Ruby master - Feature #16989

Sets: need ♥

06/26/2020 08:18 PM - marcandre (Marc-Andre Lafortune)

Status:	Assigned
Priority:	Normal
Assignee:	knu (Akinori MUSHA)
Target version:	

Description

I am opening a series of feature requests on Set, all of them based on this usecase.

The main usecase I have in mind is my recent experience with RuboCop. I noticed a big number of frozen arrays being used only to later call include? on them. This is O(n) instead of O(1).

Trying to convert them to Sets causes major compatibility issues, as well as very frustrating situations and some cases that would make them much less efficient.

Because of these incompatibilities, RuboCop is in the process of using a custom class based on Array with optimized include? and ===. RuboCop runs multiple checks on Ruby code. Those checks are called cops. RuboCop performance is (IMO) pretty bad and some cops currently are in O(n^2) where n is the size of the code being inspected. Even given these extremely inefficient cops, optimizing the 100+ such arrays (most of which are quite small btw) gave a 5% speed boost.

RuboCop PRs for reference: https://github.com/rubocop-hq/rubocop-ast/pull/29
https://github.com/rubocop-hq/rubocop/pull/8133

My experience tells me that there are many other opportunities to use Sets that are missed because Sets are not builtin, not known enough and have no shorthand notation.

In this issue I'd like to concentrate the discussion on the following request: Sets should be core objects, in the same way that Complex were not and are now. Some of the upcoming feature requests would be easier (or only possible) to implement were Sets builtin.

Related issues:

Related to Ruby master - Feature #16990: Sets: operators compatibility with A...	**Open**
Related to Ruby master - Feature #16991: Sets: add Set#join	**Open**
Related to Ruby master - Feature #16992: Sets: officially ordered	**Open**
Related to Ruby master - Feature #16993: Sets: from hash keys using Hash#key_set	**Open**
Related to Ruby master - Feature #16994: Sets: shorthand for frozen sets of s...	**Open**
Related to Ruby master - Feature #16995: Sets: <=> should be specialized	**Open**

History

#1 - 06/26/2020 08:47 PM - marcandre (Marc-Andre Lafortune)

- Related to Feature #16990: Sets: operators compatibility with Array added

#2 - 06/26/2020 08:47 PM - marcandre (Marc-Andre Lafortune)

- Related to Feature #16991: Sets: add Set#join added

#3 - 06/26/2020 08:47 PM - marcandre (Marc-Andre Lafortune)

- Related to Feature #16992: Sets: officially ordered added

#4 - 06/26/2020 08:47 PM - marcandre (Marc-Andre Lafortune)

- Related to Feature #16993: Sets: from hash keys using Hash#key_set added

#5 - 06/26/2020 08:47 PM - marcandre (Marc-Andre Lafortune)

- Related to Feature #16994: Sets: shorthand for frozen sets of symbols / strings added

#6 - 06/26/2020 08:47 PM - marcandre (Marc-Andre Lafortune)

- Related to Feature #16995: Sets: <=> should be specialized added

#7 - 06/27/2020 02:14 AM - nobu (Nobuyoshi Nakada)

- Assignee set to knu (Akinori MUSHA)

- Status changed from Open to Assigned

#8 - 08/24/2020 04:13 AM - mame (Yusuke Endoh)

We discussed this issue at the last-month meeting.

https://github.com/ruby/dev-meeting-log/blob/master/DevelopersMeeting20200720Japan.md

> matz: Positive to introduce Set into core. But we need to first introduce a set literal. { x, y, z } is good, but JavaScript uses it as another meanings, so it would bring confusion. I have no idea.
> knu: As a set library maintainer, I'll first communicate, and try to solve the easy issues that can be sorted out in the library side.

knu (Akinori MUSHA) ?

#9 - 08/24/2020 12:55 PM - marcandre (Marc-Andre Lafortune)

That's great news.

Was there discussion for a *frozen* set literal of symbols / strings as I proposed in #16994? For sets that need to be mutable or containing different types of objects, I find the existing Set[...] quite convenient.
Moreover, I would like to avoid having to add a magic comment # frozen_set_literals: true...

#10 - 08/24/2020 03:12 PM - Dan0042 (Daniel DeLorme)

> matz: Positive to introduce Set into core. But we need to first introduce a set literal. { x, y, z } is good, but JavaScript uses it as another meanings, so it would bring confusion.

I have to ask, is [lack of] similarity with javascript really a problem? Apparently Python uses { x, y, z } (and set() for an empty set). #5478 has a lot of discussion on Set literals but ultimately nothing beats the simplicity and obviousness of { x, y, z }

#11 - 09/02/2020 09:52 AM - Eregon (Benoit Daloze)

IMHO there is no need for a literal Set notation (which would anyway restrict the type of keys, etc).

> Some of the upcoming feature requests would be easier (or only possible) to implement were Sets builtin.

marcandre (Marc-Andre Lafortune) Which feature requests except having a literal notation (#16994) require Set to be builtin? I think only #16993.
I think we could make good progress here by addressing the other feature requests first.

Also moving Set to core makes might make it more complicated to backport to older Rubies.
(Unfortunately Set is only a default gem in 3.0, but it's probably easy enough to require some Set additions with a different require name)

#12 - 09/02/2020 01:44 PM - marcandre (Marc-Andre Lafortune)

Eregon (Benoit Daloze) wrote in #note-11:

> IMHO there is no need for a literal Set notation (which would anyway restrict the type of keys, etc).

I agree. Set[...] works very well already for constructing sets at runtime out of any time of elements.

> > Some of the upcoming feature requests would be easier (or only possible) to implement were Sets builtin.
>
> marcandre (Marc-Andre Lafortune) Which feature requests except having a literal notation (#16994) require Set to be builtin? I think only #16993.

Indeed, the static notation requires a change to the Ruby parser.

#16993 would be easier and faster if builtin, although it is implementable with a transform_values as I showed in the issue.

A bigger issue (and more important feature imo) is interoperability with Array #16990. If not in core, this would require monkeypatching a bunch of Array methods if not in core and would also require looping in Ruby so would not be as efficient.

> I think we could make good progress here by addressing the other feature requests first.

> Also moving Set to core makes might make it more complicated to backport to older Rubies.

If we want to backport to older Rubies, this could still partly be done with a gem, we'll just need some conditional definitions.

#13 - 09/02/2020 02:45 PM - Dan0042 (Daniel DeLorme)

> which would anyway restrict the type of keys

I don't get that part. If you have a literal like { x, y, z } then x/y/z can be any type of object. There's no restriction. But I agree Set[] is short and good enough.

#14 - 09/02/2020 09:00 PM - Eregon (Benoit Daloze)

Also, does moving Set to core mean rewriting it to C?
I think that would be suboptimal, because every implementation would have to maintain its own native implementation then (or reuse the C extension).
It would also hurt readability.
Much of Ruby's performance can be achieved by JIT'ing Ruby code, and Hash is well optimized, so I'm not sure it's a performance gain either.

BTW SortedSet has this rbtree usage, that seems problematic to move in core (core should not depend on RubyGems):
https://github.com/ruby/ruby/blob/eada6350332155972f19bad52bd8621f607520a2/lib/set.rb#L708
Only moving Set but not SortedSet in core would avoid that issue though, but be somewhat inconsistent.

#15 - 09/02/2020 09:01 PM - Eregon (Benoit Daloze)

Dan0042 (Daniel DeLorme) wrote in #note-13:

> I don't get that part. If you have a literal like { x, y, z } then x/y/z can be any type of object. There's no restriction. But I agree Set[] is short and good enough.

Right, I was thinking of #16994.
Set[] seems already good enough as a general syntax, and no confusion with JS object literals / potentially future Ruby Hash shorthand literals.

#16 - 09/02/2020 09:15 PM - marcandre (Marc-Andre Lafortune)

Eregon (Benoit Daloze) wrote in #note-14:

> Also, does moving Set to core mean rewriting it to C?
> I think that would be suboptimal, because every implementation would have to maintain its own native implementation then (or reuse the C extension).

I'm not sure, but Set is quite small, most of the processing uses Hash...

> Only moving Set but not SortedSet in core would avoid that issue though, but be somewhat inconsistent.

I should have clarified: while I am convinced that Set is a basic fundamental class, I have yet to use SortedSet or see it in use. I would keep it out of core (in set).

#17 - 09/03/2020 01:14 AM - matz (Yukihiro Matsumoto)

I agree with some of your proposals (#16990, #16991, #16993, #16995). I want knu (Akinori MUSHA) to work on this. If I missed something, he will tell us.

I strongly disagree with #16994. There's no evidence we need frozen sets of strings or symbols that much. Even if we do, I think frozen arrays should come first.

I weakly disagree with #16992. Currently set orders are determined by the internal hash. We may change the implementation in the future to improve performance or memory overhead. Fixing the order could possibly restrict the future implementation choice.

Matz.

#18 - 09/03/2020 07:48 AM - knu (Akinori MUSHA)

OK, I think it's good for us now to diverge from the original philosophy of Set when I first wrote it and pursue the performance and integrity with other parts of Ruby. There are many parts in Set where I avoided optimization in order to retain extensibility (like subclassing and stuff), but I'll unlock the bar.

I'm also planning to remove SortedSet and leave it to an external gem because of the partial dependency on rbtree and the fallback implementation

which performs quite poorly.

I'm not absolutely sure about introducing literals in the form of { a, b, c } because I myself is the one who is quite familiar with the shorthand notation introduced in ES6 and would like to have something similar in Ruby.

#19 - 09/03/2020 09:44 PM - marcandre (Marc-Andre Lafortune)

knu (Akinori MUSHA) wrote in #note-18:

> OK, I think it's good for us now to diverge from the original philosophy of Set when I first wrote it and pursue the performance and integrity with other parts of Ruby. There are many parts in Set where I avoided optimization in order to retain extensibility (like subclassing and stuff), but I'll unlock the bar.
>
> I'm also planning to remove SortedSet and leave it to an external gem because of the partial dependency on rbtree and the fallback implementation which performs quite poorly.
>
> I'm not absolutely sure about introducing literals in the form of { a, b, c } because I myself is the one who is quite familiar with the shorthand notation introduced in ES6 and would like to have something similar in Ruby.

This sounds great!
Let me know if I can help.

#20 - 09/03/2020 09:44 PM - marcandre (Marc-Andre Lafortune)

matz (Yukihiro Matsumoto) wrote in #note-17:

> I agree with some of your proposals (#16990, #16991, #16993, #16995). I want knu (Akinori MUSHA) to work on this. If I missed something, he will tell us.

Thank you for reviewing these :-)

> I strongly disagree with #16994. There's no evidence we need frozen sets of strings or symbols that much. Even if we do, I think frozen arrays should come first.

Right, I agree that there is a larger discussion about frozen & static literals to be had.

Would a non-frozen notation for Set of strings / symbols have a chance of being accepted? I would like to avoid Set.new(%w[a list of words]) as can be currently seen in RuboCop or in Rails:

https://github.com/rails/rails/blob/master/actionpack/lib/action_dispatch/http/cache.rb#L128
https://github.com/rails/rails/blob/master/actionpack/lib/action_dispatch/http/headers.rb#L25-L44

I think that many gems have simply not really taken care about Sets. For example, this file in Bundler defines a 40-element array constant, only to call include? on it...

https://github.com/rubygems/rubygems/blob/master/bundler/lib/bundler/settings.rb#L9

I feel that a builtin way to write this might help.