I would expect that:

\[
\text{array.flat_map}(...) == \text{array.lazy.flat_map}(...).force
\]

This is not always the case:

\[
\begin{align*}
[1].flat_map{|i| \{ i \mapsto i \}} & \mapsto [{1 \mapsto 1}], \text{ok} \\
[1].lazy.flat_map{|i| \{ i \mapsto i \}}.force & \mapsto [[1, 1]], \text{expected } [{1 \mapsto 1}]
\end{align*}
\]

Note that Matz confirmed that it is acceptable to return straight objects instead of arrays for flat_map [ruby-core:43365]

It looks like this was intended for nested lazy enumerators:

\[
[1].lazy.flat_map{|i| \{i.lazy \}}.force \mapsto [1]
\]

I don't think that's the correct result, and it is different from a straight flat_map:

\[
[1].flat_map{|i| \{i.lazy \}} \# \mapsto [\text{#<Enumerator::Lazy: [1]>}]
\]

This is caused by Lazy#flat_map calls each (while Enumerable#flat_map only looks for Arrays/object responding to to_ary).

---

**Associated revisions**

Revision 7dd52f89 - 01/14/2013 08:36 AM - shugo (Shugo Maeda)

- enumerator.c (lazy_flat_map_func): flat_map should call each only when the value of a block returns a forcable object. 
  [ruby-core:51401] [Bug #7690]

- enumerator.c (lazy_flat_map): add documentation.


git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@38812 b2dd03c8-39d4-4d8f-98f1-823fe69b080e

Revision 38812 - 01/14/2013 08:36 AM - shugo (Shugo Maeda)

- enumerator.c (lazy_flat_map_func): flat_map should call each only when the value of a block returns a forcable object. 
  [ruby-core:51401] [Bug #7690]

- enumerator.c (lazy_flat_map): add documentation.

• enumerator.c (lazy_flat_map_func): flat_map should call each only when the value of a block returns a forcable object. [ruby-core:51401] [Bug #7690]

• enumerator.c (lazy_flat_map): add documentation.

• test/ruby/test_lazy Enumerator.rb: related test.

Revision 38812 - 01/14/2013 08:36 AM - shugo (Shugo Maeda)

• enumerator.c (lazy_flat_map_func): flat_map should call each only when the value of a block returns a forcable object. [ruby-core:51401] [Bug #7690]

• enumerator.c (lazy_flat_map): add documentation.

• test/ruby/test_lazy_Enumerator.rb: related test.

Revision 38812 - 01/14/2013 08:36 AM - shugo (Shugo Maeda)

• enumerator.c (lazy_flat_map_func): flat_map should call each only when the value of a block returns a forcable object. [ruby-core:51401] [Bug #7690]

• enumerator.c (lazy_flat_map): add documentation.

• test/ruby/test_lazy_Enumerator.rb: related test.

Revision 38812 - 01/14/2013 08:36 AM - shugo (Shugo Maeda)

• enumerator.c (lazy_flat_map_func): flat_map should call each only when the value of a block returns a forcable object. [ruby-core:51401] [Bug #7690]

• enumerator.c (lazy_flat_map): add documentation.

• test/ruby/test_lazy_Enumerator.rb: related test.

Revision 38812 - 01/14/2013 08:36 AM - shugo (Shugo Maeda)

• enumerator.c (lazy_flat_map_func): flat_map should call each only when the value of a block returns a forcable object. [ruby-core:51401] [Bug #7690]

• enumerator.c (lazy_flat_map): add documentation.

• test/ruby/test_lazy_Enumerator.rb: related test.

History
marcandre (Marc-Andre Lafortune) wrote:

I would expect that

    array.flat_map(...) == array.lazy.flat_map(...).force

This is not always the case:

    [1].flat_map{|i| {i => i} } # => [{1 => 1}], ok
    [1].lazy.flat_map{|i| {i => i} }.force # => [[1, 1]], expected [{1 => 1}]

I agree that this looks weird.

Note that Matz confirmed that it is acceptable to return straight objects instead of arrays for flat_map [ruby-core:43365]

It looks like this was intended for nested lazy enumerators:

    [1].lazy.flat_map{|i| [i].lazy }.force # => [1]

I don’t think that’s the correct result, and it is different from a straight flat_map:

    [1].flat_map{|i| [i].lazy } # => [<Enumerator::Lazy: [1]>]

    [1].lazy.flat_map{|i| [i].lazy } should flatten nested lazy enumerators, because Enumerable::Lazy is a monad and flat_map is the monad's bind operator.

In the monad, [x].lazy is equivalent to Haskell's return and flat_map is equivalent to Haskell's >>= (bind).

# return :: a -> ma
[x].lazy

# (>>=) :: m a -> (a -> m b) -> m b
x.flat_map(&f)

Note that f's type is a -> m b, which means that f returns not an Array, but an Enumerable::Lazy.

In fact, [x].lazy and flat_map obey the monad laws.

# (return x) >>= f == f x
[x].lazy.flat_map(&f) == f.(x)

# m >>= return == m
m.flat_map { |i| [i].lazy } == m

# (m >>= f) >>= g == (\x -> f x >>= g)
m.flat_map(&f).flat_map(&g) == m.flat_map { |x| f.(x).flat_map(&g) }

That is, flat_map is an operator to compose computations which return an Enumerable::Lazy.

Do you have any use case of [1].flat_map{|i| {i => i} }?

shugo (Shugo Maeda) wrote:

    [1].lazy.flat_map{|i| [i].lazy } should flatten nested lazy enumerators, because Enumerable::Lazy is a monad and flat_map is the monad's bind operator.

Thanks for the explanation.

The idea is neat.

The problem is that:
1) This is documented nowhere
2) Most people think of flat_map as a shortcut to map.flatten(1), but flatten doesn't flatten Lazy enumerators (or Enumerables in general)
3) As Matz stated [ruby-core:26301], `flat_map` is "taken from flatMap from Scala or concatMap from Haskell". I'm not familiar with either, but I read that Scala's `flatMap` is not a monadic bind, right?

4) The argument about `flat_map` being a monadic bind applies only to monads (i.e. lazy enumerators). It should only flatten those, not arbitrary Enumerables

   Do you have any use case of `flat_map{|i| [i => i]}`?

   It's not just hashes, it could be a Range, or any Enumerable, or even any class that implements `#each`, even if it doesn't include Enumerable!

   So yes, I can think of many use cases, but instead of inventing them, here's one in Rails:

   In summary, I see the following 2 possibilities:
   1) Lazy#flat_map only flattens arrays, or
   2) Lazy#flat_map flattens Array and Enumerator::Lazy (using `is_a? Enumerator::Lazy instead of respond_to? :each) and the documentation reflects this

   If (1), maybe a new method can be introduced instead, say "bind"?
   If (2), shouldn't Enumerable#flat_map also flatten lazy enumerators?

---

#3 - 01/14/2013 11:28 AM - shugo (Shugo Maeda)

marcandre (Marc-Andre Lafortune) wrote:

   shugo (Shugo Maeda) wrote:

      [1].lazy.flat_map{|i| [i].lazy } should flatten nested lazy enumerators, because Enumerable::Lazy is a monad and flat_map is the monad's bind operator.

      Thanks for the explanation.

      The idea is neat.

      The problem is that:
      1) This is documented nowhere
      2) Most people think of flat_map as a shortcut to map.flatten(1), but flatten doesn't flatten Lazy enumerators (or Enumerables in general)

   Agreed, but 2) should be solved by documentation.

   3) As Matz stated [ruby-core:26301], `flat_map` is "taken from flatMap from Scala or concatMap from Haskell". I'm not familiar with either, but I read that Scala's `flatMap` is not a monadic bind, right?

   Where did you read that? I guess Scala's `flatMap` is also bind.

   Ruby's `Enumerable#flat_map` is also bind.
   Because `Enumerable#flat_map` returns an Array, `Enumerable#flat_map` takes a block which returns an Array.
   Because `Enumerable::Lazy#flat_map` returns an `Enumerator::Lazy`, `Enumerable::Lazy#flat_map` takes a block which returns an `Enumerator::Lazy`.
   They are consistent in that sense.

   4) The argument about `flat_map` being a monadic bind applies only to monads (i.e. lazy enumerators). It should only flatten those, not arbitrary Enumerables

   I feel difficulty about it because duck typing is preferred in Ruby.

   Do you have any use case of `flat_map{|i| [i => i]}`?

   It's not just hashes, it could be a Range, or any Enumerable, or even any class that implements `#each`, even if it doesn't include Enumerable!

   So yes, I can think of many use cases, but instead of inventing them, here's one in Rails:

   Technically, this code should be written as follows:

   ```ruby
   order_query.flat_map do |o|
     case o
       when Arel::Nodes::Ordering
   ```

   09/09/2021
In summary, I see the following 2 possibilities:
1) Lazy#flat_map only flattens arrays, or
2) Lazy#flat_map flattens Array and Enumerator::Lazy (using is_a? Enumerator::Lazy instead of respond_to? :each) and the documentation reflects this

I prefer 2), but am not sure whether `is_a? Enumerator::Lazy` is a neat solution. However, if I don't come up with a better solution, I will fix Lazy#flat_map using it.

#4 - 01/14/2013 01:15 PM - marcandre (Marc-Andre Lafortune)

shugo (Shugo Maeda) wrote:

3) As Matz stated [ruby-core:26301], flat_map is "taken from flatMap from Scala or concatMap from Haskell". I'm not familiar with either, but I read that Scala's flatMap is not a monadic bind, right?

Where did you read that? I guess Scala's flatMap is also bind.

"Scala's flatMap is indeed not a monadic bind" here [http://igstan.ro/posts/2012-08-23-scala-s-flatmap-is-not-haskell-s.html] but I only scanned this quickly and I don't know if that's correct.

Ruby's Enumerable#flat_map is also bind.
Because Enumerable#flat_map returns an Array, Enumerable#flat_map takes a block which returns an Array.
Because Enumerator::Lazy#flat_map returns an Enumerator::Lazy, Enumerator::Lazy#flat_map takes a block which returns an Enumerator::Lazy.
They are consistent in that sense.

Right. Except that both also accept straight objects.

4) The argument about flat_map being a monadic bind applies only to monads (i.e. lazy enumerators). It should only flatten those, not arbitrary Enumerables

I feel difficulty about it because duck typing is preferred in Ruby.

Right, but the core of Ruby relies more on conversions than pure duck typing.

In particular, Enumerable#flat_map uses to_ary. For the lazy flat_map, there is no "to_lazy" or similar...

Technically, this code should be written as follows:

... 

It indeed could be written with [[...]], but it does not have to, as confirmed by Matz [ruby-core:43365].

In summary, I see the following 2 possibilities:
1) Lazy#flat_map only flattens arrays, or
2) Lazy#flat_map flattens Array and Enumerator::Lazy (using is_a? Enumerator::Lazy instead of respond_to? :each) and the documentation reflects this

I prefer 2), but am not sure whether `is_a? Enumerator::Lazy` is a neat solution. However, if I don't come up with a better solution, I will fix Lazy#flat_map using it.
Sounds good.

#5 - 01/14/2013 01:26 PM - marcandre (Marc-Andre Lafortune)
PS: The only duck typing I can think of is respond_to?(:each) & respond_to?(:force); not sure if that's much better though.

#6 - 01/14/2013 04:50 PM - shugo (Shugo Maeda)
marcandre (Marc-Andre Lafortune) wrote:

shugo (Shugo Maeda) wrote:

3) As Matz stated [ruby-core:26301], flat_map is “taken from flatMap from Scala or concatMap from Haskell”. I'm not familiar with either, but I read that Scala’s flatMap is not a monadic bind, right?

Where did you read that? I guess Scala's flatMap is also bind.

"Scala’s flatMap is indeed not a monadic bind" here http://igstan.ro/posts/2012-08-23-scala-s-flatmap-is-not-haskell-s.html but I only scanned this quickly and I'm don't know if that's correct.

Thanks for the information.
I guess the comment said "Scala's flatMap is indeed not a monadic bind" because Scala's flatMap is extended to accept functions which returns another type of container.

scala> List(1, 2, 3, 4) flatMap {x => Some(x)}
res0: List[Int] = List(1, 2, 3, 4)

Here, the function {x => Some(x)} returns Some(x), which is not a List, but flatMap unwrap values from them. In this case, flatMap is not a bind operator.

However, it can be used as a bind operator if a given function returns a List.

scala> List("foo bar", "baz") flatMap {x => x.split(" ")}
res6: List[Ljava.lang.String;] = List(foo, bar, baz)

That's why I said Lazy#flat_map should flatten lazy enumerators. It's not a pure bind operator, but should be able to be used as a bind operator.

4) The argument about flat_map being a monadic bind applies only to monads (i.e. lazy enumerators). It should only flatten those, not arbitrary Enumerables

I feel difficulty about it because duck typing is preferred in Ruby.

Right, but the core of Ruby relies more on conversions than pure duck typing.

In particular, Enumerable#flat_map uses to_ary. For the lazy flat_map, there is no "to_lazy" or similar...

Yes, that's the problem I was thinking of.

I was thinking of having a predicate like lazy Enumerator?, but your idea of checking each and force sounds better, because it's too late to introduce a new method for Ruby 2.0.0.

#7 - 01/14/2013 05:36 PM - shugo (Shugo Maeda)
- Status changed from Assigned to Closed
- % Done changed from 0 to 100

This issue was solved with changeset r38812.
Marc-Andre, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

• enumerator.c (lazy_flat_map_func): flat_map should call each only when the value of a block returns a forcable object. [ruby-core:51401] [Bug #7690]
- enumerator.c (lazy_flat_map): add documentation.
- test/ruby/test_lazy_enumerator.rb: related test.