Ruby trunk - Feature #4890

Enumerable#lazy

06/16/2011 07:23 PM - yhara (Yutaka HARA)

Status:	Closed
Priority:	Normal
Assignee:	yhara (Yutaka HARA)
Target version:	2.0.0

Description

=begin
= Example
Print first 100 primes which are in form of n^{2+1}

```
require 'prime'
INFINITY = 1.0 / 0
p (1..INFINITY).lazy.map{|n| n**2+1}.select{|m| m.prime?}.take(100)
```

(Example taken from enumerable_lz; thanks @antimon2)

= Description

Enumerable#lazy returns an instance of Enumerable::Lazy.
This is the only method added to the existing bulit-in classes.

Lazy is a subclass of Enumerator, which includes Enumerable.
So you can call any methods of Enumerable on Lazy, except methods like
map, select, etc. are redefined as 'lazy' versions.

= Sample implementation

((URL:https://gist.github.com/1028609))
(also attached to this ticket)

=end

Related issues:

Related to Ruby trunk - Feature #4653: [PATCH 1/1] new method Enumerable#rude...	**Rejected**	**05/08/2011**
Related to Ruby trunk - Feature #708: Lazy Enumerator#select, Enumerator#map ...	**Rejected**	**11/03/2008**
Related to Ruby trunk - Bug #7248: Shouldn't Enumerator::Lazy.new be private?	**Closed**	**10/31/2012**

Associated revisions

Revision 0b2c4f43 - 03/08/2012 03:30 PM - nobu (Nobuyoshi Nakada)

- enumerator.c: add Enumerable#lazy. based on the patch by Innokenty Mikhailov at https://github.com/ruby/ruby/pull/101 [ruby-core:37164] [Feature #4890]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@34951 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 34951 - 03/08/2012 03:30 PM - nobu (Nobuyoshi Nakada)

- enumerator.c: add Enumerable#lazy. based on the patch by Innokenty Mikhailov at https://github.com/ruby/ruby/pull/101 [ruby-core:37164] [Feature #4890]

Revision 34951 - 03/08/2012 03:30 PM - nobu (Nobuyoshi Nakada)

- enumerator.c: add Enumerable#lazy. based on the patch by Innokenty Mikhailov at https://github.com/ruby/ruby/pull/101 [ruby-core:37164] [Feature #4890]

Revision 34951 - 03/08/2012 03:30 PM - nobu (Nobuyoshi Nakada)

- enumerator.c: add Enumerable#lazy. based on the patch by Innokenty Mikhailov at https://github.com/ruby/ruby/pull/101 [ruby-core:37164]

[Feature #4890]

Revision 34951 - 03/08/2012 03:30 PM - nobu (Nobuyoshi Nakada)

- enumerator.c: add Enumerable#lazy. based on the patch by Innokenty Mikhailov at https://github.com/ruby/ruby/pull/101 [ruby-core:37164] [Feature #4890]

Revision 34951 - 03/08/2012 03:30 PM - nobu (Nobuyoshi Nakada)

- enumerator.c: add Enumerable#lazy. based on the patch by Innokenty Mikhailov at https://github.com/ruby/ruby/pull/101 [ruby-core:37164] [Feature #4890]

Revision 34951 - 03/08/2012 03:30 PM - nobu (Nobuyoshi Nakada)

- enumerator.c: add Enumerable#lazy. based on the patch by Innokenty Mikhailov at https://github.com/ruby/ruby/pull/101 [ruby-core:37164] [Feature #4890]

History

#1 - 06/17/2011 12:01 PM - mrkn (Kenta Murata)

I think it should be an embedded feature of Enumerable.
I wish it is accepted for Ruby 1.9.3.

#2 - 10/31/2011 11:48 AM - shugo (Shugo Maeda)

- Assignee set to matz (Yukihiro Matsumoto)

- Target version changed from 2.0.0 to Next Major

Hi,

Yutaka HARA wrote:

```
require 'prime'
INFINITY = 1.0 / 0
p (1..INFINITY).lazy.map{|n| n**2+1}.select{|m| m.prime?}.take(100)
```

I prefer Enumerable#lazy to Enumerable#lazy_map or rude_map
because Enumerable#lazy can be used polymorphically as Enumerable.
I hope that Enumerable#lazy will be included in Ruby 2.0.
Is there any reason why it shouldn't be included in Ruby 2.0?

FYI, Scala has a similar method called view. I don't know whether
the name view is better than lazy because I'm not a native English speaker.

```
scala> val v = Vector(1 to 10: *)
v: scala.collection.immutable.Vector[Int] = Vector(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
scala> v.view.map( + 1).map(_ * 2)
res0: scala.collection.SeqView[Int,Seq[]] = SeqViewMM(...)
scala> v.view.map( + 1).map(_ * 2).force
res1: Seq[Int] = Vector(4, 6, 8, 10, 12, 14, 16, 18, 20, 22)
```

In Ruby, results are always arrays, so we can use to_a instead of force.
But the name force looks better than to_a for me.

Scala also has a method called withFilter, which is something like lazy_select.

```
scala> v.withFilter(_ % 2 == 0).map(_ * 1)
res12: scala.collection.immutable.Vector[Int] = Vector(2, 4, 6, 8, 10)
```

I guess it is introduced for the for expression, which is equivalent to list comprehension in Haskell.

#3 - 10/31/2011 11:58 AM - shugo (Shugo Maeda)

Shugo Maeda wrote:

> FYI, Scala has a similar method called view. I don't know whether
> the name view is better than lazy because I'm not a native English speaker.
> (snip)
> In Ruby, results are always arrays, so we can use to_a instead of force.
> But the name force looks better than to_a for me.

If the method force is added, delay sounds more natural than lazy or view for me.

p (1..10).delay.map{|i| i+1}.map{|i| i+2}.force

That's just off the top of my head, and it may be confusing.

#4 - 10/31/2011 12:04 PM - trans (Thomas Sawyer)

Please consider:

- http://github.com/rubyworks/facets/blob/master/lib/core/facets/enumerable/defer.rb
- http://github.com/rubyworks/facets/blob/master/lib/core/facets/denumerable.rb

#5 - 10/31/2011 02:16 PM - knu (Akinori MUSHA)

The idea of laziness can be or should be sublimated into a revamp of the Enumerable framework.

- Make every Enumerable object act like an immutable array by defining Array methods ([], size/length, +, -, join, etc.)
- Then make some of the methods returning an array (map, select, zip, etc.) return an Enumerable(::Lazy or object of the receiver's class)

This way we can "hide" the lazy class and basically make everything lazy where appropriate.

#6 - 10/31/2011 02:27 PM - knu (Akinori MUSHA)

One thing I should comment on the sample implementation: Enumerable::Lazy should not inherit from Enumerator which includes Enumerable.

Enumerable::Lazy has the method set similar to Enumerable but their behavior is different. Enumerable's methods defined by third party libraries (that may be non-lazy) should not automatically be exposed to Enumerable::Lazy, who must want to offer (redefine) lazy counterparts of those methods soon or later.

#7 - 10/31/2011 02:39 PM - shugo (Shugo Maeda)

Akinori MUSHA wrote:

> The idea of laziness can be or should be sublimated into a revamp of the Enumerable framework.
>
> - Make every Enumerable object act like an immutable array by defining Array methods ([], size/length, +, -, join, etc.)
> - Then make some of the methods returning an array (map, select, zip, etc.) return an Enumerable(::Lazy or object of the receiver's class)

It breaks compatibility, so may not be acceptable in Ruby 2.0.
Or It may be acceptable depending on Matz's definition of "100% compatible."

> This way we can "hide" the lazy class and basically make everything lazy where appropriate.

It may be better to specify explicitly what is lazy and what is not.
In Ruby the difference is more important than functional languages.

#8 - 10/31/2011 05:52 PM - knu (Akinori MUSHA)

My suggestion is to turn Enumerator into a lazy stream and make every Enumerable class including the new Enumerator operate on the receiver's class wherever appropriate instead of fixing Array as result container.
For example, Hash#select would return a hash, Set#reject would return a set, and so on. If the result of a method does not fit in the receiver class (Set#sort may be a good example) it can return an enumerator or an inevitable array instead.

Adding extra methods to Enumerable/Enumerator is just another story for keeping backward compatibility as much as possible.

I do not insist on 2.0 as the target at all. I just feel that we can work out a better, seamless solution than the plugin approach that can be accomplished outside of the core.

#9 - 10/31/2011 08:23 PM - greck (Artem Vorozhtsov)

I prefer to_lazy and to_a instead of delay and force

PS:
Here some student's code from
http://rails.vsevteme.ru/2009/04/20/samorazvitie/10-metaprogramming-patterns-15-kyu-lenivye-konteynery:

```
Emumerator.module_eval do
б═б═def to_lazy
б═б═б═б═LazyContainer.new(this)
б═б═end
end
```

container.map{|i| f(i)}.select{|i| g(i)}.uniq.each {|i| puts i}

module LazyEnumerable
include Enumerable

ZERO_PIPE_LAMBDA

#10 - 11/01/2011 10:12 AM - shugo (Shugo Maeda)

Akinori MUSHA wrote:

> My suggestion is to turn Enumerator into a lazy stream and make every Enumerable class including the new Enumerator operate on the receiver's class wherever appropriate instead of fixing Array as result container.

It sounds fine for me, but I don't think Enumerable (or Enumerator) should have methods of Array such as [] and size.

> For example, Hash#select would return a hash, Set#reject would return a set, and so on. If the result of a method does not fit in the receiver class (Set#sort may be a good example) it can return an enumerator or an inevitable array instead.

FYI, Hash#select returns a Hash in Ruby 1.9.
Before Ruby 1.9 Matz rejected requests to change the resulting values of Enumerable methods,
but he might have changed his mind.
Anyway, I'd like to hear his opinion.

#11 - 11/01/2011 11:57 AM - shugo (Shugo Maeda)

One more question.

Shugo Maeda wrote:

> Akinori MUSHA wrote:
>
> > My suggestion is to turn Enumerator into a lazy stream and make every Enumerable class including the new Enumerator operate on the receiver's class wherever appropriate instead of fixing Array as result container.
>
> It sounds fine for me, but I don't think Enumerable (or Enumerator) should have methods of Array such as [] and size.

How about to add Enumerable#defer that returns a lazy version of Enumerator as a transition step in Ruby 2.0?
If Enumerator gets lazy in Ruby 3.0, Enumerable#defer can be changed to be just an alias of to_enum.

#12 - 11/01/2011 01:08 PM - knu (Akinori MUSHA)

> It sounds fine for me, but I don't think Enumerable (or Enumerator) should have methods of Array such as [] and size.

It'll be OK if once we decide we don't care too much about backward compatibility in 3.0.
Though I think it would be nice if we can add attributes (aspects) like indexability and finiteness so the wrapper module (Enumerable) can take advance of them and enable [] and size as appropriate.

> How about to add Enumerable#defer that returns a lazy version of Enumerator as a transition step in Ruby 2.0?
> If Enumerator gets lazy in Ruby 3.0, Enumerable#defer can be changed to be just an alias of to_enum.

Is defer the new name for lazy in this proposal?

#13 - 11/01/2011 02:53 PM - shugo (Shugo Maeda)

Akinori MUSHA wrote:

> > It sounds fine for me, but I don't think Enumerable (or Enumerator) should have methods of Array such as [] and size.
>
> It'll be OK if once we decide we don't care too much about backward compatibility in 3.0.
> Though I think it would be nice if we can add attributes (aspects) like indexability and finiteness so the wrapper module (Enumerable) can take advance of them and enable [] and size as appropriate.

It's confusing, isn't it?

If your proposal is accepted, I want Scala-like force, which returns an instance of the original collection class.

scala> List(1,2,3).view.map(_ + 1).filter(_ % 2 == 0).force
res0: Seq[Int] = List(2, 4)

scala> Set(1,2,3).view.map(_ + 1).filter(_ % 2 == 0).force
res1: Iterable[Int] = Set(2, 4)

So there is no reason to treat Array specially, except compatibility reason.
If you need an Array, you can call to_a explicitly.

> How about to add Enumerable#defer that returns a lazy version of Enumerator as a transition step in Ruby 2.0?
> If Enumerator gets lazy in Ruby 3.0, Enumerable#defer can be changed to be just an alias of to_enum.

Is defer the new name for lazy in this proposal?

The name may be changed. I'd like to hear Matz's opinion.

#14 - 11/02/2011 02:23 AM - Anonymous

Â p (1..10).delay.map{|i| i+1}.map{|i| i+2}.force

I assume this uses "lazy map" just for the first map, is that correct?
-r

#15 - 11/02/2011 04:10 AM - trans (Thomas Sawyer)

No, the #force in that example produces the final result. (Note, #to_a should suffice, no need for a special method.)

#16 - 11/02/2011 06:29 AM - Anonymous

> No, the #force in that example produces the final result. (Note, #to_a should suffice, no need for a special method.)

So it basically collects the various blocks, then applies them when
#force is called?

#17 - 11/02/2011 07:17 AM - trans (Thomas Sawyer)

Yes, in a sense. But it also is more efficient and can handle iterating over infinities if the end result itself is finite.

#18 - 02/13/2012 08:16 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee changed from matz (Yukihiro Matsumoto) to yhara (Yutaka HARA)

- Target version changed from Next Major to 2.0.0

Congrats, this proposal had an affirmative vote from matz! (See #708)

To avoid bikeshed discussion, I propose that we tentatively commit an
implementation as OP proposed.

I know there are still some issues on the API detail.
But let's discuss about the remained issues with the actual example.
Don't worry, we can change the API until this August.

Yutaka, could you make a patch in C? Or anyone?

--
Yusuke Endoh mame@tsg.ne.jp

#19 - 02/14/2012 12:36 AM - trans (Thomas Sawyer)

Actually, is there any reason why Enumerator's usual methods themselves aren't lazy? Is it necessary to have both? If they were lazy then the notation would be pretty simple:

[1,2,3].each.select{ |e| ... }.map{ |e| .... }.to_a

No special method needed, as #each would do the job.

#20 - 02/14/2012 04:50 AM - jballanc (Joshua Ballanco)

I think the main difference is that current Enumerable#{map|select|inject} use yield semantics with the block. This is not always conducive to lazy evaluation:

```
class AlwaysThree
   include Enumerable
   def each
     yield 3
   end
end

AlwaysThree.new.each { |i| puts i } #=> 3
AlwaysThree.new.map { |i| i**2 } #=> [9]
```

I realize that's a trivial example, but I think lazy enumerables will require generator semantics. i.e. Instead of everything being based around #each, it should be based around #next.

#21 - 02/14/2012 05:27 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Thomas, I think it only makes sense for enumerables to be lazy if you call take(n) on it.

Usually, you want all elements when calling any of those methods without explicitly marking them as lazy.

#22 - 02/14/2012 05:47 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Thomas, continuing, if the "force" approach is taken instead of take(n), you'll still have to call it (or call each/select/to_a) in the end, as oppose to what happens if you just call map/collect, for example.

It can also lead to unexpected behavior. For example, on this case, the side-effect won't happen on some elements:

items.map {|item|
this_should_always_be_done_for_each(item)
}.select{item.index == 2}

If "items" is lazy by default, "this_should_always_be_done_for_each" won't be called for all "items" unless only the last one has index 2.

#23 - 02/14/2012 05:54 AM - trans (Thomas Sawyer)

@Rodrigo I'm not suggesting Enumerable methods be lazy. I asking about Enumerator methods being lazy. So first you have to convert to an Enumerator.

items.to_enum.map{ ... }

As it happens, enumerable methods without a block become enumerators, so

items.each.map{ ... }

Would be the same.

Just calling items.map would not be any different then it is now.

#24 - 02/14/2012 05:58 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Oh, I see. I didn't know about Enumerators. If I understood what they are correctly, then I agree with you that they should be lazy implicitly.

#25 - 02/14/2012 09:23 AM - matz (Yukihiro Matsumoto)

Hi,

In message "Re: [ruby-core:42556] [ruby-trunk - Feature #4890] Enumerable#lazy"
on Tue, 14 Feb 2012 00:36:10 +0900, Thomas Sawyer transfire@gmail.com writes:

|Actually, is there any reason why Enumerator's usual methods themselves aren't lazy? Is it necessary to have both? If they were lazy then the notation would be pretty simple:
|
| [1,2,3].each.select{ |e| ... }.map{ |e| .... }.to_a
|
|No special method needed, as #each would do the job.

Compatibility. I don't want to break tons of programs that expect
arrays without calling to_a.

```
                        matz.
```

#26 - 02/25/2012 08:07 AM - gregolsen (Innokenty Mikhailov)

Yusuke Endoh wrote:

> Yutaka, could you make a patch in C? Or anyone?

Hi,

I've come up with patch in C - just two lazy methods added so far: map and select.
Please, see this PR for more info https://github.com/ruby/ruby/pull/100

The idea is very simple - block that passed to lazy method (select or map) is converted to Proc and stored in enumerator itself.
When next element requested - all Proc objects are called for this value and the result returned. Proc#call result handling depends on proc_entry type.

Let me know if it makes any sense (if true - I can come up with additional lazy methods).

Thanks.

#27 - 03/03/2012 09:41 PM - gregolsen (Innokenty Mikhailov)

I've faced problems while trying to push this idea further
so I came up with the straight C implementation of ruby code, submitted by Yutaka HARA.

Please, see this new PR https://github.com/ruby/ruby/pull/101

#28 - 03/04/2012 09:48 PM - alexeymuranov (Alexey Muranov)

Rodrigo Rosenfeld Rosas wrote:

> Thomas, I think it only makes sense for enumerables to be lazy if you call take(n) on it.
>
> Usually, you want all elements when calling any of those methods without explicitly marking them as lazy.

I disagree, i think lazy versions of all methods that return arrays are useful, to avoid generating intermediate arrays and to be able to work with infinite collections.

Alexey.

#29 - 03/09/2012 12:30 AM - nobu (Nobuyoshi Nakada)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r34951.
Yutaka, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- enumerator.c: add Enumerable#lazy. based on the patch by Innokenty Mikhailov at https://github.com/ruby/ruby/pull/101 [ruby-core:37164] [Feature #4890]

Files

lazy.rb	3.85 KB	06/16/2011	yhara (Yutaka HARA)