Ruby - Feature #17312

New methods in Enumerable and Enumerator::Lazy: flatten, product, compact

11/09/2020 01:27 PM - zverok (Victor Shepelev)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

(The offspring of #16987, which was too vague/philosophical)

I propose to add to Enumerable and Enumerator::Lazy the following methods:

- compact
- product
- flatten

All of them can be performed with a one-way enumerator. All of them make sense for situations other than "just an array". All of them can be used for processing large sequences, and therefore meaningful to add to Lazy.

Related issues:

Related to Ruby - Feature #16987: Enumerator::Lazy vs Array methods	**Closed**

Associated revisions

Revision 68ea7720b367fe84da601cdbc61cb0d651c3221b - 01/02/2021 08:27 AM - nobu (Nobuyoshi Nakada)

NEWS: [Feature #17312] [ci skip]

Revision 68ea7720b367fe84da601cdbc61cb0d651c3221b - 01/02/2021 08:27 AM - nobu (Nobuyoshi Nakada)

NEWS: [Feature #17312] [ci skip]

Revision 68ea7720 - 01/02/2021 08:27 AM - nobu (Nobuyoshi Nakada)

NEWS: [Feature #17312] [ci skip]

History

#1 - 11/13/2020 04:32 PM - Dan0042 (Daniel DeLorme)

What would be the interaction between Array#flatten and Enumerable#flatten ?
It's a big compatibility problem if flatten recursively applies to any Enumerable object within an array.

```
x = Struct.new(:a).new(1)  #=> #<struct a=1>
[[[x]]].flatten            #=> [x] currently
Enumerable === x           #=> true
x.to_a                     #=> [1]
[[[x]]].flatten            #=> [1] would be a problem
```

#2 - 11/16/2020 08:24 AM - mame (Yusuke Endoh)

I think Enumerable#compact is trivial and maybe useful.

In regard to Enumrable#flatten, I agree with @Dan0042's concern. I think that it should flatten only Array elements, but it might look unnatural.

I'm unsure if Enumerable#product is useful. Its arguments are repeatedly iterated, so the arguments should be Arrays?

It would be good to separate tickets for each method, and a draft patch would be helpful for discussion.

#3 - 11/17/2020 03:50 PM - zverok (Victor Shepelev)

@mame (Yusuke Endoh) @Dan0042 Oh, you are right, starting to think from Enumerable::Lazy perspective I've missed a huge incompatibility introduced by flatten.

@mame (Yusuke Endoh) I'll split into several proposals+patches: Enumerable#compact, and, I am starting to think now, maybe Enumerator#flatten would make some sense. As for #product, I just added it for completeness (as a method which also can work with unidirectional enumeration), I can't from the top of my head remember if I needed it some time in the past.

#4 - 11/17/2020 04:14 PM - Dan0042 (Daniel DeLorme)

I understand the thinking behind #flatten; if ary.flatten is possible then why not ary.to_enum.flatten? It should be isomorphic. But even with Enumerator the recursive aspect still represents a compatibility problem. So as long as the behavior of Array#flatten is not modified I think all this is trivial to implement:

```
module Enumerable
  def compact(...); to_a.compact(...); end
  def product(...); to_a.product(...); end
  def flatten(...); to_a.flatten(...); end
end
```

edit: oops sorry, forgot the point was that Enumerator::Lazy#flatten should return a Enumerator::Lazy

#5 - 11/17/2020 04:28 PM - zverok (Victor Shepelev)

> But even with Enumerator the recursive aspect still represents a compatibility problem.

I am not sure about its severity, though. I mean, Universe is big and sure somewhere in it there should be a code which has an *array of enumerators* and then does flatten on them... But I am not sure there is much of this code in the wild.

I believe that this situation has the similar rarity class as the situation with code which does unless obj.respond_to?(:except) and will be broken by newly introduced Hash#except method... Like, *every* change is incompatibility for somebody, as https://xkcd.com/1172/ points, but Enumerator#flatten seems quite innocent.

> So as long as the behavior of Array#flatten is not modified I think all this is trivial to implement:

```
def flatten(...); to_a.flatten(...); end
```

Note that this ticket is a follow-up of #16987. What I interested in, is more usages for .lazy, and *eager* implementation of Enumerator::Lazy#flatten is definitely a no-go.

So, I actually *could* propose just Enumerator::Lazy#flatten, but it seems quite weird that lazy enumerator can be flattened, while regular one can't.

#6 - 11/18/2020 03:31 PM - p8 (Petrik de Heus)

I was really suprised that #last isn't implemented in Enumerable while #first is.

#7 - 11/18/2020 04:18 PM - zverok (Victor Shepelev)

@p8

> I was really suprised that #last isn't implemented in Enumerable while #first is.

It is natural.

That's because Enumerable is "uni-directional" (it is not guaranteed that you can iterate through it more than once, and there is no way to go back). Imagine this:

```
lines = File.each_line('foo.txt')
lines.last # -- if it worked, ALL the file is already read here, and you can't do anything reasonable with it
```

Also, last is "intuitively" cheap ("just give me last element, what's the problem?"), but as Enumerable relies on each, and each only, Enumerable#last would mean "go through entire each till it would be exhausted, and give the last value", which might be very pricey.

All the methods I am trying to propose are compatible with uni-directional #each

#8 - 11/19/2020 08:59 PM - p8 (Petrik de Heus)

@zverok (Victor Shepelev) Thanks for the explanation. That makes a lot of sense!

#9 - 11/20/2020 08:32 AM - matz (Yukihiro Matsumoto)

- Related to Feature #16987: Enumerator::Lazy vs Array methods added

#10 - 11/20/2020 09:16 AM - matz (Yukihiro Matsumoto)

My opinion for each proposed method.

- compact - OK, I can imagine use-cases too
- product - Negative; I concern about arguments (array or enumerable)

- flatten - Negative; I concern about types of elements (array of enumerable)

If you want product and flatten in Enumerable, submit separate issues to persuade me.

Matz.

#11 - 12/05/2020 11:44 AM - zverok (Victor Shepelev)

PR for #compact: https://github.com/ruby/ruby/pull/3851

#12 - 01/02/2021 08:28 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset git|68ea7720b367fe84da601cdbc61cb0d651c3221b.

NEWS: [Feature #17312] [ci skip]