Ruby master - Feature #16600

Optimized opcodes for frozen arrays and hashes literals

01/29/2020 04:59 PM - byroot (Jean Boussier)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

Context

A somewhat common pattern when trying to optimize a tight loop is to avoid allocations from some regular idioms such as a parameter default value being an empty hash or array, e.g.

```
def some_hotspot(foo, options = {})
  # ...
end
```

Is often translated in:

```
EMPTY_HASH = {}.freeze
private_constant :EMPTY_HASH
def some_hotspot(foo, options = EMPTY_HASH)
  # ...
end
```

But there are many variations, such as (something || []).each .., etc.

A search for that pattern on GitHub returns thousands of hits, and more specifically you'll often see it in popular gems such as Rails.

Proposal

I believe that the parser could apply optimizations when freeze is called on a Hash or Array literal, similar to what it does for String literals (minus the deduplication).

I implemented it as a proof of concept for [].freeze specifically, and it's not particularly complex, and I'm confident doing the same for {}.freeze would be just as simple.

Potential followup

I also went a bit farther, and did another proof of concept that reuse that opcode for non empty literal arrays. Most of the logic needed to decided wether a literal array can be directly used already exist for the duparray opcode.

So it short opt_ary_freeze / opt_hash_freeze could substitute duparray / duphash if .freeze is called on the literal, and that would save an allocation. That isn't huge, but could be useful for things such as:

```
%i(foo bar baz).freeze.include?(value)
```

Or to avoid allocating hashes and arrays in pattern matching:

```
case value
in { foo: 42 }.freeze
  # ...
end
```

Related issues:

Related to Ruby master - Feature #15393: Add compilation flags to freeze Arra...	**Open**

History

#1 - 01/29/2020 06:47 PM - shevegen (Robert A. Heiler)

I can not evaluate the speed/efficiency part, so from this point of view that may be

ok - I have no idea.

I believe the other part is a general design decision, though. Is the intention really to encourage people to add .freeze all over everywhere? I am not sure about this. It may be good to ask matz about his opinion either way.

A lot of code is e. g. in a style like this

```
EMPTY_Hash = { }.freeze
EMPTY_Array = [ ].freeze
EMPTY_String = ''.freeze
```

In particular the last part strikes me as odd. If you use e. g. frozen-string literals in the comment section set to true, then why is .freeze still used in such .rb files?

https://github.com/kstephens/runify/blob/1ed05a6d3acafe0ce9bafd68b1f44e818d279968/lib/runify.rb

Granted, that is 10 years old code by now, so we can not use it as a basis for evaluationof current use of ruby really - but it just strikes me as strange in general. I am also aware that this is used in a LOT of code bases out there, even aside from github; I saw this in some ruby gems hosted at rubygems.org.

When I remember the old pickaxe book, it rarely showed such examples with .freeze. Literally there was no recommendation for this being a "best practice". (Not that you state so either, but when lots of people use something, you can't help but wonder why they do this.)

Is the intention really to have ruby users use so many .freeze in their ruby code? Is that a idiom that should be the (or a) default?

I mean, I don't know for certain why it is used, but I suspect the most logical explanation may be because people can squeeze out more "efficiency" (probably; such as using .freeze on Strings in the past). Many examples are in the rails active* ecosystem, by the way, on that github search result, and a lot of the results actually reside within files called "bundler.rb" - so I would be a bit wary making too many assumptions in general based on that since it will have some bias. Again, perfectly fine to wish to optimize code, as matz said nobody minds more speed :) - but I would also evaluate the use case at hand and the idioms, together, before considering making optimizations based on that, since I think this is also coupled to a design decision/process (but I may be wrong).

I rarely use that idiom in my ruby code; probably others don't either, so you have to wonder why that idiom originates, and whether it is a good idiom too.

#2 - 01/29/2020 07:27 PM - Dan0042 (Daniel DeLorme)

My first thought was: I like this.

My second thought was: this is frozen strings all over again. People were adding .freeze to all their strings, but the core team (or matz?) considered this was "not the ruby way", and frozen_string_literals was introduced as a countermeasure.

#3 - 01/29/2020 08:48 PM - byroot (Jean Boussier)

Someone pointed to me that #15393 was somewhat related.

#4 - 01/30/2020 08:03 AM - nobu (Nobuyoshi Nakada)

byroot (Jean Boussier) wrote:

> Or to avoid allocating hashes and arrays in pattern matching:
>
> ```
> case value
> in { foo: 42 }.freeze
>   # ...
> end
> ```
>
> This is not a literal hash, so unrelated at all.

#5 - 01/30/2020 09:51 AM - byroot (Jean Boussier)

> This is not a literal hash, so unrelated at all.

My bad, I totally misread the disassembly output.

#6 - 01/30/2020 04:00 PM - Eregon (Benoit Daloze)

- Related to Feature #15393: Add compilation flags to freeze Array and Hash literals added