole magic comment without indicators, neither other non-space comments. [Feature #8976]

Revision 52501 - 11/09/2015 08:32 AM - shugo (Shugo Maeda)
compile.c (iseq_compile_each): Dynamic string literals should be frozen.

- compile.c (iseq_compile_each): Dynamic string literals should be frozen. [ruby-core:57574] [Feature #8976]

Revision 52501 - 11/09/2015 08:32 AM - shugo (Shugo Maeda)
compile.c (iseq_compile_each): Dynamic string literals should be frozen.

- compile.c (iseq_compile_each): Dynamic string literals should be frozen. [ruby-core:57574] [Feature #8976]

Revision 52501 - 11/09/2015 08:32 AM - shugo (Shugo Maeda)
compile.c (iseq_compile_each): Dynamic string literals should be frozen.

- compile.c (iseq_compile_each): Dynamic string literals should be frozen. [ruby-core:57574] [Feature #8976]

Revision 52501 - 11/09/2015 08:32 AM - shugo (Shugo Maeda)
compile.c (iseq_compile_each): Dynamic string literals should be frozen.

- compile.c (iseq_compile_each): Dynamic string literals should be frozen. [ruby-core:57574] [Feature #8976]
compile.c (iseq_compile_each): Dynamic string literals should be frozen.
[ruby-core:57574] [Feature #8976]

**History**

#1 - 10/02/2013 11:03 AM - nobu (Nobuyoshi Nakada)
https://github.com/nobu/ruby/tree/frozen_stringPragma

#2 - 10/02/2013 12:41 PM - sam.saffron (Sam Saffron)
coupled with this I strongly feel we need a more usable way of using the deduping elsewhere.

Currently string#freeze will only affect the current string. If we had a string#frozen we could have it return a deduped frozen copy. From memory profiling the largest leak in ruby gems is strings that really should be deduped using a mechanism like it.

raised a separate issue on this: http://bugs.ruby-lang.org/issues/8977

#3 - 10/02/2013 01:18 PM - sam.saffron (Sam Saffron)
Can we also have a global switch to enable this everywhere (for debugging), it can make it simple to isolate the spots where it would fall over.

#4 - 10/02/2013 01:23 PM - ko1 (Koichi Sasada)
(2013/10/02 13:18), sam.saffron (Sam Saffron) wrote:

Can we also have a global switch to enable this everywhere (for debugging), it can make it simple to isolate the spots where it would fall over.

+1. It should be another ticket.

--
// SASADA Koichi at atdot dot net

#5 - 10/02/2013 03:08 PM - akr (Akira Tanaka)

akr (Akira Tanaka) wrote:

There are several problems for f-suffix:

- The notation is ugly.

I forgot to mention Akira Matsuda's presentation at RubyShift 2013:
http://sssslide.com/speakerdeck.com/a_matsuda/rails-engines-from-the-bottom-up
(The presentation shows code snippets with f-suffix extensively.)

He said that the response of audience for f-suffix was negative.

#6 - 10/02/2013 03:18 PM - charliesome (Charlie Somerville)

I forgot to mention Akira Matsuda's presentation at RubyShift 2013:
http://sssslide.com/speakerdeck.com/a_matsuda/rails-engines-from-the-bottom-up
(The presentation shows code snippets with f-suffix extensively.)

He said that the response of audience for f-suffix was negative.

To be fair, you wouldn't really use f-suffix strings in all the places they were used in the presentation. They're only really useful in tight loops, or in code where aesthetics does not matter (eg. code generated from ERB).

#7 - 10/02/2013 09:18 PM - sobrinho (Gabriel Sobrinho)

Maybe I'm too late but why not use the same object when calling String#freeze?

I mean, currently this:

```
"something".freeze.object_id
=> 70273877530260
"something".freeze.object_id
=> 70273877536840
```

08/31/2022
And change the compiler to do this:

```
"something".freeze.object_id
=> 70273877530260
"something".freeze.object_id
=> 70273877530260
```

Not sure about the work that need to be done and even if it's possible but it would maintain the syntax compatibility with legacy versions of Ruby.

I'm not against the "something" syntax, regardless it's really strange for the ruby idiom, I'm concerned about libraries that needs to run on legacy rubies.

---

**#8 - 10/02/2013 11:36 PM - enebo (Thomas Enebo)**

I think having a pragma at the top of the file will be much more error prone than the f-syntax. As a file grows, the ability to notice you are in a frozen string file goes down. It would have been great if Ruby had started immutable strings by default but that ship has sailed, I think having some files be immutable will be confusing.

Are we sure we cannot find a nicer syntax for frozen strings: %f{hello, I am frozen}?

---

**#9 - 10/03/2013 01:38 AM - headius (Charles Nutter)**

I agree with Tom here. I think it's going to be almost useless to have a full-file "freeze-string" directive.

- From file to file, the meaning of a literal string would change. This would be confusing for everyone dealing with a project. They'd get frozen string errors in one file and not in another for exactly the same code.

- Users would be forced to create new mutable strings with String.new. There would be no other way. So in one file you could create a new mutable string with "" and in another you'd have to use String.new.

It would be a very bad idea to have a directive that completely changes the meaning of code from one file to another.

---

**#10 - 10/03/2013 02:26 AM - brixen (Brian Shirai)**

It would be a very bad idea to have a directive that completely changes the meaning of code from one file to another.

For consistency sake, it should be noted that, in fact, this is exactly what the existing encoding pragma does, and it's also the express purpose of refinements.

Hence, a more nuanced argument than this broad stroke of "very bad idea" may be needed.

---

**#11 - 10/03/2013 02:36 AM - headius (Charles Nutter)**

brixen (Brian Shirai) wrote:

For consistency sake, it should be noted that, in fact, this is exactly what the existing encoding pragma does, and it's also the express purpose of refinements.

The encoding directive changes the interpretation of the bytes within strings, but does not change their behavior. If m17n is working properly, you may never even see a difference in code, since even strings with different encodings can be negotiated into combining, matching regexp, and converting to other encodings.

Refinements change the meaning of code within a lexical scope...not within an entire file (unless it is the file's scope that is being refined). This is more analogous to instance_eval on a block, which changes what "self" methods are called against. You are correct that they do change the meaning of code within their scope, but whether that's a good feature or not is beyond the scope of this discussion. I do not particularly like refinements.

A frozen string directive would actually change the behavior of the strings in that file, making operations that worked before fail to work under the directive. Encoding does not make some methods on string start to raise errors, except where you may have differing encodings (which can happen without an encoding directive too).

Hence, a more nuanced argument than this broad stroke of "very bad idea" may be needed.

I'm not sure this is the place to have a meta-argument about how to argue for or against this proposal. But since you suggest a more nuanced argument, I suggest you look at the original points in my comment that explain why it would be a bad idea.

Do you have any arguments to make for or against this proposal?
Brian since I have been able to infer you dislike both M17n and refinements that you agree with Charlie and I that this particular pragma might not be an idea you endorse? Perhaps you can elucidate a better argument against it?

enebo (Thomas Enebo) wrote:

I think having a pragma at the top of the file will be much more error prone than the f-syntax. As a file grows, the ability to notice you are in a frozen string file goes down. It would have been great if Ruby had started immutable strings by default but that ship has sailed, I think having some files be immutable will be confusing.

Enhance your IDE.

Are we sure we cannot find a nicer syntax for frozen strings: %f{hello, I am frozen}?

Almost all of the idea to add new syntax break 2.0 compatibility, and it makes adoption of frozen strings slow. It is the motivation of this suggestion.

On 2013/10/03 2:27, brixen (Brian Shirai) wrote:

Issue #8976 has been updated by brixen (Brian Shirai).

It would be a very bad idea to have a directive that completely changes the meaning of code from one file to another.

For consistency sake, it should be noted that, in fact, this is exactly what the existing encoding pragma does,

The reason why there is an encoding pragma, and why it's per file, is because text editors deal with one encoding per file. Doing something like an encodingpragma e.g. on a block basis would not work well together with editors.

I agree with Charles and others that a file-based directive isn't a good idea for frozen/fixed strings.

From a more general perspective, it feels to me that introducing all these frozen options will increase performance, but at the cost of programmer effort. That would be the case also e.g. for something like type hints,..., but that's not Ruby style.

Regards, Martin.

and it's also the express purpose of refinements.

Hence, a more nuanced argument than this broad stroke of "very bad idea" may be needed.

Feature #8976: file-scope freeze_string directive
https://bugs.ruby-lang.org/issues/8976#change-42221

Author: akr (Akira Tanaka)
Status: Open
Priority: Normal
Assignee:
Category:
Target version: current: 2.1.0

From a more general perspective, it feels to me that introducing all these frozen options will increase performance, but at the cost of programmer effort. That would be the case also e.g. for something like type hints,..., but that's not Ruby style.
Personally, I think the more important benefit of having instantly-frozen literal strings, arrays, and hashes is for safer concurrency and data integrity.

- If I pass you a reference to a string, I can create that string frozen and be sure you don't modify it. Same goes for arrays and hashes.
- If I am initializing a global array or hash that should never be modified, I can create it frozen immediately.

Of course these can all be done by calling .freeze on the object as well, but creating immutable structures right away avoids any mistakes. And then the potential VM optimizations are a bonus on top of that.

But I stand by my opinion that a global pragma for frozen literal strings is a bad idea, because it makes all literal strings in the file start raising errors for half their methods, and it makes it impossible to copy/paste from one file to another without code potentially breaking.

#16 - 10/03/2013 11:51 PM - shevegen (Robert A. Heiler)
I am mildly in favour of it so +1

As it is compatible with older ruby I see little harm in it. But please don't forget proper documentation, if this is given the thumbs up by matz!

#17 - 10/04/2013 12:10 AM - enebo (Thomas Enebo)
naruse (Yui NARUSE) wrote:
enebo (Thomas Enebo) wrote:

I think having a pragma at the top of the file will be much more error prone than the f-syntax. As a file grows, the ability to notice you are in a frozen string file goes down. It would have been great if Ruby had started immutable strings by default but that ship has sailed, I think having some files be immutable will be confusing.

Enhance your IDE.

It is an answer but one I think is not acceptable (obviously that is only my opinion).

Are we sure we cannot find a nicer syntax for frozen strings: %f{hello, I am frozen}?

Almost all of the idea to add new syntax break 2.0 compatibility, and it makes adoption of frozen strings slow.

It is the motivation of this suggestion.

Yeah, ko1 talked to me yesterday about this. I have been trying to think of other ways to not break backwards compatibility and have not thought of anything.

It is possible to put out a PL to 2.0 to add the syntax but obviously a decision like that is a big one. Older 2.0 libraries will not be able to read it.

OTOH, MRI has been periodically putting out security fixes so perhaps a syntax error to force an upgrade is not a bad thing?

#18 - 10/04/2013 12:12 AM - enebo (Thomas Enebo)
enere (Thomas Enebo) wrote:
naruse (Yui NARUSE) wrote:
enere (Thomas Enebo) wrote:

I think having a pragma at the top of the file will be much more error prone than the f-syntax. As a file grows, the ability to notice you are in a frozen string file goes down. It would have been great if Ruby had started immutable strings by default but that ship has sailed, I think having some files be immutable will be confusing.

Enhance your IDE.

It is an answer but one I think is not acceptable (obviously that is only my opinion).

Are we sure we cannot find a nicer syntax for frozen strings: %f{hello, I am frozen}?

Almost all of the idea to add new syntax break 2.0 compatibility, and it makes adoption of frozen strings slow.

It is the motivation of this suggestion.

Yeah, ko1 talked to me yesterday about this. I have been trying to think of other ways to not break backwards compatibility and have not thought of anything.
It is possible to put out a PL to 2.0 to add the syntax but obviously a decision like that is a big one. Older 2.0 libraries will not be able to read it. OTOH, MRI has been periodically putting out security fixes so perhaps a syntax error to force an upgrade is not a bad thing.

Read "Older 2.0 libraries will not be able to read it" as "Older MRI 2.0 implementations will not be able to read libraries which use this new syntax."

---

#19 - 10/07/2013 11:53 PM - akr (Akira Tanaka)
2013/10/2 enebo (Thomas Enebo) tom.enebo@gmail.com:

Issue #8976 has been updated by enebo (Thomas Enebo).

I think having a pragma at the top of the file will be much more error prone than the f-syntax. As a file grows, the ability to notice you are in a frozen string file goes down. It would have been great if Ruby had started immutable strings by default but that ship has sailed, I think having some files be immutable will be confusing.

I think it is not a big problem because we don't need to aware the directive is exist or not if we use "..." to strings not modified and "...".dup to strings to be modified.

Tanaka Akira

#20 - 10/08/2013 08:50 AM - mame (Yusuke Endoh)

"...".dup looks too verbose to me.
How about using "..." for a mutable string and '...' for an immutable?

I'm not so keen on a file-scope directive itself, though...

--

Yusuke Endoh mame@tsg.ne.jp

#21 - 10/08/2013 09:29 AM - akr (Akira Tanaka)
2013/10/8 mame (Yusuke Endoh) mame@tsg.ne.jp:

Issue #8976 has been updated by mame (Yusuke Endoh).

"...".dup looks too verbose to me.
How about using "..." for a mutable string and '...' for an immutable?

I considered it at first.
But we (at the meeting) abandoned it because we want to use escape sequences, such as \\
, in immutable strings.

I wrote this concern in [ruby-core:57574] as follows.

| Note that the directive effects all static string literals regardless of |
| single quotes, double quotes, %q-string, %qq-string and here documents. |
| The reason that the directive is effective not only single quotes is |
| we want to use escape sequences such as \
 in frozen string literals. |

Tanaka Akira

#22 - 10/08/2013 04:03 PM - Anonymous

"..." might be mildly ugly, but is hard to beat.
5 minutes of my thinking did not yield any better idea.
I share mame's feeling about the file-scope directive.

#23 - 10/10/2013 12:43 AM - headius (Charles Nutter)
boris_stitnicky (Boris Stitnicky) wrote:

"..." might be mildly ugly, but is hard to beat.
5 minutes of my thinking did not yield any better idea.
I share mame's feeling about the file-scope directive.

I still don't like file-scope directive since it changes behavior rather than just content. In other words, a file that gains a frozen string directive suddenly...
creates strings that are only half functional.

See also #8992 that might address all issues by simply making the compiler and #freeze methods smarter.

- Compiler would see through "literal".freeze and do what "literal"f" does now.
- String#freeze could be adapted to use the fstring cache internally, so all frozen strings would be interned (in the Java sense).
- No new backward-incompatible syntax.
- Easy expansion to other literal syntaxes like arrays and hashes.

I think we need to kill off the "literal"f syntax and a file-scope directive is not the right way to do it.

---

#24 - 10/10/2013 12:59 AM - akr (Akira Tanaka)
2013/10/10 headius (Charles Nutter) headius@headius.com:

Issue #8976 has been updated by headius (Charles Nutter).

See also #8992 that might address all issues by simply making the compiler and #freeze methods smarter.

- Compiler would see through "literal".freeze and do what "literal"f" does now.
- String#freeze could be adapted to use the fstring cache internally, so all frozen strings would be interned (in the Java sense).
- No new backward-incompatible syntax.
- Easy expansion to other literal syntaxes like arrays and hashes.

#8992 doesn't address the problem follows.

| * Need to modify for each string literal.
| This is cumbersome.

Tanaka Akira

---

#25 - 10/10/2013 06:33 PM - Eregon (Benoit Daloze)

akr (Akira Tanaka) wrote:

2013/10/10 headius (Charles Nutter) headius@headius.com:

Issue #8976 has been updated by headius (Charles Nutter).

See also #8992 that might address all issues by simply making the compiler and #freeze methods smarter.

- Compiler would see through "literal".freeze and do what "literal"f" does now.
- String#freeze could be adapted to use the fstring cache internally, so all frozen strings would be interned (in the Java sense).
- No new backward-incompatible syntax.
- Easy expansion to other literal syntaxes like arrays and hashes.

#8992 doesn't address the problem follows.

| * Need to modify for each string literal.
| This is cumbersome.

Tanaka Akira

I think freezing these literals by adding ".freeze" to each of them is appropriate, these literals should already (in current version) be frozen to prevent any modification.

Changing semantics file-based is much too dangerous. And calling #dup just to have a mutable static literal String has no good meaning, it is just weird.

---

#26 - 01/30/2014 04:50 AM - hsbt (Hiroshi SHIBATA)
- Target version changed from 2.1.0 to 2.2.0

#27 - 10/16/2014 06:38 AM - akr (Akira Tanaka)

Recent Eric Wong's effort reminds me this issue.

I still think this issue is worth to consider.

Ruby 2.1 changed the semantics of "...".freeze to avoid string allocation.
It is not used extensively, I feel.

File scope directive provides a way to use frozen string literals with much fewer modification to programs.

Also, I think frozen string literals is a better programming style. It needs dup call ("...".dup) for string literals to be modified. It makes us to prevent unintentional modification and we can distinguish string literals to be modified or not, more easily.

I would like that future Ruby (Ruby 3.0?) will interpret string literals as frozen by default. This issue can be considered as a migration path. We can introduce file-scope directive now and change the default at future when most programs uses frozen string literals.

#28 - 10/16/2014 08:01 AM - normalperson (Eric Wong)

akr@fsji.org wrote:

Recent Eric Wong's effort reminds me this issue.

I still think this issue is worth to consider.

Ruby 2.1 changed the semantics of "...".freeze to avoid string allocation. It is not used extensively, I feel.

Right, "...".freeze is too ugly IMHO. Ruby should stay beautiful :) File scope directive provides a way to use frozen string literals with much fewer modification to programs.

Also, I think frozen string literals is a better programming style. It needs dup call ("...".dup) for string literals to be modified.

I also think needing to call "...".dup is ugly and will break much existing code[1].

It makes us to prevent unintentional modification and we can distinguish string literals to be modified or not, more easily.

My concern is optimizing Ruby for the typical script language users[2], not the (few) Rubyists who understand VM internals.

I prefer we continue to improve Ruby, transparently:

- 2.1 (past) - { "str" => ... }, "str".freeze
- 2.2 (done) - {str} (= ...)
- 2.2 (planned) - {gsub/sub/tr/tr_s}(["str"]|/../, "str"), ==, %, many more core (cfunc) methods

All of the above are transparent to existing code.

I would like that future Ruby (Ruby 3.0?) will interpret string literals as frozen by default. This issue can be considered as a migration path. We can introduce file-scope directive now and change the default at future when most programs uses frozen string literals.

I am against this; especially as a default for 3.0[1]. File scope directives makes refactoring and reorganization of code more difficult: move a method to a new file and behavior changes.

I hope we can implement more optimization transparently in the compiler. For example; we should be able to optimize more cases:

```ruby
def foo(a, b)
a.gsub(/baz/, b)
end

foo(a, "lit") # no dup if 'a' is a string
```
I think we can also use (internal) C-API changes to mark cfuncs as taking const args. It would be useful for things like Hash#merge!, too; not just string literals.

[1] - Even for Ruby 3.0; I strongly favor maintaining backwards compatibility as much as possible in Ruby-land.
1.8 -> 1.9 was very painful for some users (including myself)

#29 - 10/26/2014 08:50 PM - akr (Akira Tanaka)
Eric Wong wrote:

  Right, "...".freeze is too ugly IMHO. Ruby should stay beautiful :)

Agreed.

  I also think needing to call "...".dup is ugly and will break much existing code[1].

I don't think "...".dup is ugly.

I agree changing the default breaks much existing code.
So changing the default is long term goal.

  It makes us to prevent unintentional modification and we can distinguish string literals to be modified or not, more easily.

My concern is optimizing Ruby for the typical script language users[2], not the (few) Rubyists who understand VM internals.

Unnecessary string duplications problem is easy to understand.
I think there is possibility to change the users.

  I am against this; especially as a default for 3.0[1]. File scope directives makes refactoring and reorganization of code more difficult: move a method to a new file and behavior changes.

There are many context dependency for refactoring and reorganization now.
One more context dependency is not a big problem.

#30 - 08/21/2015 07:43 AM - matsuda (Akira Matsuda)
Let us bring this two-year-aged very well matured feature proposal back to the table!

I would like to propose again to put this magic comment for freezing String literals into next version of Ruby.

motivation

There are several reasons that we need this feature now:

(1) Freezing Strings is getting people's attraction recently, because people have been started realizing that they can somewhat improve their app performance by freezing Strings. Hence, we see so many "performance improvement" patches that add .freeze to String literals here and there for frameworks, libraries, and even for ruby stdlibs.

These "performance improvement".freeze patches would certainly improve some micro-benchmark results, but at the same time inject code ugliness in exchange for the speed. This apparently is a wrong approach. This is not a Ruby way. Ruby code must be kept beautiful.

(2) While discussing this freezing magic comment feature, Matz finally decided to make all String literals frozen by default (#11473).

Having a magic comment switch per file would be a very good transition path to the coming big change in Ruby 3.

the magic comment

We chose this at the meeting.

# frozen_string_literal: true

command line option
A command line option switch for "ruby 3 mode" will also be added. If the option and the magic comment conflicts, the magic comment should win.

**creating a String with an Encoding**

While investigating ruby code in the stdlib, we found that in most cases mutating literal Strings happens in this form

```
"".force_encoding(Encoding::ASCII_8BIT)
```

so we decided to introduce a new String constructor taking Encoding via kwarg as follows:

```
String.new("", encoding: Encoding::ASCII_8BIT)
```

**current status**

Matz already accepted this proposal at the developer meeting in Tokyo yesterday, so this feature will be put into 2.3 unless we see any strong objection here.

#31 - 08/21/2015 02:50 PM - jeremyevans0 (Jeremy Evans)

I don't think the magic comment approach for freezing string literals is a good idea. For ruby libraries that are maintained, the maintainers will change their code so that they work when string literals are frozen by default (in ruby 3). For ruby libraries that are not maintained, the magic comment will not be added.

I am strongly in favor of freezing string literals by default in ruby 3. Most of the objections I've read against freezing string literals by default are regarding breaking of unmaintained gems. I think all ruby needs is a command line flag (e.g. --freeze-string-literals=no|yes) or environment variable (e.g. RUBY_FREEZE_STRING_LITERALS=0|1) that sets whether string literals should be frozen by default.

With the command line flag or environment variable, you can test in ruby 2.3 that your code will work in ruby 3. In ruby 3, if you are using a library/app that doesn't work with frozen string literals, you can turn it off. Turning off frozen string literals in ruby 3 will make things work, but hurt performance for the entire application, which will give people a reason to fix the code. If you add the ability for per-file magic comments, it will encourage people not to make their code work with frozen string literals, and it will make it harder to reason about code that uses string literals, since you'll have to check for a magic comment every time.

The only reason I can think of to support a magic comment for frozen string literals is if the maintainer of a library did not know how to make their library work when string literals are frozen by default. They could then use a magic comment and sweep the problem under the rug. Is that something we want to encourage?

#32 - 08/21/2015 10:18 PM - akr (Akira Tanaka)

Jeremy Evans wrote:

I don't think the magic comment approach for freezing string literals is a good idea. For ruby libraries that are maintained, the maintainers will change their code so that they work when string literals are frozen by default (in ruby 3). For ruby libraries that are not maintained, the magic comment will not be added.

I am strongly in favor of freezing string literals by default in ruby 3. Most of the objections I've read against freezing string literals by default are regarding breaking of unmaintained gems. I think all ruby needs is a command line flag (e.g. --freeze-string-literals=no|yes) or environment variable (e.g. RUBY_FREEZE_STRING_LITERALS=0|1) that sets whether string literals should be frozen by default.

I think changing the default behavior without migration path is too big pain.

A command line option, --enable-frozen-string-literal, is also planned at the developer meeting DevelopersMeeting20150820Japan. [https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20150820Japan](https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20150820Japan) [https://docs.google.com/document/d/1e00tTj8ix2ofS8H2RiIMUhr39bABSc7AL0vk4KbtSF4/pub](https://docs.google.com/document/d/1e00tTj8ix2ofS8H2RiIMUhr39bABSc7AL0vk4KbtSF4/pub)

Of course, its companion, --disable-frozen-string-literal, will be available too.

With the command line flag or environment variable, you can test in ruby 2.3 that your code will work in ruby 3. In ruby 3, if you are using a library/app that doesn't work with frozen string literals, you can turn it off. Turning off frozen string literals in ruby 3 will make things work, but hurt performance for the entire application, which will give people a reason to fix the code. If you add the ability for per-file magic comments, it will encourage people not to make their code work with frozen string literals, and it will make it harder to reason about code that uses string literals, since you'll have to check for a magic comment every time.

The only reason I can think of to support a magic comment for frozen string literals is if the maintainer of a library did not know how to make their library work when string literals are frozen by default. They could then use a magic comment and sweep the problem under the rug. Is that something we want to encourage?

file-scope directive enables us to update libraries incrementally.

We can't update all library at once. There are too many libraries and authors.
Some libraries will be updated soon and other libraries don't.
We can obtain performance benefit from updated libraries.
This encourage library update with smaller pain.

#33 - 08/22/2015 12:06 AM - colindkelley (Colin Kelley)
Akira Tanaka wrote:

We can't update all libraries at once.
There are too many libraries and authors.
Some libraries will be updated soon and other libraries don't.

I completely concur. We will need to work through many projects and libraries and gems to make this change. The comment directive is perfect for that because we can mark files/projects/gems that have been converted and tested while not disrupting others. This is exactly how we managed the transition to utf-8 and that went very smoothly.

We can obtain performance benefit from updated libraries.

Yes, we want the benefit sooner than Ruby 3.0, and we want to make the transition over time, before 3.0 becomes mainstream. Just like with utf-8.

Also, here is a command line gem to automatically add the magic comment to the .rb files in the current folder:

https://github.com/Invoca/magic_frozen_string_literal

#34 - 08/22/2015 01:22 AM - jeremyevans0 (Jeremy Evans)
Colin Kelley wrote:

Akira Tanaka wrote:

We can't update all libraries at once.
There are too many libraries and authors.
Some libraries will be updated soon and other libraries don't.

I completely concur. We will need to work through many projects and libraries and gems to make this change. The comment directive is perfect for that because we can mark files/projects/gems that have been converted and tested while not disrupting others. This is exactly how we managed the transition to utf-8 and that went very smoothly.

Adding support for this magic comment will ensure that this problem will remain a problem far in the future. We should encourage people to fix the problem, not sweep the problem under the rug.

This is not like the encoding magic comment, which is necessary because the ruby source file could be in any encoding. In general, frozen string literal issues are easy to fix, unlike encoding issues. I'd guess that 80% of exceptions caused by this change will be on the line after the string literal.

No library should be mutating a string they don't own as a side effect, and this change will make it much easier to detect libraries that do. Adding a magic comment to such a library will make it more likely that the problem isn't discovered even after ruby 3.0 is released. We should not encourage behavior that will hide bugs.

Any time spent adding magic comments to existing libraries would be better spent just making sure that libraries work with frozen string literals. Assuming we have at least 2 years between 2.3 and 3.0, that should be enough time to test gems for compatibility with --enable-frozen-string-literal before such behavior becomes the default.

It needs to be painful to use libraries that don't support frozen string literals, in order to encourage people to fix those libraries. For those people who must use libraries that still don't support frozen string literals when ruby 3 is released, they can use --disable-frozen-string-literals and be no worse off than they are now.

#35 - 08/22/2015 07:01 AM - funny_falcon (Yura Sokolov)
New constructor is a good thing.
But it is better to allow construction

"":b

to be mutable string. "".force_encoding(Encoding::ASCII_8BIT) is used for compatibility with Ruby 1.9.3 (cause "":b were not backported :)

#36 - 08/22/2015 09:55 PM - wanabe (_wanabe)
- Related to Feature #11473: Immutable String literal in Ruby 3 added

#37 - 08/23/2015 02:38 AM - akr (Akira Tanaka)
It needs to be painful to use libraries that don't support frozen string literals, in order to encourage people to fix those libraries. For those people who must use libraries that still don't support frozen string literals when ruby 3 is released, they can use --disable-frozen-string-literals and be no worse off than they are now.

I don't think it should be painful.
The changing default at Ruby 3.0 is big pain.
So, we should mitigate the pain.

#38 - 08/25/2015 12:10 PM - mame (Yusuke Endoh)

Akira Matsuda wrote:

Hence, we see so many "performance improvement" patches that add .freeze to String literals here and there for frameworks, libraries, and even for ruby stdlibs.

I'm interested in the actual effect of this feature. How many times faster will Rails run actually? Anyone measured?

https://github.com/rails/rails/commit/5bb1d4d288d019e276335465d0389d2f5246bf4

This committer seemed to perform only micro benchmark. Was there any reason why he didn't measure the actual time?

0.453037333993266 miliseconds saved per request

So we're shaving off 1 second of execution time for every 220 requests.

Calculation looks wrong. Correctly, 1 second per 2200 requests?

--
Yusuke Endoh mame@ruby-lang.org

#39 - 08/25/2015 04:30 PM - mame (Yusuke Endoh)

Yusuke Endoh wrote:

I'm interested in the actual effect of this feature. How many times faster will Rails run actually? Anyone measured?

I did.

Experimental overview:

- I used akr's "immutable by default" branch
- I executed:
  - gem install rails
  - rails new scaffold-test
  - rails s
  - open "Welcome aboard" page
  - ab -u 1000 -c 100 http://localhost:3000
- I compared it with 2.2.1p85 (released version)
- Whenever "can't modify frozen String" exception is raised, I created and applied a very ad-hoc fix. I didn't count the fixes precisely, but I think about 50 fixes are needed. (rdoc, bundler, and gem install sqlite3, required many fixes.)

Results: Requests per second

- immutable by default branch: 167.14 [#/sec]
- 2.2.1p85: 168.42 [#/sec]

I tried a few times, but I couldn't see any difference.

Note:

- The current source of Rails already includes many .freeze literals. So there may be little room for improvement by this feature. The same experiment with removing all .freeze might be valuable.
- I don't know how meaningful it is to measure the "requests per second" of "Welcome aboard" page on WEBrick.
- My fixes for "can't modify frozen String" is really ad-hoc; they may break something.
- Sorry, I didn't record my fixes, so I cannot provide the sufficient information for replication. But I could perform this experiment in a few hours, so I think it is not so difficult for other people to replicate this experiment. (I'm not familiar with Rails. Actually I used Rails for the first time in about 8 years.)
Here you can see more information about the actual effect of frozen string on Ruby on Rails [https://github.com/rails/rails/pull/21057](https://github.com/rails/rails/pull/21057). It is 11% in a relative small application, in big applications this improvement can be bigger.

I don't know how meaningful it is to measure the "requests per second" of "Welcome aboard" page on WEBrick.

You are hitting a static page so it will not even hit the default Rails stack. You could try to hit the scaffold-test URL to get more meaningful results.

It seems (for me) frozen string literals are good trade-off between current Ruby (every string is mutable) and immutable strings. If all strings are immutable, it introduces huge incompatibility. With frozen string literals, we may be able to find more errors earlier. Or maybe some Ruby programs run faster.

But first we have to prove if this is a good idea. So my plan is to introduce frozen-string-literal magic comment (pragma) for 2.3, and see whether it works well or not. If it works well, I'd love to make string literals frozen by default in Ruby3. But if not, frozen-string-literal ends as an experiment.

Matz.

Nobuyoshi Nakada wrote:

```
% ./ruby -e '/*-* frozen_string_literal: true */
p "foo".frozen?'
true
```

Would you please implement the proposed command line options, --{enable,disable}-frozen-string-literal?

Great. It works fine.

Applied in changeset r51953.

fronzen-string-literal pragma

- compile.c (iseq_compile_each): override compile option by option given by pragma.
- iseq.c (rb_iseq_make_compile_option): extract a function to overwrite rb_compile_option_t.
- parse.y (parser_set_compile_option_flag): introduce pragma to override compile options.
- parse.y (magic_comments): new pragma "fronzen-string-literal". [Feature #8976]

Akira Matsuda wrote:
the magic comment

We chose this at the meeting.

# frozen_string_literal: true

This does not work correctly. The current implementation needs -*-.

# -*- frozen_string_literal: true -*-

Since encoding magic comments do not require -*-, I don't expect frozen_string_literal requires it.

If users write # frozen_string_literal: true (in fact it make no effect) and see the code to work, she/he will misunderstand that no change is required. It is dangerous.

--
Yusuke Endoh mame@ruby-lang.org

#46 - 10/03/2015 05:59 AM - akr (Akira Tanaka)
- Status changed from Open to Closed

Yusuke Endoh wrote:

Since encoding magic comments do not require -*-, I don't expect frozen_string_literal requires it.

If users write # frozen_string_literal: true (in fact it make no effect) and see the code to work, she/he will misunderstand that no change is required. It is dangerous.

I also surprised that.

However, I, you and Richard Schneeman find -*- correctly.


So, currently I think it is acceptable.

Anyway it is orthogonal to frozen_string_literal itself. Please open another ticket if you think it is really a problem.

#47 - 10/03/2015 10:42 AM - mame (Yusuke Endoh)
- Status changed from Closed to Open
- Assignee set to nobu (Nobuyoshi Nakada)

Anyway it is orthogonal to frozen_string_literal itself.

I do NOT think so.

The current problem I concern is that "frozen_string_literal" requires -*-. This feature is the key test that decides Ruby's feature. If people fail to use this feature, it will lead to the wrong decision. And it is dangerous at least in the current situation, because people are only familiar with "encoding" magic comment that does not require -*-.

I do NOT intend to discuss the format of magic comment in general. I personally prefer no -*-. But it is just my preference. I don't feel a danger or problem, at least currently, because there is no other important kind of magic comments. (I don't think that warn_indent is critical.)

--
Yusuke Endoh mame@ruby-lang.org

#48 - 10/03/2015 11:40 AM - kou (Kouhei Sutou)

I object -*- frozen_string_literal: true -*- because it shows the following buffer on Emacs when I save or open the file.

The local variables in a.rb contains values that may not be safe (*).

Do you want to apply it? You can type
y -- to apply the local variables list.
n -- to ignore the local variables list.
! -- to apply the local variables list, and permanently mark these values (*) as safe (in the future, they will be set automatically.)

* frozen_string_literal : true

frozen_string_literal: true doesn't show it.

#49 - 10/03/2015 04:12 PM - nobu (Nobuyoshi Nakada)
Kouhei Sutou wrote:

I object -*- frozen_string_literal: true -*- because it shows the following buffer on Emacs when I save or open the file.

Put another line before it.

#50 - 10/04/2015 05:42 AM - duerst (Martin Dürst)
I agree with Yusuke Endoh and Kouhei Sutou: requiring -*- creates unnecessary variation among directives. Also, putting another line before it (as Nobu suggests) to avoid problems in Emacs is counter to the idea that all these directives should be grouped at the start of the file.

#51 - 10/05/2015 05:55 AM - akr (Akira Tanaka)
Sutou-san's objection appeal me.

I occur "File mode specification error: (void-variable ruby-font-lock-keywords)" with emacs. Nakada-san's workaround (another line before it) didn't help.

There is no problem with vim which is my favorite editor, though.

#52 - 10/05/2015 07:04 AM - nobu (Nobuyoshi Nakada)
What line did you put?

Emacs opens this file fine.

#
# -*- frozen-string-literal: true -*-
p "".frozen?

#53 - 10/05/2015 07:14 AM - akr (Akira Tanaka)
Nobuyoshi Nakada wrote:

What line did you put?

Emacs opens this file fine.

#
# -*- frozen-string-literal: true -*-
p "".frozen?

This file still shows "File mode specification error: (void-variable ruby-font-lock-keywords)" in minibuffer.

boron% cat z.rb
#
# -*- frozen-string-literal: true -*-
p "".frozen?
boron% emacs -q z.rb
boron% emacs --version
GNU Emacs 21.4.1
Copyright (C) 2002 Free Software Foundation, Inc.
GNU Emacs comes with ABSOLUTELY NO WARRANTY.
You may redistribute copies of Emacs under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING.

#54 - 10/05/2015 01:10 PM - nobu (Nobuyoshi Nakada)
21...
Does ruby-mode.el work?
That is, "(void-variable ruby-font-lock-keywords)" means ruby-mode.el doesn't support your Emacs. It's irrelevant to frozen-string-literal at all.

I see.

The error is happen without -- frozen-string-literal: true --. I understand that my emacs problem is not related to "-".

Does nobu seriously require all Rubyists to write codes like this? ;-) https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ac4cbe76e83321640443

Leave emacs aside, I still believe this magic comment must not require "-". It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

In r52087, I've changed the behavior not to freeze dynamic string literals (e.g., "#{x}") because dynamic string literals don't literally represent strings.

Please give us your feedback if you have any objection.

What about "#{'foo'}"?

"#{'foo'}" can be considered literal (or static), but there's still room for discussion about it.

Another case is "#{foo}" -- if that is treated as a dynamic string, it could become an alternative idiom for "foo".dup

"#{foo}" can be considered literal (or static), but there's still room for discussion about it.

Another case is "#{foo}" -- if that is treated as a dynamic string, it could become an alternative idiom for "foo".dup

"#{foo}" is different slightly, -- it is equivalent to "#{nil.to_s}foo" and depends on NilClass#to_s, so it can't be a static literal.

Another case is "#{foo}" -- if that is treated as a dynamic string, it could become an alternative idiom for "foo".dup

"#{foo}" is different slightly, -- it is equivalent to "#{nil.to_s}foo" and depends on NilClass#to_s, so it can't be a static literal.
Benoit Daloze wrote:

Yusuke Endoh wrote:

Does nobu seriously require all Rubyists to write codes like this? ;-)  

https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ec4cbe6e83321640443  

Leave emacs aside, I still believe this magic comment must not require -*-. It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

+1

This issue should be discussed at the developers meeting today, and Matz should desicde whether -*- should be required.

---

Yusuke Endoh wrote:

> Does nobu seriously require all Rubyists to write codes like this? ;-)  
>  
> https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ec4cbe6e83321640443  

Leave emacs aside, I still believe this magic comment must not require -*-. It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

---

This issue should be discussed at the developers meeting today, and Matz should decide whether -*- should be required.

---

Benoit Daloze wrote:

Yusuke Endoh wrote:

Does nobu seriously require all Rubyists to write codes like this? ;-)  

https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ec4cbe6e83321640443  

Leave emacs aside, I still believe this magic comment must not require -*-. It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

+1

This issue should be discussed at the developers meeting today, and Matz should decide whether -*- should be required.

---

Yusuke Endoh wrote:

> Does nobu seriously require all Rubyists to write codes like this? ;-)  
>  
> https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ec4cbe6e83321640443  

Leave emacs aside, I still believe this magic comment must not require -*-. It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

+1

This issue should be discussed at the developers meeting today, and Matz should decide whether -*- should be required.

---

Yusuke Endoh wrote:

> Does nobu seriously require all Rubyists to write codes like this? ;-)  
>  
> https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ec4cbe6e83321640443  

Leave emacs aside, I still believe this magic comment must not require -*-. It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

+1

This issue should be discussed at the developers meeting today, and Matz should decide whether -*- should be required.

---

Yusuke Endoh wrote:

> Does nobu seriously require all Rubyists to write codes like this? ;-)  
>  
> https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ec4cbe6e83321640443  

Leave emacs aside, I still believe this magic comment must not require -*-. It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

+1

This issue should be discussed at the developers meeting today, and Matz should decide whether -*- should be required.

---

Yusuke Endoh wrote:

> Does nobu seriously require all Rubyists to write codes like this? ;-)  
>  
> https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ec4cbe6e83321640443  

Leave emacs aside, I still believe this magic comment must not require -*-. It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

+1

This issue should be discussed at the developers meeting today, and Matz should decide whether -*- should be required.

---

Yusuke Endoh wrote:

> Does nobu seriously require all Rubyists to write codes like this? ;-)  
>  
> https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ec4cbe6e83321640443  

Leave emacs aside, I still believe this magic comment must not require -*-. It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

+1

This issue should be discussed at the developers meeting today, and Matz should decide whether -*- should be required.

---

Yusuke Endoh wrote:

> Does nobu seriously require all Rubyists to write codes like this? ;-)  
>  
> https://github.com/ruby/ruby/commit/36ca18b84715d0c92a82ec4cbe6e83321640443  

Leave emacs aside, I still believe this magic comment must not require -*-. It is too error-prone.

Agreed, to me this is a fatal flaw of the current implementation of parsing the magic comment and it is very inconsistent.

+1

This issue should be discussed at the developers meeting today, and Matz should decide whether -*- should be required.
accept multi-lines at the beginning of programming.
accept comments between pragmas.

akr: how about dynamic strings “foo#{bar}baz”?
nobu: how about “#{‘baz’}”?
k01: it is easy to understand all “...” is frozen. Shugo-san’s first comment is for “compatibility”.
matz: i agree to

matz: i have another idea +“...” -> mutable, -“...” -> immutable. it is more clean than “...”.freeze. how about that, instead of the magic comment? i don’t care about pull-requests, but care about the representation of source code.
akr: ...
matz: i don’t like magic comment, so that i hesitate to introduce it.

MISC:
akr: should we separate this topic to another ticket? this is “how to write a pragma”.
k01: we should describe syntax of “how to write pragmas”

It’s not just for compatibility.
The original problem was that a new String object has to be allocated by a string literal for each evaluation. so i don’t understand the reason why a dynamic string literal should be frozen in spite of the fact freezing dynamic strings can’t reduce object allocation.

MISC:
akr: how about dynamic strings “foo#{bar}baz”?
nobu: how about “#{‘baz’}”?
k01: it is easy to understand all “...” is frozen. Shugo-san’s first comment is for “compatibility”.
matz: i agree to

It’s not just for compatibility.
The original problem was that a new String object has to be allocated by a string literal for each evaluation. so i don’t understand the reason why a dynamic string literal should be frozen in spite of the fact freezing dynamic strings can’t reduce object allocation.

It is not a big problem.
We can reduce extra object allocation with “foo#(exp)bar”.dup using an optimization similar for “foo”.freeze.

I think the pragma and option name should explain the behavior.
The name is “frozen-string literal”.
So, basically, all string literal should return a frozen object.

We need an explanation if we introduce an exception.

#69 - 11/05/2015 03:08 AM - kosaki (Motohiro KOSAKI)

Shugo Maeda wrote:

It's not just for compatibility.

The original problem was that a new String object has to be allocated by a string literal for each evaluation. So I don't understand the reason why a dynamic string literal should be frozen in spite of the fact freezing dynamic strings can't reduce object allocation.

It is not a big problem.
We can reduce extra object allocation with "foo#{exp}bar".dup using an optimization similar for "foo".freeze.

I think the pragma and option name should explain the behavior.
The name is "frozen-string-literal".
So, basically, all string literal should return a frozen object.

We need an explanation if we introduce an exception.

+1

Current explanation is too weak to make an exception, I think.

#70 - 11/09/2015 06:35 AM - ko1 (Koichi Sasada)

Discussion: https://docs.google.com/document/d/1D0Eo5N7NE_unlySOKG9iVi_eyXf66BQPM4PKp7NvMyQ/pub

Feel free to add your comments.

#71 - 11/09/2015 08:32 AM - shugo (Shugo Maeda)

- Status changed from Open to Closed

Applied in changeset r52501.

- compile.c (iseq_compile_each): Dynamic string literals should be frozen.
  [ruby-core:57574] [Feature #8976]

#72 - 11/09/2015 09:20 AM - shugo (Shugo Maeda)

Koichi Sasada wrote:

Discussion: https://docs.google.com/document/d/1D0Eo5N7NE_unlySOKG9iVi_eyXf66BQPM4PKp7NvMyQ/pub

Feel free to add your comments.

I didn't mean to claim that "#{exp}".dup needs more object allocation, but to claim that "#{exp}" cannot be interned.
Anyway, I agree with the conclusion and have reverted r52087.