Add strict Enumerable#single

06/27/2017 06:08 AM - dnagir (Dmytrii Nagirniak)

Status: Feedback
Priority: Normal
Assignee:
Target version:

Description

Summary

This is inspired by other languages and frameworks, such as LINQ's Single (pardon MSDN reference), which has very big distinction between first and single element of a collection.

- first normally returns the top element, and the developer assumes there could be many;
- single returns one and only one element, and it is an error if there are none or more than one.

We, in Ruby world, very often write fetch_by('something').first assuming there's only one element that can be returned there.

But in majority of the cases, we really want a single element.

The problems with using first in this case:

- developer needs to explicitly double check the result isn't nil
- in case of corrupted data (more than one item returned), it will never be noticed

Enumerable#single addresses those problems in a very strong and specific way that may save the world by simply switching from first to single.

Other information

- we may come with a better internal implementation (than self.map)
- better name could be used, maybe only is better, or a bang version?
- re-consider the "block" implementation in favour of a separate method (single!, single_or { 'default' })

The original implementation is on the ActiveSupport https://github.com/rails/rails/pull/26206
But it was suggested to discuss the possibility of adding it to Ruby which would be amazing.

Related issues:
Related to Ruby master - Feature #18135: Introduce Enumerable#detect_only

History

#1 - 06/27/2017 06:09 AM - dnagir (Dmytrii Nagirniak)

- Tracker changed from Bug to Feature

#2 - 06/27/2017 10:17 AM - Eregon (Benoit Daloze)

+1, I have found this useful a few times as well.
Usually, I just define my own on Array, but it makes sense as well for Enumerable.

#3 - 06/27/2017 08:05 PM - shevegen (Robert A. Heiler)

I am not against or in favour of it but just a question.

What would the results be for the following code? In ruby (I find it easier to read ruby code rather than the description actually):
I always feel uncomfortable whenever using `first` for this purpose.

What would the results be for the following code? In Ruby (I find it easier to read Ruby code rather than the description actually):

```ruby
[].single
[1].single
[1,2].single
[1,2,3].single

[].single
{cat: 'Tom'}.single
{cat: 'Tom', mouse: 'Jerry'}.single

(And any other Enumerable objects I may have forgotten here.)
```

I wrote a quick Ruby implementation before realizing there was a link to a Rails PR. Here are the results of your examples (and one added):

```ruby
module Enumerable
  def single
    if one?
      first
    else
      if block_given?
        yield
      else
        raise "wrong collection size (actual #{size || count}, expected 1)"
      end
    end
  end
end

[].single
#> RuntimeError: wrong collection size (actual 0, expected 1)

[1].single
#=> 1

[1,2].single
#> RuntimeError: wrong collection size (actual 2, expected 1)

[1,2,3].single
#> RuntimeError: wrong collection size (actual 3, expected 1)

[].single
#> RuntimeError: wrong collection size (actual 0, expected 1)

{cat: 'Tom'}.single
#=> [:cat, "Tom"]

{cat: 'Tom', mouse: 'Jerry'}.single
#> RuntimeError: wrong collection size (actual 2, expected 1)

[].single { 42 }
#> 42
```

Edit: Caveat, my implementation doesn't handle an Infinite unsized enumerator, unlike the Rails PR which does.
Enumerable#first returns not only the first element, the elements at the beginning up to the number given by an optional argument.

How about an optional boolean argument exact to Enumerable#first or Enumerable#take?

shevegen (Robert A. Heiler) wrote:

What would the results be for the following code?

I would expect the following:

```ruby
[].single => error
[1].single => 1
[1, 2].single => error
[1, 2, 3].single => error

[].single => error
{cat: 'Tom'}.single same as .first => [:cat, 'Tom']
{cat: 'Tom', mouse: 'Jerry'}.single => error
```

The purpose of the single suggested is to return one and only one element. So it doesn't seem right to mix it up with first as it'll only add confusion, especially when used with a block.

On the other hand, I feel like a separate method that does one small thing well would be a much better API.

ROM exposes an interface I like when reading results from the db:

#one! - raise an error unless the result's #size is exactly 1
#one - raise an error if the result's #size is greater than 1. Return the result of #first otherwise (so an empty result returns nil).

I don't think the implementation should use the #one? predicate though. It would be confusing if [nil, true, false].single gave you nil instead of raising an error.

Hmm, I don't like the name single. Besides that, I think it may be useful for database access, but I don't see the use-case of this method for generic Enumerable.

Matz.

I think of single as a method towards mutual exclusivity. If an Array or Enumerable from another expression should only have a single element, then this gives the process a much faster setup and possible rescue, as I currently have
one of my projects checking for the existence of 3 headers, X-GitHub-Event, X-GitLab-Event, and X-Gogs-Event, and I found the easiest way was to use one from Enumerable, but I wanted it to error out so that I could catch it with the rest of my raised exceptions from other errors that arise in the handling of the request.

How about these for suggestions.

one_or_raise
one_or_nothing

Part of my code for context.

```ruby
events = {'github' => github, 'gitlab' => gitlab, 'gogs' => gogs}

events_m_e = events.values.one?

case events_m_e
  when true
    event = 'push'
    service = events.select { |key, value| value }.keys.first
  when false
    halt 400, {'Content-Type' => 'application/json'}, {'message': 'events are mutually exclusive', 'status': 'failure'}.to_json
  else
    halt 400, {'Content-Type' => 'application/json'}, {'status': 'failure', 'message': 'something weird happened'}
end
```

#12 - 04/24/2018 12:40 PM - nobu (Nobuyoshi Nakada)

How about Enumerable#just(num=1) or Enumerable#only(num=1)?

#13 - 07/26/2018 08:40 PM - shan (Shannon Skipper)

nobu (Nobuyoshi Nakada) wrote:

How about Enumerable#just(num=1) or Enumerable#only(num=1)?

Or maybe a slightly more verbose Enumerable#first_and_only(num = 1)?

#14 - 04/02/2019 04:11 PM - lugray (Lisa Ugray)

I was pointed here after sharing the following code with my team mates. I really like the idea, and find I often reach for it. I second the name only.

```ruby
module Enumerable
  def only
    only!
    rescue IndexError
      nil
    end
  
  def only!
    raise( IndexError, "Count (#{count}) is not 1") if count != 1
    first
  end
end
```

#15 - 10/06/2019 08:14 PM - jonathanhefner (Jonathan Hefner)

matz (Yukihiro Matsumoto) wrote:

Hmm, I don't like the name single. Besides that, I think it may be useful for database access, but I don't see the use-case of this method for generic Enumerable.

I use (monkey-patched) Enumerable#single in Ruby scripts which must fail fast when they encounter ambiguity. For example Nokogiri::HTML(html).css(selector).single to ensure an unambiguous matching HTML element. Or Dir.glob(pattern).single to ensure an unambiguous matching file.

Also, I agree that only would be a better name. And it would read more naturally if accepting an n argument like Enumerable#first does.

#16 - 10/07/2019 02:23 AM - Dan0042 (Daniel DeLorme)
I actually have this as single in my own code, but only sounds fine also. I'd want a non-raising version (perhaps via a raise keyword arg?), as my usage tends to be like this:

```ruby
if match = filenames.select { |f| f.start_with?(prefix) }.single
    redirect_to match
end
```

#17 - 10/17/2019 05:59 AM - matz (Yukihiro Matsumoto)
I don't like only either since these names do not describe the behavior.
Matz.

#18 - 10/17/2019 06:24 AM - kinoppyd (Yasuhiro Kinoshita)

```ruby
[1, 2].mono
[1, 2].solo
[1, 2].alone
```

#19 - 10/17/2019 07:24 AM - Hanmac (Hans Mackowiak)
Dan0042 (Daniel DeLorme) wrote:

+1

I actually have this as single in my own code, but only sounds fine also. I'd want a non-raising version (perhaps via a raise keyword arg?), as my usage tends to be like this:

```ruby
if match = filenames.select { |f| f.start_with?(prefix) }.single
    redirect_to match
end
```

instead of #select, shouldn't you use #find so it doesn't need to check the others when it already found a match?

#20 - 10/17/2019 05:55 PM - Dan0042 (Daniel DeLorme)

instead of #select, shouldn't you use #find so it doesn't need to check the others when it already found a match?

No, because it should return nil when there's more than one match.

#21 - 11/13/2019 04:04 PM - kaikuchn (Kai Kuchenbecker)
I like one a lot. Especially since there's already one?.

#22 - 08/28/2020 08:19 PM - fatkodima (Dima Fatko)
I have opened a PR - https://github.com/ruby/ruby/pull/3470

```ruby
# Returns one and only one item. Raises an error if there are none or more than one.
[99].one # => 99
[].one # => RuntimeError: collection is empty
[99, 100].one # => RuntimeError: collection contains more than one item

# If collection is empty and no block was given, returns default value:
[].one(99) # => 99

# If collection is empty and a block was given, returns the block's return value:
[].one { 99 } # => 99
```

#23 - 08/28/2020 09:14 PM - marcandre (Marc-Andre Lafortune)
If we introduce one, it would be nice to support regexp; maybe use === for matching when given an argument?

```ruby
%w[hello world].one(/ll/) # => 'hello'
```

#24 - 08/29/2020 04:23 AM - sawa (Tsuyoshi Sawada)
Having Enumerable#find take an optional keyword argument, say exception:, would make more sense, be useful, and have more generality.
If collection is empty and a block was given, returns the block’s return value:

```
1].find(exception: true){true}  #=> 1
1, 2, 3].find(exception: true){true}  #=> Error
[].find(exception: true){true}  #=> Error
%w[hello world].find(exception: true){/l/ == _1}  #=> "hello"
%w[hello world].find(exception: true){/l/ == _1}  #=> Error
%w[hello world].find(exception: true){/%w\[hello world\]#/ == _1}  #=> nil
%w[hello world].find(exception: true){/%w\[hello world\]#/ == _1}  #=> Error
%w[hello world].find(exception: true){/x/ == _1}  #=> Error
```

#25 - 08/29/2020 11:11 AM - Dan0042 (Daniel DeLorme)

If collection is empty and a block was given, returns the block’s return value:

```
I really think the block form should be like find/select

[1,2,3].one{ _1.even? }  #=> 2
[1,2,3,4].one{ _1.even? }  #=> error
[1,2,3,4].one(nil){ _1.even? }  #=> nil
```

Having Enumerable#find take an optional keyword argument, say exception:, would make more sense, be useful, and have more generality.

I don't think so; find only returns the first element found. An argument that makes it continue searching and return an exception if it finds a second match... that alters the fundamental behavior too much imho.

#26 - 08/29/2020 12:29 PM - sawa (Tsuyoshi Sawada)

Dan0042 (Daniel DeLorme) wrote in #note-25:

```
I really think the block form should be like find/select

[1,2,3].one{ _1.even? }  #=> 2
[1,2,3,4].one{ _1.even? }  #=> error
[1,2,3,4].one(nil){ _1.even? }  #=> nil
```

... continue searching [...] that alters the fundamental behavior too much imho.

I think you are right.

But the word “one”, which has the same etymology as the indefinite article “an”, is not appropriate here. Its meaning is to arbitrarily pick out a single entity. This does match the concept discussed here.

I think the English word that best matches the concept is the definite article. This word presupposes that there is exactly one corresponding entity in the context, and picks that entity. If the presupposition is not satisfied, then the entire expression would be uninterpretable in the case of natural languages, which amounts to raising an exception in the case of programming languages.

```
[1, 2, 3].the(&:even?)  #=> 2
[1, 2, 3, 4].the(&:even?)  #=> error
[1, 3].the(&:even?)  #=> error
```

Using the usual block parameter, `[1,2,3].the{|x| x.even?}` could be read as “the x (out of [1, 2, 3]) such that x is even.” If such x does not uniquely exist, this expression is uninterpretable.

#27 - 08/29/2020 12:42 PM - Dan0042 (Daniel DeLorme)

Hmmm, just now I realized there's a simple idiom that's roughly equivalent to one/single

```
[1,2].inject{|x, y| break}  #=> nil
[1,2].inject{|x, y| raise}  #=> error
[1].inject{|x, y| break}  #=> 1
[1].inject{|x, y| raise}  #=> 1
[1].inject{|x, y| break}  #=> nil
[1].inject{|x, y| raise}  #=> nil (instead of error)
```

#28 - 08/30/2020 11:26 AM - Eregon (Benoit Daloze)

matz (Yukihiro Matsumoto) wrote in #note-10:

```
Hmm, I don't like the name single.
```
matz (Yukihiro Matsumoto) wrote in #note-17:

I don't like only either since these names do not describe the behavior.

Could you explain why?
I think single as in "return a single element or error out" is the best name here, and the most intuitive.
If I want a random element, then #sample already exists, so I don't think there would be any confusion to what Array#single or Enumerable#single would do.
array.only reads weirdly to me. array.single is so much nicer, no?

#29 - 09/15/2020 06:35 AM - shan (Shannon Skipper)
How about #sole since it means one and only and is concise?

[].sole
#!> SoleError: empty Array when single value expected (contains 0, expected 1)

Set.new.sole
#!> SoleError: empty Set when single value expected (contains 0, expected 1)

[41, 42, 43].sole
#!> SoleError: multiple values in Array when just one expected (contains 3, expected 1)

[42].sole
#=> 42

Or #one_and_only, but it's more wordy.

#30 - 09/25/2020 07:12 AM - nobu (Nobuyoshi Nakada)
Dan0042 (Daniel DeLorme) wrote in #note-25:

If collection is empty and a block was given, returns the block's return value:

I really think the block form should be like find/select

\[
\begin{align*}
[1,2,3].\text{one}(\_1.\text{even?}) & \Rightarrow 2 \\
[1,2,3,4].\text{one}(\_1.\text{even?}) & \Rightarrow \text{error} \\
[1,2,3,4].\text{one}(\text{nil})(\_1.\text{even?}) & \Rightarrow \text{nil}
\end{align*}
\]

It looks close to Enumerable#one? which counts truthy values only, but has a different semantics.

#31 - 09/25/2020 07:14 AM - mame (Yusuke Endoh)
A practical use case: When scraping a HTML document or something, sometimes we assume that an array length is 1.

\[
\text{nodes = html_node.children.select } | \{ \text{node| node.name == "table" } \}
\]
raise if nodes.size == 1
\text{the_node_i_want = nodes.first}

The raise is needed for the case where my assumption is wrong.

This proposal makes it a bit helpful:

\text{the_node_i_want = html_node.children.select } | \{ \text{node| node.name == "table" } \}.sole

#32 - 10/02/2020 02:14 PM - nobu (Nobuyoshi Nakada)
What about pattern matching?

\[
\text{case } []; \text{ in } [a]; \text{ p a; end } \Rightarrow \text{NoMatchingPatternError } ([])
\]
\[
\text{case } [1]; \text{ in } [a]; \text{ p a; end } \Rightarrow 1
\]
\[
\text{case } [1,2]; \text{ in } [a]; \text{ p a; end } \Rightarrow \text{NoMatchingPatternError } ([1, 2])
\]

#33 - 10/06/2020 09:06 PM - shan (Shannon Skipper)
For Arrays, pattern matching does seem like a fair solution. Wouldn't that not work for other Enumerable collections?

\text{Set.new}([42]) \text{ in } [\text{one}] \#!> \text{NoMatchingPatternError } (#<Set: \{42\}>)
It seems an Enumerable method would be more slick and nice for method chaining.

```ruby
Set.new([42]).sole #=> 42
Set.new([42]).soledigits #=> [2, 4]
```

#34 - 09/02/2021 12:28 AM - mame (Yusuke Endoh)
* Related to Feature #18135: Introduce Enumerable#detect_only added

#35 - 09/02/2021 05:52 PM - meisel (Michael Eisel)

+1 for this, I've needed it in many cases, generally where I have to select an element from a set and there's no stable/document way of doing this. So, I find some criteria that seem to work, but would like to protect against my criteria being wrong or invalidated by future changes. In the past, I've needed it for sets like sibling directories and sibling XML nodes.

As for the name, I think #take_only would be another reasonable option