Ruby - Feature #8992

Use String#freeze and compiler tricks to replace "str"f suffix

10/08/2013 02:56 AM - headius (Charles Nutter)

Status:	Closed	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	2.1.0	

Description

BACKGROUND:

In https://bugs.ruby-lang.org/issues/8579 @charliesome introduced the "f" suffix for creating already-frozen strings. A string like "str"f would have the following characteristics:

- It would be frozen before the expression returned
- It would be the same object everywhere, pulling from a global "fstring" table

To avoid memory leaks, these pooled strings would remove themselves from the "fstring" table on GC.

However, there are problems with this new syntax:

- It will never parse in Ruby 2.0 and earlier.
- It's not particularly attractive, though this is a subjective matter.
- It does not lend itself well to use in other scenarios, such as for arrays and hashes (http://bugs.ruby-lang.org/issues/8909)

PROPOSAL:

I propose that we eliminate the new "f" suffix and just make the compiler smart enough to see literal strings with .frozen the same way.

So this code:

str = "mystring".freeze

Would be equivalent in the compiler to this code:

str = "mystring"f

And the fstring table would still be used to return pooled instances.

IMPLEMENTATION NOTES:

The fstring table already exists on master and would be used for these pooled strings. An open question is whether the compiler should forever optimize "str".frozen to return the pooled version or whether it should check (inline-cache style) whether String#freeze has been replaced. I am ok with either, but the best potential comes from ignoring String#freeze redefinitions...or making it impossible to redefine String#freeze.

BONUS BIKESHEDDING:

If we do not want to overload the existing .freeze method in this way, we could follow suggestions in http://bugs.ruby-lang.org/issues/8977 to add a new "frozen" method (or some other name) that the compiler would understand.

If it were "frozen", the following two lines would be equivalent:

str = "mystring".frozen
str = "mystring"f

In addition, using .frozen on any string would put it in the fstring table and return that pooled version.

I also propose one alternative method name: the unary ~ operator.

There is no ~ on String right now, and it has no meaning for strings that we'd be overriding. So the following two lines would be equivalent:

str = ~"mystring"
str = "mystring"f

JUSTIFICATION:

Making the compiler aware of normal method-based String freezing has the following advantages:

- It will parse in all versions of Ruby.
- It will be equivalent in all versions of Ruby other than the fstring pooling.
- It extends neatly to Array and Hash; the compiler can see Array or Hash with literal elements and return the same object.
- It does not require a pragma (http://bugs.ruby-lang.org/issues/8976)
- It looks like Ruby.

Related issues:

Related to Ruby - Feature #8579: Frozen string syntax	**Closed**	**06/29/2013**
Related to Ruby - Misc #20222: Dedup-ing clarification	**Closed**	

History

#1 - 10/08/2013 03:04 AM - ko1 (Koichi Sasada)

- Category set to core

- Assignee set to matz (Yukihiro Matsumoto)

> If we do not want to overload the existing .freeze method in this way, we could follow suggestions in http://bugs.ruby-lang.org/issues/8977 to add a new "frozen" method (or some other name) that the compiler would understand.

I like this idea. No compatibility issue.

(except naming issue)

#2 - 10/08/2013 03:10 AM - calebthompson (Caleb Thompson)

Optimizing the #freeze or #frozen methods to do this make a lot more sense to me
than the ""f suffix to me.

I'm +1 on #frozen and preventing redefinition of that method, but I further
propose that we raise an error if redefining frozen rather than silently
ignoring it, which would be very surprising.

#3 - 10/08/2013 03:18 AM - Anonymous

+1 for optimized String#freeze as it will work on previous versions of Ruby:

I've written a patch implementing this feature: https://gist.github.com/charliesome/6836600

#4 - 10/08/2013 03:27 AM - headius (Charles Nutter)

=begin
I am leaning toward #frozen if we want a new name... as in "give me the frozen version of this string". I know that there was some concern that "frozen" was too similar to "freeze" in http://bugs.ruby-lang.org/issues/8977 but it still feels like the best name.

If we can't do a new name that refers to freezing, I'd rather just stick with .freeze.

And I still kinda like ~@ :-)

class String
alias ~@ freeze
end

...although there's operator precedence problems (puts ~"str".frozen? does not work).
=end

#5 - 10/08/2013 05:33 AM - Eregon (Benoit Daloze)

I am happy to see another discussion on this, I feel "str"f is just a hack.

I strongly agree and think #freeze is the right name.

On the aesthetics side, I personally dislike prefix/suffix forms, they feel like u'str' in python which just makes me think it does not support the right

strings by default. I kind of like %f{ ... } but #freeze fits even more in Ruby I think, even if a bit long but at least its semantics are clear.

#6 - 10/08/2013 07:59 AM - normalperson (Eric Wong)

"headius (Charles Nutter)" headius@headius.com wrote:

> I propose that we eliminate the new "f" suffix and just make the compiler
> smart enough to see literal strings with .frozen the same way.
>
> So this code:
>
> str = "mystring".freeze
>
> Would be equivalent in the compiler to this code:
>
> str = "mystring"f
>
> And the fstring table would still be used to return pooled instances.

This is a great idea IMHO. The backwards compatibility is a huge win
and I think this is the best idea so far regarding frozen strings.

> IMPLEMENTATION NOTES:
>
> The fstring table already exists on master and would be used for these
> pooled strings. An open question is whether the compiler should
> forever optimize "str".frozen to return the pooled version or whether
> it should check (inline-cache style) whether String#freeze has been
> replaced. I am ok with either, but the best potential comes from
> ignoring String#freeze redefinitions...or making it impossible to
> redefine String#freeze.

Initially (a few minutes ago), I thought it'd be better to inline-cache
to minimize surprise/keep compatibility. And maybe spew a loud warning on
String#freeze redefinition.

But thinking about this more, string literals are already special.
String#initialize is already ignored for string literals, so perhaps
#freeze may be unredefinedable, as well.

> If we do not want to overload the existing .freeze method in this way,
> we could follow suggestions in http://bugs.ruby-lang.org/issues/8977
> to add a new "frozen" method (or some other name) that the compiler
> would understand.

I think having only .freeze is better (especially for compatibility) and
a new .frozen method would be of minimal benefit.
(But you know Ruby far better than I do)

> I also propose one alternative method name: the unary ~ operator.
>
> There is no ~ on String right now, and it has no meaning for strings that we'd be overriding. So the following two lines would be equivalent:
>
> str = ~"mystring"
> str = "mystring"f

I hate this alternative "name". It's too ambiguous/confusing
(consider: ~~"username" or /regexp/~~ ~"foo"),
not backwards compatible, and hard to search for documentation on.

I think a beginner new to the language would be very confused by this.
You have my strong support of #freeze, but my strongest disapproval of ~

> JUSTIFICATION:
>
> Making the compiler aware of normal method-based String freezing has the following advantages:
>
> - It will parse in all versions of Ruby.
> - It will be equivalent in all versions of Ruby other than the fstring pooling.
> - It extends neatly to Array and Hash; the compiler can see Array or Hash with literal elements and return the same object.
> - It does not require a pragma (http://bugs.ruby-lang.org/issues/8976)

- It looks like Ruby.

Amen!

#7 - 10/09/2013 04:24 AM - cibernox (Miguel Camba)

I have another idea What about wrap strings in double backquotes or accents?

"this is an interpolable string"
'this is an uninterpolable string'
this is a frozen string => double backquoute
´this is also a frozen string´ => simple accent

Accents looks elegant, but I don't know if they are cumbersome in some keyboard distributions and I am not sure if they al limited to UTF8 code, but double backquote seems easy to add to the parser and feels pretty natural.

Whatever character(s) you like the most, (string, ´sting´, ^string^, ~~string~~, \string) I like the idea to be used as a wrapper, not adding a special symbol before or after the string definition.

#8 - 10/09/2013 05:35 AM - headius (Charles Nutter)

cibernox (Miguel Camba) wrote:

> I have another idea What about wrap strings in double backquotes or accents?
>
> "this is an interpolable string"
> 'this is an uninterpolable string'
> this is a frozen string => double backquoute

I don't think the parser can distinguish this from normal backquotes that are empty...but there's no value to supporting empty shelling-out backquotes, so maybe it's not hard to special-case it.

if backquotes_contains_content
parse_as_normal_backquote
else
parse_as_frozen_string
end

Or something :-D

> ´this is also a frozen string´ => simple accent

Not on standard US keyboards, so this will never fly.

> Whatever character(s) you like the most, (string, ´sting´, ^string^, ~~string~~, \string) I like the idea to be used as a wrapper, not adding a special symbol before or after the string definition.

Agreed. I still like just adding smarts for .freeze over any of the other options.

#9 - 10/09/2013 06:04 AM - cibernox (Miguel Camba)

Yep, we can discard the accent.

The double pipe can be tricky is you want to set a frozen string as default value of a block argument.
The double ^ has confict with the xor operator.
The double ~ has conflict with the complement operator
The double backslash seems fancy, like an opposite of regex, but since can also be used to break lines and escape characters. And probably is more difficult to parse . You tell me.

I know nothing about the parser internals, but the double backquote feels like a string and seems reasonable easy to implement. Seems the best option.

This idea can just coexist with the .freeze method. Is a bit of syntax sugar.

#10 - 10/09/2013 06:47 AM - Anonymous

I feel like we're getting a bit off topic in this thread.

The main benefit of optimizing String#freeze is maintaining backwards compatibility with older Rubies. Think of it as a kind of 'progressive enhancement'.

Introducing new syntax or a new method completely removes any backwards compatibility benefits and will mean that most Ruby code will not be able to use this syntax for quite some time.

#11 - 10/09/2013 06:56 AM - sam.saffron (Sam Saffron)

I am actually very concerned about compiler tricks with freeze cause it leads to non-obvious code.

```
x = "hello".freeze
y = "hello".freeze
x.object_id
```

```
10
x.object_id == y.object_id
```

```
a = "hello"
a.object_id
```

```
100
a.freeze
a.object_id
100 # must be 100
```

So the way #freeze operates then depends on where it is being executed, I dislike that.

Much prefer just adding #frozen, we can implement it sort of cleanly in 2.0 (except for GC hooking) and simply alias #freeze in 1.9 and earlier.

#12 - 10/09/2013 09:25 AM - headius (Charles Nutter)

sam.saffron (Sam Saffron) wrote:

> I am actually very concerned about compiler tricks with freeze cause it leads to non-obvious code.
>
> ```
> x = "hello".freeze
> y = "hello".freeze
> x.object_id
> ```
>
> ```
> 10
> x.object_id == y.object_id
> ```

I don't think you should *ever* rely on this to be true, since it won't be on older Ruby impls or impls that don't yet have #freeze optimizations. Even Java, with its interned Strings, strongly discourages *ever* using object identity to compare strings. IDEs even flag it as a warning.

> ```
> a = "hello"
> a.object_id
> ```
>
> ```
> 100
> a.freeze
> a.object_id
> 100 # must be 100
> ...
> So the way #freeze operates then depends on where it is being executed, I dislike that.
> ```
>
> Much prefer just adding #frozen, we can implement it sort of cleanly in 2.0 (except for GC hooking) and simply alias #freeze in 1.9 and earlier.

So here's the same question I asked in the #frozen feature: why can't #freeze just use the fstring table?

- fstrings will GC and clear themselves from that table
- large strings put into the table will take up no more space than if they were not frozen

So #freeze *could* do what you suggest here and always use the fstring table. In the "literal".freeze case, the compiler could do additional magic to go to the table immediately.

#13 - 10/09/2013 11:29 AM - normalperson (Eric Wong)

"headius (Charles Nutter)" headius@headius.com wrote:

> So here's the same question I asked in the #frozen feature: why can't
> #freeze just use the fstring table?

That would be an interesting experiment. After all, it is #freeze and not #freeze!, so maybe we have some leverage there.

- fstrings will GC and clear themselves from that table

I think this needs some work for the non-parser case, there seems to be a bad interaction with lazy sweep. My analysis of my failed patch for Feature #8998: http://mid.gmane.org/20131009021547.GA1839@dcvr.yhbt.net

I also get (identical?) segfaults with the following:

```
diff --git a/object.c b/object.c
--- a/object.c
+++ b/object.c
@@ -1029,6 +1029,8 @@ VALUE
rb_obj_freeze(VALUE obj)
{
if (!OBJ_FROZEN(obj)) {
```

- if (TYPE(obj) == T_STRING)
- [redacted]

  ```
  OBJ_FREEZE(obj);
  if (SPECIAL_CONST_P(obj)) {
  if (!immediate_frozen_tbl) {
  ```

#14 - 10/09/2013 12:01 PM - sam.saffron (Sam Saffron)

@hedius

What happens when a string pointer leaks out to a c extension?

#15 - 10/09/2013 02:46 PM - headius (Charles Nutter)

sam.saffron (Sam Saffron) wrote:

> @hedius
>
> What happens when a string pointer leaks out to a c extension?

This question applies equally to "str"f logic. I'm not sure what the answer is, because I don't know how frozen strings in @charliesome's patch interact with C extensions.

I don't think making freeze act like Java's String#intern has any better or worse interaction with C extensions than normal "str"f fstrings interact with C extensions.

#16 - 10/09/2013 02:48 PM - headius (Charles Nutter)

headius (Charles Nutter) wrote:

> This question applies equally to "str"f logic. I'm not sure what the answer is, because I don't know how frozen strings in @charliesome's patch interact with C extensions.

Actually, it occurred to me that the interaction with C extensions is actually even simpler; if a string leaks out to C exts, it's no worse than *any* string leaking out. The only difference is that the C ext would still have a reference while the fstring table does not. So I think it's no worse than current interaction with strings.

#17 - 10/09/2013 02:53 PM - headius (Charles Nutter)

normalperson (Eric Wong) wrote:

> "headius (Charles Nutter)" headius@headius.com wrote:
>
> > So here's the same question I asked in the #frozen feature: why can't
> > #freeze just use the fstring table?
>
> That would be an interesting experiment. After all, it is #freeze and

> not #freeze!, so maybe we have some leverage there.

I think we do. The worst case scenario is that *while referenced* we have more entries in the table, which may include strings that become "shady" and pass out to C exts. But those strings would stay alive under the current definition of "shady" and even under older Ruby versions with a purely conservative GC the effects are no worse.

So basically:

- If the string is long lived normally, it will take up X bytes for its lifetime.
- If the string gets stored in the fstring table, it will last no longer than it would without the fstring table.
- If the string is short-lived, it will have a bit more overhead for dealing with fstring table, but very little; hash calculation and table management at most.

It seems acceptable to have #freeze basically be Java's #intern.

> - fstrings will GC and clear themselves from that table
>
> I think this needs some work for the non-parser case, there seems to
> be a bad interaction with lazy sweep. My analysis of my failed
> patch for Feature #8998:
> http://mid.gmane.org/20131009021547.GA1839@dcvr.yhbt.net

I don't doubt your analysis, but I don't think it's any worse with #freeze using fstring table. It's just multiplied by the number of strings that get frozen. Critical failure * N is still a critical failure.

> I also get (identical?) segfaults with the following:
>
> diff --git a/object.c b/object.c
> --- a/object.c
> +++ b/object.c
> @@ -1029,6 +1029,8 @@ VALUE
> rb_obj_freeze(VALUE obj)
> {
> if (!OBJ_FROZEN(obj)) {
>
> - if (TYPE(obj) == T_STRING)
>
> - [redacted]
>
> OBJ_FREEZE(obj);
> if (SPECIAL_CONST_P(obj)) {
> if (!immediate_frozen_tbl) {

Same argument here, I think.

#18 - 10/14/2013 08:40 AM - sam.saffron (Sam Saffron)

@hedius

There are 3 things being discussed here, I think it is fairly important we split them out.

1. Parser optimisation for "string".freeze
2. Unconditionally have #freeze return a pooled string
3. Change the semantics of #freeze so it amends the current object and operates like .NET / Java intern does.

1. is completely doable with little side-effects. My caveat is that if #1 is the only thing done, the semantics for #freeze depend on the invocation. That said, this is minor. I totally accept that and prefer "string".freeze to "string"f.

2. without 3) really scares me.

Imagine the odd semantics:

a = "hello"
a.freeze # freezes one RVALUE in memory and returns a different RVALUE

As to 3) I don't think it can be implemented in MRI. If an RVALUE is moved in memory, MRI is going to have to crawl the heap and rewrite all the RVALUE that hold a ref to it, it does not keep track of this internally.

@charliesome thoughts?

#19 - 10/14/2013 12:53 PM - nobu (Nobuyoshi Nakada)

(13/10/14 8:40), sam.saffron (Sam Saffron) wrote:

> Issue #8992 has been updated by sam.saffron (Sam Saffron).

Thank you for summarizing.

> There are 3 things being discussed here, I think it is fairly important we split them out.
>
> 1. Parser optimisation for "string".freeze
>
> 1. is completely doable with little side-effects. My caveat is that if #1 is the only thing done, the semantics for #freeze depend on the invocation. That said, this is minor. I totally accept that and prefer "string".freeze to "string"f.

It's a part of byte-code optimization, not parser. Since we have done
it already for several methods, no problem there.

> 1. Unconditionally have #freeze return a pooled string
> 2. Change the semantics of #freeze so it amends the current object and operates like .NET / Java intern does.
>
> 1. without 3) really scares me.
>
> Imagine the odd semantics:
>
> a = "hello"
> a.freeze # freezes one RVALUE in memory and returns a different RVALUE
>
> As to 3) I don't think it can be implemented in MRI. If an RVALUE is moved in memory, MRI is going to have to crawl the heap and rewrite all the RVALUE that hold a ref to it, it does not keep track of this internally.

Totally agree.

2+3 is wrong idea, I think.

#20 - 10/14/2013 03:59 PM - headius (Charles Nutter)

Ok, let's just focus on #1 for now...

It seems like everyone agrees that "string".freeze is a better choice than adding incompatible syntax now. That was the original proposal in this issue.

Should we remove "string"f on master and replace it with charliesome's patch for "string".freeze? Or do we want to bikeshed a shorter name?

It occurred to me the there's already "string".b which returns a binary string. Should we consider "string".f which is similar to "string"f syntax but is just a normal method?

I think we're in agreement that we want the method format rather than the "f" suffix, so it's just a matter of deciding if we want a different method name for the new compiler-aware method.

#21 - 10/14/2013 05:23 PM - phluid61 (Matthew Kerwin)

On Oct 14, 2013 5:00 PM, "headius (Charles Nutter)" headius@headius.com
wrote:

> Ok, let's just focus on #1 for now...
>
> It seems like everyone agrees that "string".freeze is a better choice
> than adding incompatible syntax now. That was the original proposal in this
> issue.
>
> Should we remove "string"f on master and replace it with charliesome's
> patch for "string".freeze? Or do we want to bikeshed a shorter name?
>
> It occurred to me the there's already "string".b which returns a binary
> string. Should we consider "string".f which is similar to "string"f syntax
> but is just a normal method?
>
> I think we're in agreement that we want the method format rather than the
> "f" suffix, so it's just a matter of deciding if we want a different method
> name for the new compiler-aware method.

Yes. I feel like regexen have suffixes because of decades of perl precedence, but they (suffixes) don't belong anywhere else.

For a method, I feel like #freeze is the better name, my only question is: is anyone monkeypatching it (and therefore will be bitten by this optimisation)? I doubt it, but we should still ask. The same question would have to be asked of the new method; I think there's more chance of #f being used in the wild than an overridden #freeze

Also in the favour of #freeze, it gives existing code a boost without any modification.

Sent from my phone, so excuse the typos.

#22 - 10/14/2013 05:29 PM - normalperson (Eric Wong)

Nobuyoshi Nakada nobu@ruby-lang.org wrote:

> (13/10/14 8:40), sam.saffron (Sam Saffron) wrote:
>
> > Issue #8992 has been updated by sam.saffron (Sam Saffron).
>
> Thank you for summarizing.
>
> > There are 3 things being discussed here, I think it is fairly important we split them out.
> >
> > 1. Parser optimisation for "string".freeze
> >
> > 1. is completely doable with little side-effects. My caveat is that if #1 is the only thing done, the semantics for #freeze depend on the invocation. That said, this is minor. I totally accept that and prefer "string".freeze to "string"f.
>
> It's a part of byte-code optimization, not parser. Since we have done it already for several methods, no problem there.

So can we move this optimization to the parser instead? I think Sam means:

```
# optimized literal by parser.  This may use pooled string since
# the string never existed in ObjectSpace before this line of code:
"string".freeze
```

However, if .freeze operates on a variable (and not literal):

```
# leave this optimized by parser to avoid incompatibility
a = "string"
a.freeze => preserve existing behavior
```

> > 1. Unconditionally have #freeze return a pooled string
> > 2. Change the semantics of #freeze so it amends the current object and operates like .NET / Java intern does.
> >
> > 1. without 3) really scares me.
> >
> > Imagine the odd semantics:
> >
> > a = "hello"
> > a.freeze # freezes one RVALUE in memory and returns a different RVALUE

Yes, this scares me if a.freeze made a different RVALUE

> > As to 3) I don't think it can be implemented in MRI. If an RVALUE is moved in memory, MRI is going to have to crawl the heap and rewrite all the RVALUE that hold a ref to it, it does not keep track of this internally.
>
> Totally agree.
>
> 2+3 is wrong idea, I think.

Agreed.

#23 - 10/14/2013 11:46 PM - Anonymous

Totally agree with ssaffron on options 2 and 3 being scary. 3 is also impossible in MRI at the moment.

I'm happy to commit my patch optimizing #freeze on a literal string if it looks good.

I think we should move the discussion of removing f-suffix and perhaps adding String#f to another ticket.

#24 - 10/15/2013 07:57 AM - sam.saffron (Sam Saffron)

+1 for removing f suffix and amending the optimiser for #freeze

#25 - 10/22/2013 09:19 PM - headius (Charles Nutter)

I have added #9042 and #9043 for removing the "f" suffix and adding the #f method, respectively.

I'm starting to lean toward making #f be the only magic form, so nobody can complain that we're adding incompatible syntax ("f" suffix) or changing the semantics of an existing method (#freeze optimization).

#26 - 10/23/2013 05:13 AM - Anonymous

> I'm starting to lean toward making #f be the only magic form, so nobody can complain that we're ... changing the semantics of an existing method (#freeze optimization).

I don't get this argument. Optimized String#freeze doesn't really change semantics in any real way. I'm happy to just ignore anyone that complains about optimizing #freeze on a string literal.

There's only one way this could possibly affect any Ruby code - and that's if the code inspects the object_id of literal strings that it immediately calls #freeze on. I'd say any code that breaks due to this was already fairly brittle anyway.

#27 - 10/23/2013 08:52 PM - Eregon (Benoit Daloze)

I agree having optimized #freeze is better than #f.

#28 - 10/23/2013 09:23 PM - akr (Akira Tanaka)

2013/10/8 headius (Charles Nutter) headius@headius.com:

> Feature #8992: Use String#freeze and compiler tricks to replace "str"f suffix
> https://bugs.ruby-lang.org/issues/8992

Does anyone measure actual performance benefit?

Tanaka Akira

#29 - 10/24/2013 02:23 AM - normalperson (Eric Wong)

Tanaka Akira akr@fsij.org wrote:

> 2013/10/8 headius (Charles Nutter) headius@headius.com:
>
> > Feature #8992: Use String#freeze and compiler tricks to replace "str"f suffix
> > https://bugs.ruby-lang.org/issues/8992
>
> Does anyone measure actual performance benefit?

Not directly, but I wasn't able to come up with performance benefits
from my patch for https://bugs.ruby-lang.org/issues/8998
However, I don't have real apps which depend on hash/string-keys
performance.

Perhaps other improvements (RGenGC) make it less useful.

#30 - 11/02/2013 11:23 AM - ko1 (Koichi Sasada)

Matz, could you conclude this ticket?

I like this idea because:

- No syntax change
- Semantics was changed ("literal".freeze.object_id => anytime same),
  but I can't imagine the apps which rely on this behavior.
- Except 2nd point, this is no compatibility issue.

(2013/10/08 2:56), headius (Charles Nutter) wrote:

> Issue #8992 has been reported by headius (Charles Nutter).
>
> ---
>
> Feature #8992: Use String#freeze and compiler tricks to replace "str"f suffix
> https://bugs.ruby-lang.org/issues/8992

--
// SASADA Koichi at atdot dot net

#31 - 11/02/2013 11:59 AM - matz (Yukihiro Matsumoto)

- I am OK with adding a new method (e.g. String#f) and compiler trick.
- I hesitate a bit to change String#freeze semantics.
- String#~ is very concise and in that sense attractive, but relation of '~' with patterns may cause confusion.

So, my conclusion is adding String#f with compiler trick. I still need some discussion for changing String#freeze.
Note that I don't have strong objection against String#freeze.

Matz.

#32 - 11/02/2013 12:18 PM - mame (Yusuke Endoh)

matz (Yukihiro Matsumoto) wrote:

> - I am OK with adding a new method (e.g. String#f) and compiler trick.

Somewhat oppose against "f" with compiler trick.

"f" is a typical meta-syntactic variable of functions.
I guess some programs have the following code.

```
class String
def f
...
end
end
```

In fact, I've found some instances in a few minutes:

http://blog.goo.ne.jp/ruby-index/e/91995e39fe1fe72d339d882cf5ded055
http://python.g.hatena.ne.jp/mhrs/20060530

--
Yusuke Endoh mame@tsg.ne.jp

#33 - 11/02/2013 12:33 PM - Anonymous

Matz, I believe adding a new method will significantly limit the usefulness of this feature.

The reason optimizing "".freeze is superior to a new syntax is that gems can start using this feature now without dropping support for versions < 2.1.

I think many Gem authors will be very hesitant to monkey patch String#f to String#freeze.

#34 - 11/04/2013 12:28 AM - Anonymous

@mame (Yusuke Endoh): Good catch. How about String#fz then?

#35 - 11/08/2013 04:20 AM - headius (Charles Nutter)

matz: Thank you for weighing in. I think we are on the same page.

There are a few different versions of this proposal getting tossed around, so I'll try to summarize the key points.

1. We would like a compiler trick to replace the "str"f suffix.
2. The compiler trick would apply to literal strings with immediate calls to a freeze method and behave as "str"f does today.
3. The freeze method optimized by the compiler will be #freeze or a new shorter-named method #f or both.
4. Calls to #freeze against non-literals would behave as now.
5. Calls to #f, would use the fstring functionality to return a globally cached string (similar to interning in Java, from #8997)

It sounds like almost everyone is in favor of eliminating the "str"f suffix, so I think we can take that as written.

Decisions to make:

1. Do we add a new method that can do #8977 behavior?
   1a. What do we call it?
2. If we do not add a new method, should compiler tricks also optimize "string".freeze?

I think the following is a good plan:

- If we all agree that "string"f should go away, then we should switch to "string".freeze compiler trick asap. @charliesome provided a patch.
- If we want another method, we should decide on a name and decide whether "string".freeze should still have compiler trick.

#36 - 11/10/2013 05:52 AM - Anonymous

At RubyConf Matz approved optimizing #freeze on static string literals and removing f-suffix.

#37 - 11/10/2013 06:03 AM - Eregon (Benoit Daloze)

charliesome (Charlie Somerville) wrote:

> At RubyConf Matz approved optimizing #freeze on static string literals and removing f-suffix.

Great!

#38 - 11/13/2013 07:20 AM - headius (Charles Nutter)

- Status changed from Open to Closed

This has been completed as of revisions r43627 and r43634.

Thanks for the collaboration, everyone :-)

#39 - 12/10/2013 07:54 PM - Anonymous

sawa: %-strings are already covered by #freeze optimization:

```
λ ./miniruby --dump=insns -e '%{}.freeze'
== disasm: <RubyVM::InstructionSequence:<main>@-e>======================
0000 trace            1                                               (   1)
0002 opt_str_freeze   ""
0004 leave
```

#40 - 12/10/2013 07:55 PM - sawa (Tsuyoshi Sawada)

charliesome (Charlie Somerville) wrote:

> sawa: %-strings are already covered by #freeze optimization:
>
> ```
> λ ./miniruby --dump=insns -e '%{}.freeze'
> == disasm: <RubyVM::InstructionSequence:<main>@-e>======================
> 0000 trace            1                                               (   1)
> 0002 opt_str_freeze   ""
> 0004 leave
> ```

Thanks. I should have checked before. I deleted my previous post.

#41 - 05/13/2019 04:41 PM - bughit (bug hit)

I recently had to remind myself what kind of magic is hiding behind 'str'.freeze and would like to provide some feedback.

While tactically this may have been justified, strategically this is a wrong direction. This approach moves the language towards a collection of magical spells. You end up with code that can't be understood through decomposition into primitives.

'str'.method1 - means construct a string object with a value 'str' and call method 'method1' on it.

The notion that a method call should have time travel powers and be able to influence how its already constructed receiver is allocated is absurd. It

makes mockery of such fundamental concepts as expression evaluation order and method invocation. Perhaps this is already being done elsewhere, not sure, but that's not a justification, doubling down on nonsense does not make it sense.

Whereas special syntax (whatever it is) does not have this flaw. You get to define what new syntax means without corrupting existing concepts, and in the long run this is more valuable than helping gem authors transition to immutable strings.

#42 - 02/09/2024 11:23 AM - byroot (Jean Boussier)

- Related to Misc #20222: Dedup-ing clarification added