Ruby - Feature #12075

some container#nonempty?

02/16/2016 08:08 AM - naruse (Yui NARUSE)

Status:	Feedback	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

I sometimes write following code.

```
ary = some_metho_returns_nil_or_empty_container() # nil or "" or [] or {}

if ary && !ary.empty?
  #  some code
end
```

But the condition ary && !ary.empty? is too long and complex.
Though Ruby 2.3 introduces &., but this can't be written as ary&.empty?.

One idea is add nonempty? write as ary&.nonempty?.

akr: nonempty? is not good name because human is not good at handling

This discussion matches following core classes:

- String
- Array
- Hash

Related issues:

Related to Ruby - Feature #13395: Add a method to check for not nil	**Open**
Related to Ruby - Feature #17330: Object#non	**Open**
Related to Ruby - Feature #20498: Negated method calls	**Open**

History

#1 - 02/16/2016 08:30 AM - mrkn (Kenta Murata)

How about ary.include_something? ?

#2 - 02/16/2016 08:31 AM - akr (Akira Tanaka)

How about "some?".

#3 - 02/16/2016 08:52 AM - sawa (Tsuyoshi Sawada)

That is a use case for Rails' blank? (or present?).

```
class Object
  def blank?
    respond_to?(:empty?) ? !!empty? : !self
  end
end

unless ary.blank?
  #  some code
end
```

What about incorporating these methods from Rails?

#4 - 02/16/2016 10:02 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

+1 for bringing blank? and present? to Ruby core. I often use (a || '').empty? checks for not having to depend on ActiveSupport directly in my Ruby code. I'd love to see them in Ruby core though.

#5 - 02/16/2016 02:17 PM - shyouhei (Shyouhei Urabe)

No, the OP wants to detect things that are *not* empty. This is not what blank? means. Also note that blank? in ActiveSupport has different (far more Rails-centric) semantics than what is described in #3.

https://github.com/rails/rails/blob/b3eac823006eb6a346f88793aabef28a6d4f928c/activesupport/lib/active_support/core_ext/object/blank.rb#L115

#6 - 02/16/2016 03:28 PM - sawa (Tsuyoshi Sawada)

Shyouhei Urabe wrote:

> No, the OP wants to detect things that are *not* empty. This is not what blank? means.

Yes, I meant that blank? is the opposite of what OP wants. present? is what the OP wants. I replaced if in the original example with unless, so it is the same (Or, to retain if, present? can be used).

> ActiveSupport has different (far more Rails-centric) semantics than what is described in #3.

I see.

#7 - 02/16/2016 05:30 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

The implementation doesn't have to be the same as the one implemented by ActiveSupport. I think it would be fine to simply check for nil? and empty?. But I'd like to keep the names present? and blank? anyway.

#8 - 02/19/2016 11:10 PM - shevegen (Robert A. Heiler)

Perhaps the name .contains? might be good?

Although, .include? sort of is more or less synonymous with .contains? so perhaps this is not a good choice either.

.nonempty? seems a bit long, .non_empty? would be even longer :)

.empty? is a very good name already, I am not sure if "! .empty?" has a good name, though ruby uses the keyword "not" already. Could use .notempty? haha sorry, I have no good suggestion for a fitting name for negation either, but I am totally fine with the idea and functionality of the proposal itself, it's a good one.

#9 - 02/25/2016 02:47 AM - shyouhei (Shyouhei Urabe)

I have just learned that zsh(1) calls this concept being "full". http://zsh.sourceforge.net/Doc/Release/Expansion.html#Glob-Qualifiers

#10 - 02/25/2016 05:12 AM - sawa (Tsuyoshi Sawada)

What about introducing NilClass#empty?:

```
nil.empty? # => true
```

The code in question can be written simply as:

```
unless ary.empty?
  #  some code
end
```

If the original proposal is going to be realized, then a new method would have to be added to NilClass, String, Array, and Hash, but my proposal makes use of the existing method empty?, and needs to add to only NilClass, keeping the change minimal.

Even if the original proposal is going to be realized, extending empty? as above would make it the complete opposite of such method, and would introduce parallelism.

#11 - 02/25/2016 04:37 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I like this idea a lot, Tsuyoshi. I'm +1 for introducing nil.empty? as returning true.

#12 - 02/26/2016 04:33 AM - nobu (Nobuyoshi Nakada)

First, along this line, we'll need negative forms for all predicate methods.

And I think nil.empty? makes no sense.

Just an idea:

```
module Kernel
  def not(*a)
    not a.empty? ? self : __send__(*a)
```

```
  end
end

ary = nil;   ary&.not(:empty?) #=> nil
ary = [];    ary&.not(:empty?) #=> false
ary = [nil]; ary&.not(:empty?) #=> true
```

#13 - 02/26/2016 04:40 AM - nobu (Nobuyoshi Nakada)

Or

```
module Kernel
  def !(*a)
    a.empty? ? super() : !__send__(*a)
  end
end

ary = nil;   ary&.!:empty? #=> nil
ary = [];    ary&.!:empty? #=> false
ary = [nil]; ary&.!:empty? #=> true
```

#14 - 02/26/2016 05:22 AM - phluid61 (Matthew Kerwin)

Nobuyoshi Nakada wrote:

> First, along this line, we'll need negative forms for all predicate methods.
>
> And I think nil.empty? makes no sense.
>
> Just an idea:
>
> ```
> module Kernel
>   def not(*a)
>     not a.empty? ? self : __send__(*a)
>   end
> end
>
> ary = nil;   ary&.not(:empty?) #=> nil
> ary = [];    ary&.not(:empty?) #=> false
> ary = [nil]; ary&.not(:empty?) #=> true
> ```

I like this proposal. I definitely prefer the word 'not' over the symbol '!', because ary&.!:empty? has too much consecutive punctuation for my eyes.

Would there be value in extending it to accept a block?

```
module Kernel
  def not(*a, &b)
    not a.empty? ? self : __send__(*a, &b)
    # or even:
    #not a.empty? ? (b ? (yield self) : self) : __send__(*a, &b)
  end
end

ary = [];  ary&.not(:any?){|x|x>0} #=> true
ary = [1]; ary&.not(:any?){|x|x>0} #=> false
```

#15 - 02/26/2016 07:25 AM - nobu (Nobuyoshi Nakada)

https://github.com/ruby/ruby/compare/trunk...nobu:feature/12075-not

#16 - 02/26/2016 09:31 AM - Eregon (Benoit Daloze)

Nobuyoshi Nakada wrote:

> https://github.com/ruby/ruby/compare/trunk...nobu:feature/12075-not

I like it!

#17 - 05/17/2016 07:31 AM - nobu (Nobuyoshi Nakada)

- Description updated

#18 - 05/17/2016 09:26 AM - naruse (Yui NARUSE)

- Status changed from Assigned to Feedback

Array#any? seems to work usual use cases.
Feedback if another cases are discovered.

#19 - 09/26/2016 06:27 PM - sorah (Sorah Fukumori)

I know any? works on some use cases, but I'm positive to have a proposed method because using any? has a pitfall. We have to guarantee an array doesn't have only false or nil. Also I'm worrying users who started to use any? for this use case, but doesn't know this pitfall.

I'm positive on Object#not idea.

#20 - 04/12/2017 12:30 PM - nobu (Nobuyoshi Nakada)

- Related to Feature #13395: Add a method to check for not nil added

#21 - 11/17/2020 12:54 PM - nobu (Nobuyoshi Nakada)

- Related to Feature #17330: Object#non added

#22 - 06/04/2021 06:04 AM - mame (Yusuke Endoh)

I like Array#some? and Hash#some?. It is so frequent to write !ary.empty?.

I think the main concern is only about empty?, not other predicate methods. So I'm skeptical about the need of the generalization like ary.not(:empty?). I don't like it very much because it is longer and slower than the dedicated method.

#23 - 06/04/2021 08:37 AM - sawa (Tsuyoshi Sawada)

I want to mention that some of the method names proposed so far would break symmetry.

Note that "empty" is a property of the container, not the elements. When you have a = [1], what is questioned whether it is empty or not is a, but not 1. Among the words proposed so far, "any" and "some" are not a property of the container, but are a word used together with the elements, e.g., "some element 1". "present" in this respect is also not appropriate because it is a property of the element ("1 is present"), not the container (not "a is present"). If the method in question is to be defined as the negation of empty?, then its name should be a property of the container.

Among the words proposed so far, "non-empty", "not empty", and "full" would work. Another word that comes to mind is "occupied", but it may be too long.

#24 - 08/17/2021 02:31 AM - knu (Akinori MUSHA)

I tend to like ary.size > 0 more than !ary.empty? because the former literally has a "positive" nuance and therefore it's more readable in many cases. In that sense, the name nonempty? does not sound ideal to me.

So, I came up with this. What about size?? It's in the existing vocabulary; FileTest and File::Stat have size?. It presumably originates from the shell script expression [-s file] (test(1)) that tests if a file has a size greater than zero. Those methods return nil when the size is zero and return the actual size otherwise, but that's an old convention (cf. nonzero?) and we can just add size? as a boolean method to Array/Hash/String.

I don't think we should add one to NilClass nor Object (Kernel) in general, but I'm not absolutely sure about what to do with other existing classes with size: Integer/MatchData/Range etc. □□

#25 - 08/17/2021 05:43 AM - p8 (Petrik de Heus)

What about ary.filled? ?
It nicely pairs with empty and is not too long.
It’s also a property of the container. And unlike full?, it doesn’t imply there is no more room.

#26 - 08/18/2021 01:21 PM - gotoken (Kentaro Goto)

Boolean size? looks good to me. I vote for it.

My first choice was nonempty? because this is very common word in popular algorithm text books, e.g., Knuth, Cormen, Sedgewick, Tarjan, Aho, Hopcroft, etc.

However some people pointed if array.nonempty? looks negative and does not improve if !array.empty?. They don't want to write negative condition there. I can agree that.

filled? also makes me mind capacity of the container and it is misleading. IMHO.

some? is cool but we have already any?. These are very confusing.

not(...) is quite general idea, and it seems to need too complicated discussions for this simple feature request. Neither not method does not meet avoidance of negation.

#27 - 08/18/2021 02:25 PM - sawa (Tsuyoshi Sawada)

p8 (Petrik de Heus) wrote in #note-25:

> What about ary.filled? ?
> It nicely pairs with empty and is not too long.
> It's also a property of the container. And unlike full?, it doesn't imply there is no more room.

I think filled? is a perfect method name. You nailed it.

Or, if you are going with past participles, perhaps loaded? may also work.

#28 - 08/19/2021 12:54 AM - duerst (Martin Dürst)

sawa (Tsuyoshi Sawada) wrote in #note-27:

> I think filled? is a perfect method name.

I agree with @gotoken (Kentaro Goto) that filled? is confusing.

#29 - 08/19/2021 06:57 AM - p8 (Petrik de Heus)

duerst (Martin Dürst) wrote in #note-28:

> I agree with @gotoken (Kentaro Goto) that filled? is confusing.

Yes, it might be confusing when you have an array like: Array.new(3) # => [nil, nil, nil]
Is it filled or only if all nils are replaced with non-nil data?

It's a bit longer (1 more character than nonempty), but ary.populated? doesn't have the capacity confusion.

> Computers To fill (an empty field or array) with data.

https://www.thefreedictionary.com/populate

#30 - 08/19/2021 03:00 PM - austin (Austin Ziegler)

I'm not entirely sure we need a new method for this, because we could always use ary.size.nonzero? or ary.size.positive?. It is also possible to tell the difference between an empty vs non-empty array with [nil].any? { true } and [].any? { true }. The advantage to #size.nonzero? or #size.positive? is that it works on String, Hash, and Array. It might work on a lot of enumerables, but only if they respond to #size and #size doesn't require enumeration. It even works on endless ranges ((3..).size.positive?).

Of the suggested methods, I think I like #nonempty? best, and #size? second-best. I like #some? least because it is confusing with any?.

#31 - 09/11/2021 09:31 AM - sawa (Tsuyoshi Sawada)

Some additional candidates for the method name:

content?
substantial?

#32 - 09/11/2021 12:32 PM - knu (Akinori MUSHA)

Here's my opinion about the idea that we should just import ActiveSupport's present?.

ActiveSupport's present? is defined for all kinds of objects, and Rails application programmers are so much used to calling present? on any object including what can be evaluated to nil. I think this is probably because present? predates the &. operator and I guess when it came out it felt handy to be able to test if an object is neither nil nor empty with just one method call. But today, you don't need that because you can just say array_or_nil&.nonempty? instead of array_or_nil.present?. In other words, we wouldn't need non-container objects, namely nil, to respond to a present? method, and this is one reason not to import present?. We could just add a new non-emptiness tester method only to container classes.

Secondly, present? is defined as the opposite of blank?, and string.present? is not equivalent to !string.empty?. So, String would need a different method anyway even if we were to add present? only to Array and Hash. I can also point out that String#blank? considers a wider set of characters as "blank" than our stock method String#strip does, so String#blank? as it is now might not fit in the core without reconsideration from the I18n point of view.

So, no matter if importing partially or not, I think there would be mass confusion.

#33 - 09/11/2021 01:04 PM - knu (Akinori MUSHA)

One of the selling points of size?, aside from the appearance of the name in the existing class File::Stat, is that it explicitly says it would check the size, which means it is essentially equivalent to size > 0 and never like each { true } that can cause a side effect. Plus, the Enumerator API has a size

property (cf. Enumerator.new(size) { ... }) defined inside, so Enumereator#size? would really fit there.

#34 - 10/06/2021 11:19 AM - p8 (Petrik de Heus)

If size? will be chosen, should we also add length? as size and length are aliases?

#35 - 10/06/2021 12:46 PM - knu (Akinori MUSHA)

Probably not, considering that some classes I named earlier, File::Stat and Enumerator, only have #size and no #length.
Adding #length? in itself would be fine, but that'd make you feel awkward not to add #length also. □□

#36 - 10/07/2021 12:51 AM - sawa (Tsuyoshi Sawada)

knu (Akinori MUSHA) wrote in #note-33:

> [...]size? [...] explicitly says it would check the size, which means it is essentially equivalent to size > 0

While this might make sense in JavaScript, given that 0 is truthy in Ruby, I do not agree with this opinion. Where does > 0 come from? It looks like it appeared from nowhere. I think the most natural definition of such method size? would be mapping of size to boolean values (via !!), which would be true in Ruby as long as size is defined (i.e., is numeric), even when it is 0.

Intuitively, "x.size?" is more likely interpreted as "is the size defined for x?" rather than "is the size of x greater than 0?"

Hence, I think the method name size? is inappropriate for the suggested feature.

#37 - 10/07/2021 03:11 AM - knu (Akinori MUSHA)

> Intuitively, "x.size?" is more likely interpreted as "is the size defined for x?" rather than "is the size of x greater than 0?"

When one knew size? always comes with size, one should see that asking that question does not make much sense, simply because every container instance has a size just like every file has a size, which is assumed by File::Stat#size?. I already opposed adding a non-emptiness predicate method to Object/Kernel.

#38 - 08/21/2022 06:22 PM - dsisnero (Dominic Sisneros)

I am assuming you do not want to use each which does correct thing if container is non empty . I assume you want map method that return same type of container. Why not just add a new map method?

Hash#fmap>. Hash
Array#fmap -> Array
Option fmap -> Option
Result fmap -> Result

Bonus, have fmap follow functor laws we can start using it with monads.

fmap id = id

[1,2,3].fmap{|i| i} == [1,2,3]

[1,2,3].map{|x| x*2}.map{|x| x*3} = [2,4,6].map{_1*3} = [6,12,18]

[1,2,3].map{ |x| 2*(3*x)} [6,12,18]

#39 - 05/20/2024 04:24 AM - nobu (Nobuyoshi Nakada)

- Related to Feature #20498: Negated method calls added