Ruby master - Feature #9278

Magic comment "immutable: string" makes "literal".freeze the default for that file

12/22/2013 09:19 AM - colindkelley (Colin Kelley)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:	2.2.0	

Description

Building on https://bugs.ruby-lang.org/issues/9042, this pull request adds the magic comment # -- *immutable: string* -- that implies .freeze on every string literal in the file. To get a mutable string in a file that starts with the magic comment, use String.new or ''.dup.

Here is a corresponding github pull request:

```
https://github.com/ruby/ruby/pull/487
```

For more details, background, and rationale, please see this blog post:

```
http://development.invoca.com/magic-comment-immutable-string-makes-ruby-2-1s-literal-freeze-optimization-the-default/
```

Related issues:

Is duplicate of Ruby master - Feature #8976: file-scope freeze_string directive	**Closed**

History

#1 - 12/23/2013 08:21 AM - nobu (Nobuyoshi Nakada)

- % Done changed from 100 to 0

#2 - 12/29/2013 08:47 AM - colindkelley (Colin Kelley)

With the 'immutable: string' magic comment, this benchmark runs 1.6X faster (that is, in 61% of the time) compared to stock Ruby 2.1.0.

https://gist.github.com/ColinDKelley/8156708

#3 - 12/29/2013 11:42 PM - headius (Charles Nutter)

Adding .freeze to the frequently-used strings has the same result for me, and it doesn't also gather up other strings that may not need/want to be immutable.

https://gist.github.com/ColinDKelley/8156708

I have the same objections to this magic comment that I did when it was proposed before:

- A magic comment should not completely change the semantics of a literal type. Encoding magic comments do not suffer from the same issue since they only change how the bytes of the strings are interpreted by the encoding subsystem...they do not change semantics.
- A magic comment is far removed from the actual literal strings, meaning that every developer that enters a file will have to keep in mind whether the strings have been forced to be immutable before doing any work with literal strings.
- Although this eliminates or reduces calls to .freeze, it causes the opposite effect to get a mutable string in the same file...specifically, you have to call .dup.

I think it would be better to consider adding the String#f method proposed during the rework of .freeze optimizations. Adding a .f to a literal string is not very much to ask, and in your particular script, it would actually add *fewer* characters to the code than the magic comment.

#4 - 12/30/2013 09:38 AM - colindkelley (Colin Kelley)

Thanks for the quick reply, Charles. Here are my thoughts:

> - A magic comment should not completely change the semantics of a literal type. Encoding magic comments do not suffer from the same issue since they only change how the bytes of the strings are interpreted by the encoding subsystem...they do not change semantics.

I view this immutable: string comment as *less* intrusive than the encoding magic comment. The magic encoding was typically required in order get code to compile at all in 1.9. In this case the magic immutable: string comment is just a tip to Ruby saying "Feel free to optimize string literals here".

The comment is completely optional, as is the optimization. But it's worth it because the optimization has a big speed payoff. 1.6X in the benchmark above. I think most projects will run at least 1.1X faster.

> - A magic comment is far removed from the actual literal strings, meaning that every developer that enters a file will have to keep in mind whether the strings have been forced to be immutable before doing any work with literal strings.

I expect that the immutable: string magic comment would typically be dropped at the top of all files in a project. (As many did with the encoding comment when porting to 1.9.) Any project that has automated test coverage is going to want to put the comment throughout as a project policy, because of the speed payoff. [BTW to make it easy to add the comment, we've cloned the magic_encoding gem into a magic_immutable gem here: https://github.com/Invoca/magic_immutable_string]

> - Although this eliminates or reduces calls to .freeze, it causes the opposite effect to get a mutable string in the same file...specifically, you have to call .dup.

Yes, exactly! This magic comment eliminates a giant/infeasible/ugly change (.freeze after every string literal) and replaces it with a tiny/feasible one.

Consider Rails. I just did a quick/approximate regex search to count string literals in Rails 3.2.12 and found on the order of 50,000 of string literals. How many of those do you think are mutated? I'm going to guess no more than 10 places across all of Rails. (I'm hoping to get some actual stats using a hack of this branch, but for a quick estimate I audited about 150 cases that mutate a string and didn't find a single case where the receiver was a string literal.)

So if we add the magic comment in Rails, we'll need to add .dup (or change to String.new) in up to 10 places instead of adding .freeze after 50,000 string literals!

> I think it would be better to consider adding the String#f method proposed during the rework of .freeze optimizations. Adding a .f to a literal string is not very much to ask, and in your particular script, it would actually add *fewer* characters to the code than the magic comment.

I'm with Paul Graham, who argues that expressiveness is counted in tokens, not bytes. So .f is no more expressive than .freeze. Whether .f or .freeze, it's still 2 tokens to be added after *every* string literal.

One of Ruby's main differentiators is its beauty and English-like readability. That's certainly what drew me to Ruby. No one who feels that way is going to put .freeze or .f after every string literal.

In summary, there will certainly be projects that don't bother with the immutable: string magic comment (for example, those that lack automated test coverage). But many projects will use it because they will get a big, immediate performance payoff in Ruby 2.1. We're already looking forward to it!

#5 - 12/30/2013 09:58 AM - headius (Charles Nutter)

colindkelley (Colin Kelley) wrote:

> > - A magic comment should not completely change the semantics of a literal type. Encoding magic comments do not suffer from the same issue since they only change how the bytes of the strings are interpreted by the encoding subsystem...they do not change semantics.
>
> I view this immutable: string comment as *less* intrusive than the encoding magic comment. The magic encoding was typically required in order get code to compile at all in 1.9. In this case the magic immutable: string comment is just a tip to Ruby saying "Feel free to optimize string literals here". The comment is completely optional, as is the optimization. But it's worth it because the optimization has a big speed payoff. 1.6X in the benchmark above. I think most projects will run at least 1.1X faster.

I'm not trying to draw a comparison with the encoding comment. Encoding specification is necessary for the many encodings that cannot be reliably determined solely from inspecting the first few lines of code.

The concern I have here is that by adding a comment at the top of the file, you change the set of methods you can call on every literal string in the file. That has no parallel in any other feature, and certainly does not apply to encoding magic comment.

I do not believe there should be a file-global pragma that basically makes half of String methods start raising errors on any literals in that file. Being able to declare in a simple way that each string is immutable localizes that semantic change to specific literals.

> > - A magic comment is far removed from the actual literal strings, meaning that every developer that enters a file will have to keep in mind whether the strings have been forced to be immutable before doing any work with literal strings.
>
> I expect that the immutable: string magic comment would typically be dropped at the top of all files in a project. (As many did with the encoding comment when porting to 1.9.) Any project that has automated test coverage is going to want to put the comment throughout as a project policy, because of the speed payoff. [BTW to make it easy to add the comment, we've cloned the magic_encoding gem into a magic_immutable gem here: https://github.com/Invoca/magic_immutable_string]

I don't agree that the primary benefit of immutable strings is the performance gain, but that's largely a matter of opinion. I believe the benefit of immutable strings is guaranteeing very early on that a string is safe to pass across threads and to downstream libraries since it can't be modified.

By your logic, if performance was a good enough reason to apply such a change throughout a project, then all projects should specify and use UTF-32 encoding since it makes String slicing and index operations (heavily used in many libraries) O(1).

I stand by my assertion that having a file-global pragma that changes the behavior -- not just the contents -- of every literal string is a bad idea.

> > - Although this eliminates or reduces calls to .freeze, it causes the opposite effect to get a mutable string in the same file...specifically, you have to call .dup.
>
> Yes, exactly! This magic comment eliminates a giant/infeasible/ugly change (.freeze after every string literal) and replaces it with a tiny/feasible one.
>
> Consider Rails. I just did a quick/approximate regex search to count string literals in Rails 3.2.12 and found on the order of 50,000 of string literals. How many of those do you think are mutated? I'm going to guess no more than 10 places across all of Rails. (I'm hoping to get some actual stats using a hack of this branch, but for a quick estimate I audited about 150 cases that mutate a string and didn't find a single case where the receiver was a string literal.)
>
> So if we add the magic comment in Rails, we'll need to add .dup (or change to String.new) in up to 10 places instead of adding .freeze after 50,000 string literals!

These are not concrete numbers, so I can't really comment.

> > I think it would be better to consider adding the String#f method proposed during the rework of .freeze optimizations. Adding a .f to a literal string is not very much to ask, and in your particular script, it would actually add *fewer* characters to the code than the magic comment.
>
> I'm with Paul Graham, who argues that expressiveness is counted in tokens, not bytes. So .f is no more expressive than .freeze. Whether .f or .freeze, it's still 2 tokens to be added after *every* string literal.

I'm not with Paul Graham :-) I believe expressiveness is very much tied to how much I have to type. Keywords could be arbitrarily long and constitute one token, but it's very hard to make the claim that a 20-character keyword is more expressive than the same keyword in two or three characters.

In any case, .f is hardly intrusive and immediately shows which strings are intended to be immutable and which strings are not.

I fully appreciate that all of these features are attempts to correct the fact that MRI defaults to mutable strings rather than immutable strings. Honestly, if you want to make strings immutable by default, let's make single-quoted strings or a %{} format be immutable by default, rather than adding semantic-changing pragmas.

> One of Ruby's main differentiators is its beauty and English-like readability. That's certainly what drew me to Ruby. No one who feels that way is going to put .freeze or .f after every string literal.

Tell me at a glance whether the strings in this snippit of code are mutable or immutable:

...
def foo
"foo"
end
...

With the magic comment in play, you can't know.

Now do the same for the following code:

def foo
"foo".f
end

I think the latter is more readable because you don't have to look at a part of the file far away from the literal to know what sort of String you're dealing with.

#6 - 12/30/2013 02:16 PM - jacknagel (Jack Nagel)

It seems like it would be logical for the VM to automatically freeze/dedup string literal keys in hash literals and hash lookups. Is there a reason that can't be done?

#7 - 12/30/2013 03:23 PM - normalperson (Eric Wong)

"jacknagel (Jack Nagel)" redmine@ruby-lang.org wrote:

> It seems like it would be logical for the VM to automatically
> freeze/dedup string literal keys in hash literals and hash lookups. Is
> there a reason that can't be done?

That's a goal: https://bugs.ruby-lang.org/issues/8998
Just need the time/effort.

#8 - 12/30/2013 07:18 PM - tmm1 (Aman Gupta)

> > It seems like it would be logical for the VM to automatically
> > freeze/dedup string literal keys in hash literals and hash lookups. Is
> > there a reason that can't be done?
>
> That's a goal: https://bugs.ruby-lang.org/issues/8998
> Just need the time/effort.

I implemented such an optimization for Hash#[]= in https://bugs.ruby-lang.org/issues/9188#note-8

A similar patch could be developed for the aref case, but I'm not sure if the perf improvement would be worth a specialized instruction. In fact, there's not really a performance improvement, but rather less pressure on the GC. That can be beneficial, but it can also be addressed by GC improvements.

As far as a immutable string magic comment, I'm still unconvinced. Perhaps if you show concrete gains in a large code-base like rails the case will be more compelling.

#9 - 12/31/2013 01:28 AM - colindkelley (Colin Kelley)

> Perhaps if you show concrete gains in a large code-base like rails the case will be more compelling.

Working on it! First to see how many tests actually fail from trying to mutate string literals, then to get some general performance numbers.

#10 - 01/30/2014 05:53 AM - hsbt (Hiroshi SHIBATA)

- Target version changed from 2.1.0 to 2.2.0

#11 - 09/27/2015 06:44 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

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]