Ruby master - Feature #15144

Enumerator#chain

09/21/2018 07:00 PM - zverok (Victor Shepelev)

Status:	Closed
Priority:	Normal
Assignee:	knu (Akinori MUSHA)
Target version:	

Description

I am not sure I am not missing something, but...

```
[1, 2, 3].each.chain([3, 4, 5].each) # => Enumerator
```

...seem to be a useful pattern.

It especially shows itself in case of lazy enumerators, representing several long-calculated sequences, like something...

```
# just data from several sources, abstracted into enumerator, fetching it on demand
process = URLS.lazy.map(&Faraday.method(:get)))
  .chain(LOCAL_FILES.lazy.map(&File.method(:read)))
  .chain(FALLBACK_FILE.then.lazy.map(&File.method(:read)))
# with yield_self aka then we can even chain ONE value

process.detect { |val| found?(val) }
# uniformely search several sources (lazy-loading them) for some value

# tty-progressbar is able to work with enumerables:
bar = TTY::ProgressBar.new("[:bar]", total: URLS.count + LOCAL_FILES.count + 1)
bar.iterate(process).detect { |val| found?(val) }
# shows progress-bar for uniform process of detection
```

Prototype impl. is dead simple, of course:

```
class Enumerator
  def chain(*others)
    Enumerator.new { |y|
      [self, *others].each { |e| e.each { |v| y << v } }
    }
  end
end
```

Obviously, the effect could be reached with flat_map, but it seems "chaining" of iterations is pretty common and clear concept (and Google search for "ruby enumerator chain" shows people constantly ask about the way).

Related issues:

Related to Ruby master - Feature #709: Enumerator#+	**Rejected**

Associated revisions

Revision 045b0e54 - 11/24/2018 08:38 AM - knu (Akinori MUSHA)

Implement Enumerator#+ and Enumerable#chain [Feature #15144]

They return an Enumerator::Chain object which is a subclass of
Enumerator, which represents a chain of enumerables that works as a
single enumerator.

```
e = (1..3).chain([4, 5])
e.to_a #=> [1, 2, 3, 4, 5]

e = (1..3).each + [4, 5]
e.to_a #=> [1, 2, 3, 4, 5]
```

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@65949 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 65949 - 11/24/2018 08:38 AM - knu (Akinori MUSHA)

Implement Enumerator#+ and Enumerable#chain [Feature #15144]

They return an Enumerator::Chain object which is a subclass of Enumerator, which represents a chain of enumerables that works as a single enumerator.

```
e = (1..3).chain([4, 5])
e.to_a #=> [1, 2, 3, 4, 5]

e = (1..3).each + [4, 5]
e.to_a #=> [1, 2, 3, 4, 5]
```

Revision 65949 - 11/24/2018 08:38 AM - knu (Akinori MUSHA)

Implement Enumerator#+ and Enumerable#chain [Feature #15144]

They return an Enumerator::Chain object which is a subclass of Enumerator, which represents a chain of enumerables that works as a single enumerator.

```
e = (1..3).chain([4, 5])
e.to_a #=> [1, 2, 3, 4, 5]

e = (1..3).each + [4, 5]
e.to_a #=> [1, 2, 3, 4, 5]
```

History

#1 - 09/21/2018 10:57 PM - shevegen (Robert A. Heiler)

> Obviously, the effect could be reached with flat_map

I always found that name weird so ... assumingly that it is the same as .flatten.map, I always used the latter. :-)

(I don't recall flat_map offhand and I happily admit that I have not googled; I try to keep ruby so simple to need only few things if possible.)

> but it seems "chaining" of iterations is pretty common and clear concept (and Google search for "ruby enumerator chain" shows people constantly ask about the way).

Well, I think this may have more to do how to name something. It may be best to actually ask matz about the "chaining" here.

My personal opinion, which may be wrong, is that the term chaining is used here just to put method after method onto an object (e. g. send message after message to your object) - and have it perform the corresponding code defined in these methods. A bit like a stream of data through a pipe, a filter.

If this is a correct point of view (and I really don't know; we may have to ask matz), then there should not be a need to call it a "chain" - but most definitely to not use it as a word for a new or additional method, be it .chain() or .chaining(). But again, ultimately only matz knows.

By the way:

```
[1, 2, 3].each.chain([3, 4, 5].each)
```

The repetition of .each seems a bit awkward and the intention is also not very clear to me. But that is just my personal opinion; people write ruby code in very different ways. The above code looks alien to me too. :)

#2 - 10/10/2018 06:02 AM - knu (Akinori MUSHA)

- Related to Feature #709: Enumerator#+ added

#3 - 10/10/2018 06:18 AM - knu (Akinori MUSHA)

In today's developer meeting, Matz said Enumerator#+ would be OK to add, so I'm going to work on it first and then we'll think about an alias, and a constructor that takes many enumerators later.

#4 - 11/15/2018 09:17 PM - knu (Akinori MUSHA)

- Assignee set to matz (Yukihiro Matsumoto)

- Status changed from Open to Assigned

I'm working on this and the implementation of Enumerator#+(enum) and Enumerator#chain(*enums) is about to complete.

Matz, what do you think about the name "chain"? Python has chain(), Rust too has chain, and I cannot think of any better name.

#5 - 11/21/2018 06:26 AM - knu (Akinori MUSHA)

- File 0001-Implement-Enumerator-Chain-and-Enumerator-chain-Feat.patch added

I've written an initial implementation as attached:

- Enumerator.chain(*enums) to generate an enumerator chain of enums
- Enumerator#+(other) to generate an enumerator chain of [self, other]
- Enumerator#chain(*others) to generate an enumerator chain of [self, *others]

Some notes:

- The constructor is currently Enumerator::Chain.new(*enums) but it should probably be Enumerator::Chain.new(enums) to make it extensible to take an enumerable in general, with itertools.chain.from_iterable of Python in mind.
- Enumerator.chain(Enumerator.chain(e1, e2), e3) cannot be optimized to Enumerator.chain(e1, e2, e3) because it is expected that the intermediate object Enumerator.chain(e1, e2) receive a call for each when Enumerator.chain(Enumerator.chain(e1, e2), e3).each {…} is called.

#6 - 11/21/2018 08:10 AM - knu (Akinori MUSHA)

knu (Akinori MUSHA) wrote:

> - The constructor is currently Enumerator::Chain.new(*enums) but it should probably be Enumerator::Chain.new(enums) to make it extensible to take an enumerable in general, with itertools.chain.from_iterable of Python in mind.

This could be implemented as a different class if needed, because it would have little in common with an array-based chain.

#7 - 11/22/2018 07:36 AM - knu (Akinori MUSHA)

We got Matz's approval for adding Enumerable#chain (instead of Enumerator#chain) and Enumerator#+.

#8 - 11/22/2018 07:37 AM - knu (Akinori MUSHA)

- Assignee changed from matz (Yukihiro Matsumoto) to knu (Akinori MUSHA)

#9 - 11/24/2018 08:38 AM - knu (Akinori MUSHA)

- Status changed from Assigned to Closed

Applied in changeset trunk|r65949.

Implement Enumerator#+ and Enumerable#chain [Feature #15144]

They return an Enumerator::Chain object which is a subclass of
Enumerator, which represents a chain of enumerables that works as a
single enumerator.

```
e = (1..3).chain([4, 5])
e.to_a #=> [1, 2, 3, 4, 5]

e = (1..3).each + [4, 5]
e.to_a #=> [1, 2, 3, 4, 5]
```

Files

0001-Implement-Enumerator-Chain-and-Enumerator-chain-Feat.patch	13.3 KB	11/21/2018	knu (Akinori MUSHA)