Ruby master - Bug #7696

Lazy enumerators with state can't be rewound

01/15/2013 06:45 AM - marcandre (Marc-Andre Lafortune)

Status:	Closed		
Priority:	Normal		
Assignee:	matz (Yukihiro Matsumoto)		
Target version:	2.1.0		
ruby -v:	r38800	**Backport:**	

Description

The 4 lazy enumerators requiring internal state, i.e. {take|drop}{_while}, don't work as expected after a couple next and a call to rewind.

For example:

```
e=(1..42).lazy.take(2)
e.next # => 1
e.next # => 2
e.rewind
e.next # => 1
e.next # => StopIteration: iteration reached an end, expected 2
```

This is related to #7691; the current API does not give an easy way to handle state.

Either there's a dedicated callback to rewind, or data must be attached to the yielder.

Related issues:

Related to Ruby master - Bug #7691: 3 bugs with Lazy enumerators with state	**Closed**	**01/14/2013**
Related to Ruby master - Feature #8840: Yielder#state	**Rejected**	

Associated revisions

Revision de0e8876 - 01/24/2013 06:22 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#take
  [bug #7696]
- test/ruby/test_lazy_enumerator.rb: test for above

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38920 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 38920 - 01/24/2013 06:22 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#take
  [bug #7696]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38920 - 01/24/2013 06:22 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#take
  [bug #7696]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38920 - 01/24/2013 06:22 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#take
  [bug #7696]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38920 - 01/24/2013 06:22 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#take
  [bug #7696]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38920 - 01/24/2013 06:22 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#take
  [bug #7696]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38920 - 01/24/2013 06:22 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#take
  [bug #7696]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision bcbeb5d0 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop_while
  [bug #7696] [bug #7691]
- test/ruby/test_lazy_enumerator.rb: test for above

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38921 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 38921 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop_while
  [bug #7696] [bug #7691]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38921 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop_while
  [bug #7696] [bug #7691]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38921 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop_while
  [bug #7696] [bug #7691]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38921 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop_while
  [bug #7696] [bug #7691]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38921 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop_while
  [bug #7696] [bug #7691]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38921 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop_while
  [bug #7696] [bug #7691]
- test/ruby/test_lazy_enumerator.rb: test for above

Revision 9d94a1a5 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop
  [bug #7696] [bug #7691]
- test/ruby/test_lazy_enumerator.rb: test for above

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38922 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 38922 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38922 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38922 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38922 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38922 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38922 - 01/24/2013 06:23 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#drop
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 41d6ba87 - 01/24/2013 06:24 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#zip

[bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38923 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 38923 - 01/24/2013 06:24 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#zip
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38923 - 01/24/2013 06:24 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#zip
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38923 - 01/24/2013 06:24 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#zip
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38923 - 01/24/2013 06:24 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#zip
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38923 - 01/24/2013 06:24 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#zip
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

Revision 38923 - 01/24/2013 06:24 AM - marcandre (Marc-Andre Lafortune)

- enumerator.c: Fix state handling for Lazy#zip
  [bug #7696] [bug #7691]

- test/ruby/test_lazy_enumerator.rb: test for above

History

#1 - 01/16/2013 05:39 PM - shugo (Shugo Maeda)

marcandre (Marc-Andre Lafortune) wrote:

> The 4 lazy enumerators requiring internal state, i.e. {take|drop}{_while}, don't work as expected after a couple next and a call to rewind.
>
> For example:
>
> ```
> e=(1..42).lazy.take(2)
> e.next # => 1
> e.next # => 2
> e.rewind
> e.next # => 1
> e.next # => StopIteration: iteration reached an end, expected 2
> ```
>
> This is related to #7691; the current API does not give an easy way to handle state.

#7691 is the basically same issue as #6142.

Do you have a draft API spec in mind?
It might be too late to introduce a new API for 2.0.0, though.

#2 - 01/17/2013 02:01 AM - marcandre (Marc-Andre Lafortune)

I had not see #6142. Yes, it's the same.

1) Add attr_accessor to Yielder and document the fact that the yielder is unique to a single enumeration run.

```
def drop(n)
  n = Backports::coerce_to_int(n)
  Lazy.new(self) do |yielder, *values|
    yielder.memo ||= n
    if yielder.memo > 0
      yielder.memo -= 1
    else
      yielder.yield(*values)
    end
  end
end
```

2) Optional named argument prepare for Lazy.new to be called before each:

```
def drop(n)
  remain = nil
  Lazy.new(self, prepare: ->{remain = n}) do |yielder, *values|
    if remain > 0
      remain -= 1
    else
      yielder.yield(*values)
    end
  end
end
```

3) Add a method 'prepare' method to Enumerator::Lazy and encourage subclassing. prepare would be called before each

```
class LazyDropper < Enumerator::Lazy
  def initialize(obj, n)
    @n = n
    super do |yielder, *values|
      if @remain > 0
        @remain -= 1
      else
        yielder.yield(*values)
      end
   end

   def prepare
     @remain = @n
   end
```

```
end
```

MRI should use the same kind of mechanism for the enums requiring state

#3 - 01/17/2013 02:02 AM - marcandre (Marc-Andre Lafortune)

Oh oh.

Let me add a third example to problems of handling states:

```
e = (1..6).lazy.drop(3)
e.flat_map{e}.force # => [*(1..6)]*3, should be  [*(1..3)]*3
```

I believe only the first solution would work for this.

#4 - 01/17/2013 03:11 AM - marcandre (Marc-Andre Lafortune)

Here's a patch for take. Does it look ok?

diff --git a/enumerator.c b/enumerator.c
index b65712f..1522a3f 100644
--- a/enumerator.c
+++ b/enumerator.c
@@ -105,7 +105,7 @@
VALUE rb_cEnumerator;
VALUE rb_cLazy;
static ID id_rewind, id_each, id_new, id_initialize, id_yield, id_call, id_size;
-static ID id_eqq, id_next, id_result, id_lazy, id_receiver, id_arguments, id_method, id_force;
+static ID id_eqq, id_next, id_result, id_lazy, id_receiver, id_arguments, id_memo, id_method, id_force;
static VALUE sym_each, sym_cycle;

VALUE rb_eStopIteration;
@@ -1641,14 +1641,18 @@ lazy_zip(int argc, VALUE *argv, VALUE obj)
static VALUE
lazy_take_func(VALUE val, VALUE args, int argc, VALUE *argv)
{

- NODE *memo = RNODE(args);
- long remain;
- VALUE memo = rb_ivar_get(argv[0], id_memo);
- if (NIL_P(memo)) {
- memo = args;

- }

  rb_funcall2(argv[0], id_yield, argc - 1, argv + 1);

- if (--memo->u3.cnt == 0) {
- [redacted]
- if ((remain = NUM2LONG(memo)-1) == 0) {
  return Qundef;
  }
  else {
- [redacted]
  return Qnil;

  }
  }
  @@ -1666,7 +1670,6 @@ lazy_take_size(VALUE lazy)
  static VALUE
  lazy_take(VALUE obj, VALUE n)
  {

- NODE *memo;
  long len = NUM2LONG(n);
  int argc = 1;
  VALUE argv[3];
  @@ -1680,9 +1683,8 @@ lazy_take(VALUE obj, VALUE n)
  argv[2] = INT2NUM(0);
  argc = 3;
  }

- memo = NEW_MEMO(0, len, len);
  return lazy_set_method(rb_block_call(rb_cLazy, id_new, argc, argv,
- [redacted]
- [redacted]
  rb_ary_new3(1, n), lazy_take_size);

  }

@@ -1955,6 +1957,7 @@ Init_Enumerator(void)
id_eqq = rb_intern("===");
id_receiver = rb_intern("receiver");
id_arguments = rb_intern("arguments");

- id_memo = rb_intern("memo");
  id_method = rb_intern("method");
  id_force = rb_intern("force");
  sym_each = ID2SYM(id_each);
  diff --git a/test/ruby/test_lazy_enumerator.rb b/test/ruby/test_lazy_enumerator.rb
  index acd4843..35e92c9 100644
  --- a/test/ruby/test_lazy_enumerator.rb
  +++ b/test/ruby/test_lazy_enumerator.rb
  @@ -243,6 +243,23 @@ class TestLazyEnumerator < Test::Unit::TestCase
  assert_equal((1..5).to_a, take5.force, bug6428)
  end

- def test_take_nested
- bug7696 = '[ruby-core:51470]'
- a = Step.new(1..10)
- take5 = a.lazy.take(5)
- assert_equal([*(1..5)]*5, take5.flat_map{take5}.force, bug7696)
- end
  +
- def test_take_rewound
- bug7696 = '[ruby-core:51470]'
- e=(1..42).lazy.take(2)
- assert_equal 1, e.next
- assert_equal 2, e.next
- e.rewind
- assert_equal 1, e.next
- assert_equal 2, e.next
- end
  +
  def test_take_while
  a = Step.new(1..10)
  assert_equal(1, a.take_while {|i| i < 5}.first)

#5 - 01/19/2013 10:33 AM - shugo (Shugo Maeda)

marcandre (Marc-Andre Lafortune) wrote:

> Here's a patch for take. Does it look ok?

Does it work well for zip?
I wonder how arguments are rewound.

#6 - 01/20/2013 02:33 AM - marcandre (Marc-Andre Lafortune)

The same idea will work for zip too; the arguments must be converted to enumerators only when yielder.memo is not set, i.e. every new yielder.

The arguments are never really rewound, but the first yielder holding them will not be reused after enum.rewind; a new yielder is given.
I haven't checked if that's the current behavior in other implementations, but if yielder is to be the state holder, that's the way it should work.

```
enum = (1..2).lazy.zip(1..2)
enum.next # => yielder.memo was nil, so (1..2).each is called and stored in the memo
enum.rewind # => the original yielder is discarded
enum.next # => the second yielder.memo is nil, so (1..2).each is called again and stored in the memo
```

I'm using the same idea in my backports gem to implement this in pure Ruby:

`https://github.com/marcandre/backports/blob/master/lib/backports/2.0.0/enumerator/lazy.rb`

The only other possibility that would work is to pass yield extra 'state' argument when required, like:

def drop(n)
Lazy.new(self, state: ->{{remain: n}}) do |yielder, state, *values|
if state[:remain] > 0
state[:remain] -= 1
else
yielder.yield(*values)
end
end
end

Maybe the :state option could be an object instead of a lmabda, which would be dupped before each enumerations:

def drop(n)
Lazy.new(self, state: {remain: n}) do |yielder, state, *values|
if state[:remain] > 0
state[:remain] -= 1
else
yielder.yield(*values)
end
end
end

Both solutions seem complex to me, but would definitely put the correct emphasis on how to deal with state.

So, does the patch look acceptable as far as MRI is concerned?
For the public api, should there be a public Yielder#memo and a guarantee that there is exactly one yielder object per iteration? or instead an extra state yielded when required?

#7 - 01/21/2013 06:14 PM - shugo (Shugo Maeda)

marcandre (Marc-Andre Lafortune) wrote:

> The same idea will work for zip too; the arguments must be converted to enumerators only when yielder.memo is not set, i.e. every new yielder.
>
> The arguments are never really rewound, but the first yielder holding them will not be reused after enum.rewind; a new yielder is given.
> I haven't checked if that's the current behavior in other implementations, but if yielder is to be the state holder, that's the way it should work.

So, the following behavior is intended, right?

$ ruby1.9.3 -I ~/src/backports/lib -r backports -r backports/2.0.0/enumerable -e "a = (1..3).lazy.zip(('a'..'z').each); p a.to_a; p a.to_a"
[1, "a"], [2, "b"], [3, "c"
[1, "d"], [2, "e"], [3, "f"

> So, does the patch look acceptable as far as MRI is concerned?

If the above behavior is intended, the patch looks acceptable.

> For the public api, should there be a public Yielder#memo and a guarantee that there is exactly one yielder object per iteration? or instead an extra state yielded when required?

It might be too late to introduce a new API for 2.0.0, so how about to move it to next minor?

#8 - 01/22/2013 04:44 AM - marcandre (Marc-Andre Lafortune)

shugo (Shugo Maeda) wrote:

> So, the following behavior is intended, right?
>
> $ ruby1.9.3 -I ~/src/backports/lib -r backports -r backports/2.0.0/enumerable -e "a = (1..3).lazy.zip(('a'..'z').each); p a.to_a; p a.to_a"
> [1, "a"], [2, "b"], [3, "c"]
> [1, "d"], [2, "e"], [3, "f"]

That's a very good question.

It probably would be best to call to_enum instead of each. Calling next|rewind on an enumerator should really only affect other calls to next. With to_enum, we'll get the same result every time.

Similarly, we expect (1..3).zip(('a'..'z').each.tap(&:next)) to return [[1, 'a'], ..., and not [[1, 'b'], ... even though next was called on the given enumerator.

> If the above behavior is intended, the patch looks acceptable.

Thanks for reviewing it. I'll commit it, changing the call to each to to_enum (unless there are objections). I'll use the same technique to fix the other 3 lazy enumerators with state.

> > For the public api, should there be a public Yielder#memo and a guarantee that there is exactly one yielder object per iteration? or instead an extra state yielded when required?
>
> It might be too late to introduce a new API for 2.0.0, so how about to move it to next minor?

I understand. On the other hand, the API for Lazy.new was never decided upon and really should be finalized before 2.0.0!

If we opt for the Yielder#memo way (and agree on the name), maybe we can convince mame to accept such a trivial change. In that case, the biggest "change" is the explicit guarantee of a different yielder per iteration. It's already the case (also for JRuby and rubinius), but AFAIK it's never been official.

With the modified yielding way, it would be nice to include it in Lazy.new's api in this version, especially since it is still being finalized.

At the very least, a note in the documentation about handling state would be needed for 2.0.0.

#9 - 01/22/2013 10:24 AM - shugo (Shugo Maeda)

- Assignee set to matz (Yukihiro Matsumoto)

marcandre (Marc-Andre Lafortune) wrote:

> shugo (Shugo Maeda) wrote:
>
> > So, the following behavior is intended, right?
> >
> > $ ruby1.9.3 -I ~/src/backports/lib -r backports -r backports/2.0.0/enumerable -e "a = (1..3).lazy.zip(('a'..'z').each); p a.to_a; p a.to_a"
> > [1, "a"], [2, "b"], [3, "c"]
> > [1, "d"], [2, "e"], [3, "f"]
>
> That's a very good question.
>
> It probably would be best to call to_enum instead of each. Calling next|rewind on an enumerator should really only affect other calls to next. With to_enum, we'll get the same result every time.

Even if to_enum is called, IO instances never be rewound, but I guess IO instances need not be rewound.

> > It might be too late to introduce a new API for 2.0.0, so how about to move it to next minor?
>
> I understand. On the other hand, the API for Lazy.new was never decided upon and really should be finalized before 2.0.0!
>
> If we opt for the Yielder#memo way (and agree on the name), maybe we can convince mame to accept such a trivial change. In that case, the biggest "change" is the explicit guarantee of a different yielder per iteration. It's already the case (also for JRuby and rubinius), but AFAIK it's never been official.

`The explicit guarantee of a different yielder per iteration' sounds acceptable for me, but I'm not sure whether Yielder#memo is a good name.

> With the modified yielding way, it would be nice to include it in Lazy.new's api in this version, especially since it is still being finalized.
>
> At the very least, a note in the documentation about handling state would be needed for 2.0.0.

I believe Matz should decide it.

#10 - 01/22/2013 10:24 AM - shugo (Shugo Maeda)

- Status changed from Open to Assigned

#11 - 01/24/2013 03:22 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Assigned to Closed

- % Done changed from 0 to 100

This issue was solved with changeset r38920.
Marc-Andre, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

- enumerator.c: Fix state handling for Lazy#take
  [bug #7696]
- test/ruby/test_lazy_enumerator.rb: test for above

#12 - 01/24/2013 03:26 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Closed to Open

Bugs in MRI are fixed, but keeping open so Matz can decide how users should handle state.

#13 - 02/17/2013 03:50 PM - mame (Yusuke Endoh)

- Subject changed from Lazy enumerators with state can't be rewound to Lazy enumerators with state can't be rewound

- Status changed from Open to Assigned

- Target version changed from 2.0.0 to 2.6

#14 - 08/07/2013 12:21 PM - naruse (Yui NARUSE)

- Target version changed from 2.6 to 2.1.0

#15 - 08/31/2013 07:57 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Assigned to Closed

I created a new feature request #8840, so I'm closing this.