Description

There are a number of methods in Enumerable that build an Array of results from the entire collection: map, select, take, etc.

I propose that the Enumerator class have its own implementations of these methods, which return another Enumerator. Enumerators can then be chained:

```ruby
seq.to_enum.map { ... }.select { ... }.take(...).each { |x| puts x }
```

This runs horizontally, that is, each element is processed left to right. No intermediate arrays are created, and it works happily with sequences of arbitrary length.

There are precedents for SomeClass#select behaving differently to Enumerable#select. For example, Hash#select returns a Hash. So I believe it would be reasonable for Enumerator to return another Enumerator.

You can then choose between array-building or lazy evaluation, depending on whether there is an Enumerator in the chain. Of course, the last Enumerator has to be turned into something useful, e.g. by calling to_a or each { ... }.

Normal

```ruby
res = (1..1_000_000).map { |x| x * 2 }.take(100)
```

Lazy

```ruby
res = (1..1_000_000).to_enum.map { |x| x * 2 }.take(100).to_a
```

I have attached a simple implementation of this for select, map, take and a new method skip. There are further methods like take_until, zip and so on which would also need to be implemented.

Related issues:

- Related to Ruby master - Feature #4653: [PATCH 1/1] new method Enumerable#rud...
  Rejected 05/08/2011
- Related to Ruby master - Feature #4890: Enumerable#lazy
  Closed 06/16/2011

History

#1 - 11/29/2008 04:29 PM - ko1 (Koichi Sasada)
- Assignee set to matz (Yukihiro Matsumoto)

```ruby
=begin
=end

#2 - 09/14/2010 04:51 PM - shyouhei (Shyouhei Urabe)
- Status changed from Open to Assigned

```ruby
=begin
=end

#3 - 11/02/2010 12:07 AM - rogerdpick (Roger Pack)
=begin
another option might be to add new methods called "e_select" or what not, to avoid changing current functionality.
=end
Can anyone take a feedback? If nothing, I have to close this ticket sadly.

There may be easy "first edition" solution to this. Facets has Denumerable/Denumerator and defer.

- https://github.com/rubyworks/facets/blob/master/lib/core/facets/denumerable.rb

Use this as starting point. Modify API as needed. Could start out as standard library until someone has time to translate to C.

Hello,

I think no one doubt if this feature is useful and actually needed. In fact, there are some proposals for the same (or similar) motivation.

- #4653 (focuses only map ?)
- #4890 (a new class Enumerable::Lazy)
- #5663 (focuses only select+map ?)

Especially, #5663 is recently discussed. So it is too early to close this ticket.

Once, I also created the similar proof-of-concept library [1]. It provides some methods like Enumerable#mapper, #selector, etc., which are Enumerator-version of corresponding Enumerable methods. Matz said in [2] that it can be accepted except the name convention ('er).


So it might be good to suggest another name convention. In #4653, some convention is proposed.

- mapper
- mapping
- map
- map_lz
- lazy_map
- enum_map
- map_enum

Matz himself suggested enum_* {enum_map, enum_select, ...}, though I don't like it because it is too long. I don't know if matz still like it.

--
Yusuke Endoh mame@tsg.ne.jp

Hi,

In message "Re: [ruby-core:42411] [ruby-trunk - Feature #708] Lazy Enumerator#select, Enumerator#map etc." on Wed, 8 Feb 2012 04:22:24 +0900, Yusuke Endoh mame@tsg.ne.jp writes:

| I think no one doubt if this feature is useful and actually needed. |
| In fact, there are some proposals for the same (or similar) motivation. |
| - #4653 (focuses only map ?) |
| - #4890 (a new class Enumerable::Lazy) |
| - #5663 (focuses only select+map ?) |
| Especially, #5663 is recently discussed. So it is too early to close this ticket. |

I vote for #4890.
Konnichiwa matz,

Which term do you prefer for the method #lazy or #defer?

Also organization (regardless of actual names), there is Enumerable namespace, e.g.

    module Enumerable
    class Deferred < Enumerator

Or toplevel namespace, e.g.

    class Denumerator < Enumerator

And note that previously mentioned Denumerable module allows option of mixin independent of Enumerable. If is organized like:

    module Denumerable
    ...
    end

    class Denumerator
    include Denumerable
    end

If you prefer "lazy" term I supposed the names for these would be "LazyEnumerable" and "LazyEnumerator" -- b/c I don't think "Lazierable" and "Lazierator" are going to cut it ;-)