Ruby master - Feature #15836

[Proposal] Make Module#name and Symbol#to_s return their internal fstrings

05/07/2019 02:33 PM - byroot (Jean Boussier)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

Why ?

In many codebases, especially Rails apps, these two methods are the source of quite a lot of object allocations.

Module#name is often accessed for various introspection features, autoloading etc.

Symbol#to_s is access a lot by HashWithIndifferentAccess other various APIs accepting both symbols and strings.

Returning fstrings for both of these methods could significantly reduce allocations, as well as sligthly reduce retention as it would reduce some duplications.

Also, more and more Ruby APIs are now returning fstrings. frozen_string_literalAFAIK should become the default some day, string used as hash keys are now automatically interned as well.

Backward compatibilty

Of course this is not fully backward compatible, it's inevitable that some code in the wild is mutating the strings returned by these methods, but I do believe it's a rare occurence, and easy to fix.

Implementation

I implemented it here: https://github.com/ruby/ruby/pull/2175

History

#1 - 05/07/2019 02:42 PM - mame (Yusuke Endoh)

- Status changed from Open to Feedback

Could you show us a benchmark, especially non-micro one? I believe that it is definitely required to discuss this proposal.

#2 - 05/07/2019 02:54 PM - Hanmac (Hans Mackowiak)

is the String returned by rb_sym2str not always frozen?

how is the GC handling in case the Symbol was a dynamic generated one?
wouldn't that remove the string that got returned from rb_sym2str too?

#3 - 05/07/2019 03:36 PM - chrisseaton (Chris Seaton)

> how is the GC handling in case the Symbol was a dynamic generated one?

It copies it - strings and symbols aren't the same thing so it couldn't jus return the symbol again.

#4 - 05/07/2019 03:46 PM - byroot (Jean Boussier)

mame (Yusuke Endoh) very good point. I'll try to run our app against that patch tomorrow.

#5 - 05/07/2019 04:47 PM - shevegen (Robert A. Heiler)

> frozen_string_literalAFAIK should become the default some day

Matz said that but it will not be for ruby 3.0 at the least.

In my own code bases I am using frozen strings a lot, through the shebang; either "# frozen_string_literal: false" (initially) but these days more and more "# frozen_string_literal: true". I think other ruby users may be able to transition into frozen strings if enough time is given AND recommendations are given from the ruby core team in due time whenever there are (future) changes.

> string used as hash keys are now automatically interned as well.

Although the use case for HashWithIndifferentAccess (I hate how long that name is ...) is probably not completely void, with strings being frozen it appears to me as if one use case (the speed factor) is nullified. There may be still other use cases probably, such as API design e. g. when people have to make a decision between "do I have to use a String or a Symbol here". Personally I like both strings and symbols, though; I think jeremy evans once gave a good explanation or wrote documentation to emphasis the distinction in the official doc (but I may misremember).

#6 - 05/07/2019 11:05 PM - byroot (Jean Boussier)

> Matz said that but it will not be for ruby 3.0 at the least.

I assumed it was due for 3.0, but good to know it isn't.

> I think other ruby users may be able to transition into frozen strings if enough time is given

I contribute to many gems, and from what I can see # frozen_string_literal: true is extremely common. A good part is likely due to rubocop enforcing it, another is likely due to various article about how freezing strings made many codebases faster (sometimes oversold but that's another topic).

> with strings being frozen it appears to me as if one use case (the speed factor) is nullified

I'm not 100% sure I understood your point correctly. What I meant by hash keys being frozen is:

```
hash = {}
string = :foo.to_s # One string allocated here
hash[string] = true # A second string is allocated here because `Hash#[]=` apply: `-string.dup`
```

If Symbol#to_s was to return it's internal fstring, the above snippet would save 2 string allocations.

#7 - 05/08/2019 03:15 AM - marcandre (Marc-Andre Lafortune)

Is HashWithIndifferentAccess the main rationale behind this request?

I have doubts about the usefulness of HashWithIndifferentAccess today, now that Rails has protected parameters.

Moreover, now that symbols are garbage collected, shouldn't its implementation use symbols for keys instead of strings?

#8 - 05/08/2019 08:28 AM - byroot (Jean Boussier)

> Is HashWithIndifferentAccess the main rationale behind this request?

No. It's simply the poster child of how common Symbol#to_s is in code bases.

I shouldn't have mentioned HashWithIndifferentAccess because clearly lots of people have a feud with it, and now it's totally shifting the conversation.

What the proposal is actually about

The question here, is wether Module#name and Symbol#to_s should return a new string on every call.

My own understanding of why it's like this is because historically all strings were mutable, so the way to prevent them to be mutated was to duplicate them.

But now that frozen strings are very common in code bases, and that fstring are a thing, IMHO getting a new string on every call is what is surprising.

#9 - 05/08/2019 11:30 AM - byroot (Jean Boussier)

mame (Yusuke Endoh) re benchmark

So I decided to run this against redmine boot, using this branch: https://github.com/redmine/redmine/compare/master...byroot:boot-benchmark

Eager loading is enabled so that the entire codebase is loaded, and it uses https://github.com/SamSaffron/memory_profiler to measure allocations and retentions.

Full benchmark output: https://gist.github.com/byroot/845a5877c1cde91c50b43be446dfb20f

Baseline (official 2.6.3):

```
Total allocated: 121.11 MB (1234362 objects)
Total retained:  24.86 MB (200539 objects)

allocated memory by class
-----------------------------------
  63.36 MB  String

allocated objects by class
-----------------------------------
    980623  String
```

With the patch (official 2.6.3 + this patch):

```
Total allocated: 120.01 MB (1206699 objects)
Total retained:  24.82 MB (199397 objects)

allocated memory by class
-----------------------------------
  62.25 MB  String


allocated objects by class
-----------------------------------
    952953  String
```

Diff:

```
-27 663 allocations (-2.24%)
-1.10MB allocations (-0.9%)
-1 142 retentions   (-0.57%)
-0.4MB retentions   (-0.16%)
```

IMHO that is significant, especially for a small sized application like Redmine. However I can't say wether it outweigh the backward compatibility concern or not.

Backward compatibility

One thing to note is that I had to patch https://github.com/rails/rails/blob/28aca474d48b6acdbe8c7861d9347e27c65fafd9/activesupport/lib/active_support/ordered_options.rb#L43 because it was mutating the result of Symbol#to_s. Also running the Redmine test suite shows a couple breakage in the i18n gem.

IMHO these are fairly simple to fix, but I would totally understand if that was considered as a no-go.

Typical code benefiting from this change

- Rails autoloader and Zeitwerk would both benefit from the Module#name change as they both keep references to class names as hash keys
- Various parts of Rails would benefit as well since they use the class names extensively to derive other class names, as well as symbols belongs_to :post.
- def method_missing very often call name.to_s to match the method name, hence would benefit from the Symbol#to_s as well.
- Serialization of symbols into various formats, e.g. {foo: 42}.to_json. That pattern is fairly common IMO.

#10 - 05/08/2019 02:21 PM - ahorek (Pavel Rosický)

byroot (Jean Boussier) thanks for sharing the benchmark!

IMO module names should be frozen

even if it's easy to fix, this change could definitely break existing apps that depend on it, see https://github.com/jruby/jruby/issues/5229

#11 - 06/13/2019 06:05 AM - duerst (Martin Dürst)

Isn't one main purpose of converting a Symbol to a String that you want to change the symbol string? This proposal would make that use case more tedious.

#12 - 06/13/2019 06:12 AM - matz (Yukihiro Matsumoto)

- Status changed from Feedback to Rejected

The compatibility breakage from changing those methods (especially Symbol#to_s) is too big. Sorry.
Maybe we should work on HashWithIndifferentAccess to improve (memory) performance.

Matz.