Ruby - Feature #14593

Add `Enumerator#concat`

03/08/2018 03:37 PM - skalee (Sebastian Skalacki)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

I propose introducing an Enumerator#concat(other_enum1, other_enum2, ...) which returns an enumerator being a concatenation of self and passed arguments.

Expectation

```
enum1 = [1, 2, 3].each
enum2 = %w[a b c].each
enum3 = %i[X Y Z].each
concatenation = enum1.concat(enum2, enum3)
concatenation.kind_of?(Enumerator)
#=> true
concatenation.to_a
#=> [1, 2, 3, "a", "b", "c", :X, :Y, :Z]
concatenation.size
#=> 9

enum_without_size = Enumerator.new {}
enum_without_size.size
#=> nil
concatenation2 = enum1.concat(enum2, enum_without_size, enum3)
concatenation2.kind_of?(Enumerator)
#=> true
concatenation2.to_a
#=> [1, 2, 3, "a", "b", "c", :X, :Y, :Z]
concatenation2.size
#=> nil
```

Reasoning

Enumerators are generally useful. They allow to iterate over some data set without loading them fully into memory. They help separating data generation from its consumption. If enumerators are desirable, then enumerator concatenation is desirable as well.

Reference implementation

```
class Enumerator
  def concat(*enums)
    enumerators = [self, *enums]

    size = enumerators.reduce(0) do |acc, enum|
      s = enum.size
      break nil unless s
      acc + s
    end

    Enumerator.new(size) do |y|
      enumerators.each do |enum|
        enum.each { |item| y << item }
      end
    end
  end
end
```

Flat map one-liner

There's an answer on Stack Overflow suggesting a neat one-liner – https://stackoverflow.com/a/38962951/304175

```
enums.lazy.flat_map{|enum| enum.lazy }
```

It yields items correctly. However, it is not very idiomatic. Neither it implements #size method properly (see example below). For these reasons, I think that implementing Enumerator#concat is a better option.

```
enums = [enum1, enum2, enum3]
#=> [#<Enumerator: [1, 2, 3]:each>, #<Enumerator: ["a", "b", "c"]:each>, #<Enumerator: [:X, :Y, :Z
]:each>]
concatenation3 = enums.lazy.flat_map{|enum| enum.lazy }
#=> #<Enumerator::Lazy: #<Enumerator::Lazy: [#<Enumerator: [1, 2, 3]:each>, #<Enumerator: ["a", "b
", "c"]:each>, #<Enumerator: [:X, :Y, :Z]:each>]>:flat_map>
concatenation3.to_a
#=> [1, 2, 3, "a", "b", "c", :X, :Y, :Z]
concatenation3.size
#=> nil
```

Example use cases

Process 20 tweets/posts without fetching more than needed. Generate some example posts if less than 20 is available

```
enum_tweets = lazy_fetch_tweets_from_twitter(count: 20)
#=> Enumerator
enum_fb_posts = lazy_fetch_posts_from_facebook(count: 20)
#=> Enumerator
enum_example_posts = Enumerator.new { |y| loop { y << generate_random_post } }
#=> Enumerator
posts = enum_tweets.concat(enum_fb_posts).concat(enum_example_posts).take(20)
process(posts)
```

Perform a table union on large CSV files

```
csv1_enum = CSV.foreach("path/to/1.csv")
csv2_enum = CSV.foreach("path/to/2.csv")

csv1_enum.concat(csv2_enum).detect { |row| is_what_we_are_looking_for?(row) }
```

History

#1 - 03/08/2018 04:33 PM - shan (Shannon Skipper)

Just for fun I tried modifying the one-liner to add a lazily caluculated size:

```
enums.lazy.flat_map(&:lazy).to_enum { enums.sum(&:size) if enums.all?(&:size) }
```

That ^ pure Ruby implementation is a bit confusing and like the original one-liner it produces a lazy rather than eager enum.

#2 - 03/08/2018 04:51 PM - Hanmac (Hans Mackowiak)

the size object in Enumerator.new can be a proc too

that would make the call lazy?

#3 - 03/08/2018 06:02 PM - skalee (Sebastian Skalacki)

shan (Shannon Skipper) wrote:

> That ^ pure Ruby implementation is a bit confusing and like the original one-liner it produces a lazy rather than eager enum.

The reference implementation I have presented is meant to clarify what I mean by enumerator concatenation. I believe it can be implemented better, but it isn't important at this point.

When it comes to one-liner, it is quite long, and IMO not very readable. I consider adding Enumerator#concat as better solution.

#4 - 03/08/2018 09:15 PM - shan (Shannon Skipper)

> When it comes to one-liner, it is quite long, and IMO not very readable. I consider adding Enumerator#concat as better solution.

I totally agree the one-liner is strange, unwieldy and hard to read.