Ruby master - Feature #16833

Add Enumerator#empty?

05/06/2020 02:09 PM - f3ndot (Justin Bull)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

It was surprising to me that Enumerator, something mixed into Array, does not include #empty?. I think it is reasonable to assume people may have to guard iterating and other logic based on the emptiness of an enumerator, such was my case.

```
 # pretend there's convoluted enumerator logic to produce this structure
table_rows = [{ data: ['First', 'Second', 'Third'], config: {} }, { data: [4, 5, 6], config: { color: 'red' } }].to_enum

return if table_rows.empty?

table_header = table_rows.first[:data] # requires an empty guard
# ...
```

I propose that it simply behaves as #take(1).to_a.empty? instead of aliasing to something like #none? because of falsey elements or #size == 0 because of potential nil returns:

```
[].to_enum.empty?         # => true
[false].to_enum.empty?    # => false
[nil].to_enum.empty?      # => false
[0].to_enum.empty?        # => false
[1, 2, 3].to_enum.empty? # => false
```

History

#1 - 05/06/2020 02:30 PM - f3ndot (Justin Bull)

- Description updated

- File add-enumerable-empty.patch added

#2 - 05/06/2020 04:34 PM - sawa (Tsuyoshi Sawada)

Not sure what you mean by

> Enumerator, something mixed into Array

Do you mean this?

> Enumerator, which is mixed into Array

If that was what you meant, then that is not the case. Array mixes-in Enumerable, not Enumerator.

Even if that was the case, it is not clear how that is relevant to your argument. Array has its own empty? method, so even if its ancestor had an empty? method, the latter would be overridden by the former.

#3 - 05/06/2020 04:40 PM - sawa (Tsuyoshi Sawada)

Your use case is not clear. Why can't you do this?

```
table_rows = [{ data: ['First', 'Second', 'Third'], config: {} }, { data: [4, 5, 6], config: { color: 'red' } }].to_enum
table_rows.to_a.dig(0, :data) # => ["First", "Second", "Third"]

table_rows = [].to_enum
table_rows.to_a.dig(0, :data) # => nil
```

I don't understand why you need to conditionally return as in your code.

#4 - 05/06/2020 04:44 PM - marcandre (Marc-Andre Lafortune)

Enumerators can behave very differently than arrays. Some have side-effects. What's supposed to happen with those?

```
require 'stringio'
s = StringIO.new("first\nsecond\nthird\nlast").each_line
s.first # => "first"
s.first # => "second"
s.empty? # => false
s.first # => "third" or "last"?
s.empty? # => false or true?
```

You may want to rethink your algorithm, or simply call to_a once and act on the array.

For similar reasons, Enumerator#size exist and is lazy, but it may return nil (e.g. s.size above) so wouldn't completely serve your purpose either.

For these reason I think you should either close your request or elaborate on an actual use case where a generic enumerator is being used, as well as the actual effect of empty? on external iterators, ...

#5 - 05/06/2020 07:55 PM - shevegen (Robert A. Heiler)

sawa wrote:

> If that was what you meant, then that is not the case. Array mixes-in
> Enumerable, not Enumerator.

Happens to me as well a lot - I always mix these two up. May be because
the name is so similar. :)

To the suggestion itself: I have no particular preference either way so
I won't really comment on the proposal itself.

Files

add-enumerable-empty.patch	1.99 KB	05/06/2020	f3ndot (Justin Bull)