Ruby trunk - Feature #15393

Add compilation flags to freeze Array and Hash literals

12/08/2018 12:42 AM - tenderlovemaking (Aaron Patterson)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Hi,

I would like to add VM compilation options to freeze array and hash literals. For example:

```
frozen = RubyVM::InstructionSequence.compile(<<-eocode, __FILE__, nil, 0, frozen_string_literal:
true, frozen_hash_and_array_literal: true)
  { 'a' => ['b', { 'c' => 'd' }] }
eocode
puts frozen.disasm
```

Output is:

```
$ ./ruby thing.rb
== disasm: #<ISeq:<compiled>@thing.rb:0 (0,0)-(0,34)> (catch: FALSE)
0000 putobject                  {"a"=>["b", {"c"=>"d"}]}
0002 leave
```

Anything nested in the hash that can't be "frozen" will cause it to not be frozen.

For example:

```
not_frozen = RubyVM::InstructionSequence.compile(<<-eocode, __FILE__, nil, 0,
frozen_string_literal: true, frozen_hash_and_array_literal: true)
  { 'a' => some_method }
eocode
puts not_frozen.disasm
```

Output:

```
$ ./ruby thing.rb
== disasm: #<ISeq:<compiled>@thing.rb:0 (0,0)-(0,24)> (catch: FALSE)
0000 putobject                  "a"
0002 putself
0003 opt_send_without_block     <callinfo!mid:some_method, argc:0, FCALL|VCALL|ARGS_SIMPLE>, <ca
llcache>
0006 newhash                    2
0008 leave
```

Eventually I would like to freeze array and hash literals in source code itself, but I think this is a good first step.

The reason I want this feature is I think we can reduce some object allocations, and once Guilds are implemented, easily create immutable data.

I've attached a patch that implements the above.

(Also I think maybe "frozen_literals" would be a better name, but I don't want to imply that numbers or booleans are frozen too)

Thanks!

History

#1 - 12/08/2018 01:58 PM - shevegen (Robert A. Heiler)

About the name - some suggestions if only to show which variants may be

better than others. :)

frozen: true # could say "ruby, freeze as much as you possibly can!"

In other words, that setting/flag could mean that strings, hashes and arrays are frozen (if the above is approved).

Or:

```
frozen_string_array_hash_literals: true
frozen_string_hash_array_literals: true
frozen_three_literals: true
```

I am not saying any of these names are good names. Just putting them down. :)

```
frozen_literals: true
```

I think one advantage for frozen_literals is that it is short. People could use that in .rb files too.

Another approach could of course also be to only allow strings to be frozen (which will become the default in ruby 3.x if I remember correctly); and not allow a change of hash and arrays in a .rb file, but to allow it for the suggestion above, e. g. in RubyVM.

One could also split the above up into hash and array as separate calls, so rather than:

```
RubyVM::InstructionSequence.compile(<<-eocode, __FILE__, nil, 0, frozen_string_literal: true, frozen_hash_and_array_literal: true)
```

to have:

```
RubyVM::InstructionSequence.compile(<<-eocode, __FILE__, nil, 0, frozen_string_literal: true, frozen_hash_literal: true, frozen_array_literal: true)
```

I mention the last primarily because we already have frozen string literals; so for consistency it may be useful to have it split up into hash and arrays too. If this is too cumbersome to use three separate calls, then one could always use keys that unify these, such as in the example given by Aaron:

```
frozen_hash_and_array_literal: true
```

could also include Strings into it:

```
frozen_string_and_hash_and_array_literal: true
```

It's a bit cumbersome though. Perhaps a simple variant may then be:

```
frozen: true
frozen_literal: true
frozen_literals: true
```

This could then mean to "freeze all that can be frozen".

Anyway; I think the first step is to ask matz about the functionality itself, and then see which API may be best if this is approved.

Personally I think "frozen_literals" would be ok, even if the name may not be 100% "perfect". Ruby is not necessarily perfect in all names e. g. I remember a few folks thinking that the word constant implies "can not be changed", which may be the accurate meaning, but from within ruby itself, I think it is perfectly fine to let people change constants if they want to (e. g. the philosophy of ruby to be practical and useful).

#2 - 12/08/2018 05:58 PM - Eregon (Benoit Daloze)

One alternative idea which would achieve a similar goal is having #deep_freeze on core types, and recognizing { 'a' => ['b', { 'c' => 'd' }] }.deep_freeze. This would be more general, since it would also work for freezing non-constant/non-literal data structures.

I would think many [] and {} are meant to be mutated, so it seems frozen Array/Hash literals them would rather be the exception than the norm (which is less clear for Strings, so I think there the magic comment makes sense).

#3 - 12/10/2018 12:55 AM - shyouhei (Shyouhei Urabe)

Off topic:

Eregon (Benoit Daloze) wrote:

> One alternative idea which would achieve a similar goal is having #deep_freeze on core types, and recognizing { 'a' => ['b', { 'c' => 'd' }] }.deep_freeze.
> This would be more general, since it would also work for freezing non-constant/non-literal data structures.

Object#deep_freeze can be written without any extensions right now, like this:
https://github.com/shyouhei/optdown/blob/master/lib/optdown/deeply_frozen.rb
I heard this trick from akr (Akira Tanaka) .

#4 - 12/10/2018 01:13 PM - Eregon (Benoit Daloze)

On Mon, Dec 10, 2018 at 1:55 AM shyouhei@ruby-lang.org wrote:

> Off topic:
>
> Eregon (Benoit Daloze) wrote:
>
> > One alternative idea which would achieve a similar goal is having #deep_freeze on core types, and recognizing { 'a' => ['b', { 'c' => 'd' }] }.deep_freeze.
> > This would be more general, since it would also work for freezing non-constant/non-literal data structures.
>
> Object#deep_freeze can be written without any extensions right now, like this:
> https://github.com/shyouhei/optdown/blob/master/lib/optdown/deeply_frozen.rb
> I heard this trick from akr (Akira Tanaka) .

Interesting code :)
But also inefficient of course, making extra copies (changing identity) and allocations.

#5 - 12/10/2018 08:10 PM - tenderlovemaking (Aaron Patterson)

Eregon (Benoit Daloze) wrote:

> One alternative idea which would achieve a similar goal is having #deep_freeze on core types, and recognizing { 'a' => ['b', { 'c' => 'd' }] }.deep_freeze.
> This would be more general, since it would also work for freezing non-constant/non-literal data structures.

I thought about doing this with ".freeze", introducing a special instruction the same way we do for the "string".freeze optimization, but dealing with deoptization in the case someone monkey patches the method seems like a pain.

The reason I went this direction first is that it side steps the deoptimization problem, and if we want to support a ".deep_freeze" method later we can. I imagine the implementation would be a branch where one of the branches is the same putobject instruction that I have in this patch.

Anyway, I think the advantages of this patch are:

1. No new methods like "deep_freeze", so you can try this with existing code
2. Any code you write that works with these flags enabled will also work with them disabled (vs deep_freeze that isn't on existing Rubys)
3. This patch should be forward compatible when we figure out what "deep_freeze" solution we're going to have in the future

> I would think many [] and {} are meant to be mutated, so it seems frozen Array/Hash literals them would rather be the exception than the norm (which is less clear for Strings, so I think there the magic comment makes sense).

I thought the same thing, but I'm not 100% convinced. Lots of code in Rails (as well as our application at work) is merging some kind of "default hash" like:

```
def method(some_hash)
  some_hash = { :defaults => 'thing' }.merge(some_hash)
end
```

I'll try to get some numbers on this though. :D

#6 - 12/12/2018 09:35 PM - shan (Shannon Skipper)

I had the same thought as shevy, that it'd be nice to have a:

```
# frozen_literals: true
```

It might also be worth considering a separate frozen_array_literal and frozen_hash_literal. I have files where I'd initially only want to enable one of the above.

#7 - 12/13/2018 08:47 AM - janfri (Jan Friedrich)

shan (Shannon Skipper) wrote:

> I had the same thought as shevy, that it'd be nice to have a:
>
> ```
> # frozen_literals: true
> ```
>
> It might also be worth considering a separate frozen_array_literal and frozen_hash_literal. I have files where I'd initially only want to enable one of the above.

+1

#8 - 12/14/2018 07:51 PM - devpolish (Nardo Nykolyszyn)

I've been waiting this for a while. I'm completely agree.

#9 - 12/15/2018 01:27 PM - Eregon (Benoit Daloze)

tenderlovemaking (Aaron Patterson) wrote:

> I thought about doing this with ".freeze", introducing a special instruction the same way we do for the "string".freeze optimization, but dealing with deoptization in the case someone monkey patches the method seems like a pain.

I think it might semantically make some sense to ignore monkey-patching of .freeze (and .deep_freeze) for calls on literals, which would then avoid needing deoptimization for this (although deoptimization would be a flag check + the current logic as fallback, it doesn't seem so bad).
It's somewhat similar to String literals not calling String#initialize for instance (same for Array, Hash).
And it's probably a bad idea to override #freeze in the hope it would be used on frozen literals anyway, as of course this would only work if the monkey-patch is reliably loaded before everything else.

Another issue is it's quite ugly to opt out of frozen array/hash literals, if a mutable copy is wanted:

```
# frozen_hash_and_array_literal: true
my_mutable_data = { 'a' => ['b', { 'c' => 'd' }.dup].dup }.dup
```

That's why I think deep_freeze would better express the intent in some cases, and be finer-grained:

```
MY_CONSTANT = { 'a' => ['b', { 'c' => 'd' }] }.deep_freeze
my_mutable_data = { 'a' => ['b', { 'c' => 'd' }] }

{ :defaults => 'thing' }.deep_freeze.merge(some_hash)
```

and this would work regardless of what is the value to freeze (but only avoid allocations if it's all literals, otherwise a constant must be used).

Furthermore, frozen literals don't allow composition or extraction in different constants:

```
# frozen_hash_and_array_literal: true
MY_KEY = 'a'
MY_CONSTANT = { MY_KEY => ['b', { 'c' => 'd' }] } # Not frozen, and breaks referential transparency

MY_CONSTANT = { MY_KEY => ['b', { 'c' => 'd' }] }.deep_freeze # Works
```

OTOH, "string".freeze has shown the magic comment is much nicer in many cases than adding "string".freeze in many places (at the price of making a mutable String not so nice, but those seem rarer).
It would be interesting to get an idea of what's a typical ratio of immutable/mutable Array and Hash literals, and what converting a codebase to use frozen Array/Hash literals would look like.

#10 - 12/18/2018 05:08 PM - Anonymous

- File frequency-2.rb added

- File frequency-1.rb added

#11 - 12/18/2018 05:13 PM - Anonymous

- File deleted (frequency-2.rb)

- File deleted (frequency-1.rb)

- File deleted (0001-Add-compile-options-for-freezing-hash-and-array-lite.patch)

#12 - 12/18/2018 05:14 PM - Anonymous

- File frequency-2.rb added

- File frequency-1.rb added

I would like to add VM compilation options to freeze array and hash literals. For example:

My two samples:

```
#!/usr/local/bin/ruby -w

def frequency(text)
  def prepare_string(text)
    text.gsub!(",", "")
        .split(" ")                                #
  end                                              # <-- freeze the result and return it
                                                   #
  def sort_frequency_report(fq)
    fq.to_h
      .sort_by{|k, v| v }
      .reverse
      .to_h                                        #
  end                                              # <-- freeze the result and return it
                                                   #
  word_list = prepare_string(text)

  fq = Hash.new()

  for word in word_list
    if fq.has_key?(word)
      fq[word] += 1
    else
      fq[word] = 1
    end
  end
                                                   #
  sort_frequency_report(fq)                        # <-- freeze the result and return it
end                                                #

p frequency("text text text text,
             text txet text text,
             text text text")
```

And:

```
#!/usr/local/bin/ruby -w

def frequency(text)
  def prepare_string(text)
    text.gsub!(",", "")
        .split(" ")
  end

  def sort_frequency_report(fq)
    #                                              #
    # fq = Hash.new()                              # Why?!.. is it correct?..
    #                                              #

    fq.to_h
      .sort_by{|k, v| v }
      .reverse
      .to_h
  end

  word_list = prepare_string(text)

  fq = Hash.new()

  for word in word_list
    if fq.has_key?(word)
      fq[word] += 1
```

```
    else
      fq[word] = 1
    end
  end

  sort_frequency_report(fq)
end

p frequency("text text text text,
             text txet text text,
             text text text")
```

#13 - 12/18/2018 05:20 PM - Anonymous

- File deleted (frequency-2.rb)

- File deleted (frequency-1.rb)

- File frequency-2.rb added

- File frequency-1.rb added

#14 - 12/18/2018 05:21 PM - Anonymous

- File deleted (frequency-2.rb)

- File deleted (frequency-1.rb)

#15 - 01/15/2019 12:15 AM - tenderlovemaking (Aaron Patterson)

Eregon (Benoit Daloze) wrote:

> tenderlovemaking (Aaron Patterson) wrote:
>
> > I thought about doing this with ".freeze", introducing a special instruction the same way we do for the "string".freeze optimization, but dealing with deoptization in the case someone monkey patches the method seems like a pain.
>
> I think it might semantically make some sense to ignore monkey-patching of .freeze (and .deep_freeze) for calls on literals, which would then avoid needing deoptimization for this (although deoptimization would be a flag check + the current logic as fallback, it doesn't seem so bad).
> It's somewhat similar to String literals not calling String#initialize for instance (same for Array, Hash).
> And it's probably a bad idea to override #freeze in the hope it would be used on frozen literals anyway, as of course this would only work if the monkey-patch is reliably loaded before everything else.
>
> Another issue is it's quite ugly to opt out of frozen array/hash literals, if a mutable copy is wanted:
>
> ```
> # frozen_hash_and_array_literal: true
> my_mutable_data = { 'a' => ['b', { 'c' => 'd' }.dup].dup }.dup
> ```
>
> That's why I think deep_freeze would better express the intent in some cases, and be finer-grained:
>
> ```
> MY_CONSTANT = { 'a' => ['b', { 'c' => 'd' }] }.deep_freeze
> my_mutable_data = { 'a' => ['b', { 'c' => 'd' }] }
>
> { :defaults => 'thing' }.deep_freeze.merge(some_hash)
> ```

Yes, I totally agree. I think deep_freeze is going to be necessary for adoption of Guilds, and the code I'm proposing in this patch would be an optimization of the deep_freeze call on a literal.

> and this would work regardless of what is the value to freeze (but only avoid allocations if it's all literals, otherwise a constant must be used).
>
> Furthermore, frozen literals don't allow composition or extraction in different constants:
>
> ```
> # frozen_hash_and_array_literal: true
> MY_KEY = 'a'
> MY_CONSTANT = { MY_KEY => ['b', { 'c' => 'd' }] } # Not frozen, and breaks referential transparency
>
> MY_CONSTANT = { MY_KEY => ['b', { 'c' => 'd' }] }.deep_freeze # Works
> ```
>
> OTOH, "string".freeze has shown the magic comment is much nicer in many cases than adding "string".freeze in many places (at the price of making a mutable String not so nice, but those seem rarer).
> It would be interesting to get an idea of what's a typical ratio of immutable/mutable Array and Hash literals, and what converting a codebase to use frozen Array/Hash literals would look like.

Right. I'm not proposing adding the magic comment at all, just adding a parameter to the ISeq constructor that will allow you to "deep freeze" literals

in the code passed to ISeq#new.