Ruby - Bug #6158

Enumerator::Lazy#take: should it be lazy?

03/16/2012 08:49 PM - Eregon (Benoit Daloze)

Status:	Closed		
Priority:	Normal		
Assignee:	shugo (Shugo Maeda)		
Target version:			
ruby -v:	ruby 2.0.0dev (2012-03-15 trunk 35042) [x86_64-darwin10.8.0]	**Backport:**	

Description

Hello,

I noticed #take is now defined on Enumerator::Lazy, and it has lazy behavior:

```
(1..1000).lazy.select(&:even?).take 3 # => #<Enumerator::Lazy ...>
```

I would expect #take to not be lazy and always produce an Array, like the original Enumerable#take does.
I think many rubyists would expect that too.

Do you have a use case for a lazy #take ? I can't find one right now.

I've seen #6152, although I can't read it.
I think lazy.take(5) should be equal to lazy.first(5) and it reads better.

Associated revisions

Revision 831898bd9971ec5997267226417ce0ce6f76ecd7 - 03/19/2012 06:41 AM - shugo (Shugo Maeda)

- enumerator.c (enumerable_lazy): add an example of take and first
  to the documentation. [ruby-core:43344] [Bug #6158]
  add the description of the behavior when a block is given to zip
  or cycle.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@35090 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 831898bd - 03/19/2012 06:41 AM - shugo (Shugo Maeda)

- enumerator.c (enumerable_lazy): add an example of take and first
  to the documentation. [ruby-core:43344] [Bug #6158]
  add the description of the behavior when a block is given to zip
  or cycle.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@35090 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 03/16/2012 11:22 PM - shugo (Shugo Maeda)

- Status changed from Open to Assigned

- Assignee set to shugo (Shugo Maeda)

Hello,

Benoit Daloze wrote:

> I noticed #take is now defined on Enumerator::Lazy, and it has lazy behavior:
>
> ```
> (1..1000).lazy.select(&:even?).take 3 # => #<Enumerator::Lazy ...>
> ```
>
> I would expect #take to not be lazy and always produce an Array, like the original Enumerable#take does.
> I think many rubyists would expect that too.
>
> Do you have a use case for a lazy #take ? I can't find one right now.

At first, I had the same opinion, but lazy take is sometimes useful.
For example, the following code consume much memory if take is not lazy.

sum = e.lazy.take(1000000).inject(:+)

Scheme's stream-take, which is equivalent to Enumerator::Lazy#take, is also lazy.

gosh> (use util.stream)
#
gosh> (stream-take (stream-iota -1 3 2) 3)
#<promise(stream) 0x9e4fc80>
gosh> (stream->list (stream-take (stream-iota -1 3 2) 3))
(3 5 7)

> I've seen #6152, although I can't read it.
> I think lazy.take(5) should be equal to lazy.first(5) and it reads better.

I think it's good to have both the eager and lazy version of take.

#2 - 03/17/2012 12:55 AM - trans (Thomas Sawyer)

Both "takes" on this seem valid. Could there be a special method similar to take for de-lazying and taking a segment of the enumeration? e.g. maybe #pinch ?

```
e.lazy.pinch(3)
```

```
e.lazy.pinch(4..6)
```

```
e.lazy.pinch(2, 4)
```

#3 - 03/17/2012 07:36 AM - gregolsen (Innokenty Mikhailov)

- File lazy_bang.diff added

=begin
Same for me - at first was thinking about having take evaluating makes sense.
But having ability to continue building the lazy chain looks more logical.

Having something like #pinch totally makes sense.

However I'm thinking about more radical solution: having methods with bang (!) always evaluate.

Like (({[1,2,3,4,5].lazy.map! { |x| x * 10 }})) will return the evaluated result.
This will solve the problem of calling (({to_a})) manually.
It might be confusing at first but then I think people will love that feature.
What do you think?
=end

#4 - 03/17/2012 10:02 AM - matz (Yukihiro Matsumoto)

use #first. map! etc. do not conform other bang method naming convention.

#5 - 03/17/2012 10:35 PM - Eregon (Benoit Daloze)

matz:

> use #first. map! etc. do not conform other bang method naming convention.

Thank you for taking the decision.

Shugo:
I understand the use, so I also think it's worth having some form of lazy #take.

However, I think many would expect #take to be eager, so it should be very well documented in the Enumerator::Lazy class documentation (with an example of #take and #first).

Thomas Sawyer:

> Both "takes" on this seem valid.
> You just made my day.

> Could there be a special method similar to take for de-lazying and taking a segment of the enumeration? e.g. maybe #pinch ?
> Sounds interesting, could you open a new issue?

Innokenty Mikhailov:

> However I'm thinking about more radical solution: having methods with bang (!) always evaluate.

I like it, it reminds me of #force, the alias of #to_a. And it has some notion of "I want it now, please evaluate the result".

But as matz said, the naming convention is not respected, and defining all these methods simply to avoid an explicit method call is probably not worth it.

#6 - 03/19/2012 03:41 PM - shugo (Shugo Maeda)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r35090.
Benoit, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- enumerator.c (enumerable_lazy): add an example of take and first
  to the documentation. [ruby-core:43344] [Bug #6158]
  add the description of the behavior when a block is given to zip
  or cycle.

Files

lazy_bang.diff	1.11 KB	03/17/2012	gregolsen (Innokenty Mikhailov)