Alias `String#-@` as `String#dedup`

02/21/2022 10:56 AM - byroot (Jean Boussier)

Status: Open
Priority: Normal
Assignee: 
Target version: 

Description
This is a rescoped feature request for https://bugs.ruby-lang.org/issues/16295

Rationale

Unary operator have some precedence oddities (headius (Charles Nutter))

This often force to use parentheses, which is awkward and breaks the chaining flow.

It's really not obvious what it does. I submitted many pull requests to various open source projects to reduce their memory footprint, and I am constantly asked what it does and I have to point to the String#-@ documentation. The last example was 3 days ago.

I believe that String#dedup would help users discover this feature, and in projects where 3.2 is the oldest supported version, it would allow for much clearer code.

Proposal

It's all in the title: Alias String#-@ as String#dedup.

Or maybe even rename String#-@ as String#dedup, and make String#-@ the alias?

Related issues:
Related to Ruby master - Feature #16295: Chainable aliases for String#-@ and ...

Closed

History

#1 - 02/21/2022 10:57 AM - byroot (Jean Boussier)
- Related to Feature #16295: Chainable aliases for String#-@ and String#+@ added

#2 - 02/21/2022 11:18 AM - byroot (Jean Boussier)
Proposed patch: https://github.com/ruby/ruby/pull/5583

#3 - 02/21/2022 12:09 PM - Eregon (Benoit Daloze)
:+1:, dedup seems an helpful name and it is chainable.

Also if this is ever called from some other language, e.g., C extensions, then rb_funcall(str, "dedup") is much cleaner than rb_funcall(str, "-@").

#4 - 02/21/2022 12:11 PM - byroot (Jean Boussier)

Well, there's VALUE rb_str_to_interned_str(VALUE str); now, so rb_funcall(str, "-@") is only to support older rubies.

#5 - 02/21/2022 12:15 PM - Eregon (Benoit Daloze)

Right. It applies to other languages too so I could imagine e.g. Python or JS calling dedup on a Ruby String, but calling -@ would be awkward/unclear.

#6 - 02/21/2022 07:06 PM - danh337 (Dan H)

[Copied from the previous open issue https://bugs.ruby-lang.org/issues/16295 ]

Eregon (Benoit Daloze) wrote in #note-19:

.dup is not quite as good, as it always allocates a copy.

It creates a new String instance, which is what one needs to guarantee safe mutation without affecting other parts of the program.
Hence, \+@ should rarely be used (only if we know where all the strings passed to this method come from and it's OK to mutate them). .dup does not copy the actual bytes until mutated, because Strings are copy-on-write.

how is \-@ different from .freeze? The meaning of these seems very much the same.

\-@ interns and might return a different String instance, .freeze does not intern and always returns the receiver.

I see you are talking about the internal workings of the code for these, but the semantics is more important.

If I want to use a method name to have the same effect as "foo" I do "foo".freeze. There isn't really another way and no other way is needed.

If I want to use a method name for +"foo" you say to use "foo".dup, but semantically that doesn't work well.

Does anybody else say .dup is the best alternative to \-@? I'm sorry I do not agree.

Is adding a .dedup method when we have .freeze really the final decision here? I guess if I'm the only objection then so be it.

And it seems odd to just close this when there are some open questions.

#7 - 02/21/2022 07:20 PM - Eregon (Benoit Daloze)

danh337 (Dan H) \-@ and the proposed dedup intern/deduplicate.

This is the main feature of those methods and it is very much part of the semantics (as the docs say).

It's the whole point of these methods really, to reduce the number of duplicate strings and reduce memory usage (which byroot (Jean Boussier) and others successfully used in many gems).

freeze does not intern/deduplicate. That has the advantage it's faster, but it doesn't help memory footprint if there are many duplicates of the same string.

Regarding \+@/dup feel free to continue discussing that on #16295, this issue should remain focused on dedup, that is the purpose of the new issue.

#8 - 02/21/2022 07:32 PM - dannh337 (Dan H)

That original ticket #16295 was about both \@+ and \@- methods. You're addressing \@- here which is great. I see what you mean about the difference from .freeze, however subtle. If there's something really more valuable in a .dedup method than in .freeze, then why couldn't .freeze do that more valuable thing?

The \@+ is still not resolved, even though #16295 is closed. If the behavior of \@+ and .dup is the same, that's fine and I get your point, but x = "".dup is semantically weird. I realize that x = +"" is probably what most would use anyway, but for method chains that mutate a String, where I know the String is expected to mutate, a "fast" inverse of .freeze would be nice instead of always .dup. If a String is already mutable, we do not want to duplicate it.

#9 - 02/21/2022 08:07 PM - byroot (Jean Boussier)

Then why couldn't .freeze do that more valuable thing?

Because it's not free, it incurs an additional hash lookup, and may grow the interned string hash. So in many cases where you know that deduping isn't necessary, but you do want to freeze a string, it would slow things down.

The \@+ is still not resolved, even though #16295 is closed.

I closed it because I was the feature requester and after the two years of discussion I no longer believe \@+ needs a chainable alias. If you think it does, it might be best to open a distinct, very focused feature request. It's usually better that way anyway, as the smaller the feature request is, the easier it is to make a point.

#10 - 02/21/2022 08:21 PM - dannh337 (Dan H)

byroot (Jean Boussier) wrote in #note-9:

Then why couldn't .freeze do that more valuable thing?

Because it's not free, it incurs an additional hash lookup, and may grow the interned string hash. So in many cases where you know that deduping isn't necessary, but you do want to freeze a string, it would slow things down.

Ok. So if "blah" does do interning then that makes more sense. However I'm just not sure I could be in a situation where freezing a String is not also interning it. There is probably a good example but I can't think of it. Even if I'm in a loop, processing many mutable Strings, and I want to make them all unmutable, it seems worth the cost to do the intern. I confess I mistakenly thought .freeze on a String did that anyway.
The .@+ is still not resolved, even though #16295 is closed.

I closed it because I was the feature requester and after the two years of discussion I no longer believe +@ needs a chainable alias. If you think it does, it might be best to open a distinct, very focused feature request. It's usually better that way anyway, as the smaller the feature request is, the easier it is to make a point.

Ok fair enough. Thanks.

#11 - 02/21/2022 08:25 PM - byroot (Jean Boussier)

So if "blah" does do interning

Quick sidenote, it's best not to use string literals for examples of this feature, because they behave a bit differently in this context.

it seems worth the cost to do the intern

The only point of interning is to reduce the memory footprint of the app, so it's only worth doing it for data that is "constant". You may want to freeze a string to safely pass it around even though you know it won't survive the current request cycle or whatever your unit of work is.

#12 - 02/21/2022 08:55 PM - danh337 (Dan H)

byroot (Jean Boussier) wrote in #note-11:

So if "blah" does do interning

Quick sidenote, it's best not to use string literals for examples of this feature, because they behave a bit differently in this context.

No worries. That was my example because this ticket is to give it a name. As in "blah" will truly be equivalent to "blah".dedup. If that's not the case then this is still too subtle and many Ruby devs are gonna need help with it.

it seems worth the cost to do the intern

The only point of interning is to reduce the memory footprint of the app, so it's only worth doing it for data that is "constant". You may want to freeze a string to safely pass it around even though you know it won't survive the current request cycle or whatever your unit of work is.

Ok. It sounds like you're saying this is bad:

CONSTANT_A = "bing".freeze
CONSTANT_B = "bing".freeze
CONSTANT_C = "bang".freeze

Because the first 2 constants actually create 2 objects instead of the single equivalent interned object.

And it becomes good if I change those .freeze calls to either .@ or .dedup, because that memory on the 2nd constant is saved.

And all Ruby devs should use .freeze when: (a) we want it to run faster, and (b) we don't care about possible extra memory usage.

If that is a fair picture then it might finally make sense to me. And incidentally, if that's a fair picture, in my own code I probably would almost never use .freeze any more. (Again, I have been operating for a long time thinking that .freeze on Strings already did internning.)

#13 - 02/21/2022 10:01 PM - byroot (Jean Boussier)

Because the first 2 constants actually create 2 objects instead of the single equivalent interned object.

Well, that's why I suggested not to use literals, because in your example the first two constants will point to the same object because .freeze has specific support in the parser (it's equivalent to enabling frozen_string_literal)

So yes "string_literal".freeze does intern, but "foo".upcase.freeze doesn't.

should use .freeze when: (a) we want it to run faster, and (b) we don't care about possible extra memory usage.
Pretty much yes. I'd add: (c) when we know the string is most likely unique.