Ruby - Feature #7241

Enumerable#to_h proposal

10/30/2012 07:23 AM - nathan.f77 (Nathan Broadbent)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

I often use the inject method to build a hash, but I always find it annoying when I need to return the hash at the end of the block.
This means that I often write code like:

```
[1,2,3,4,5].inject({}) {|hash, el| hash[el] = el * 2; hash }
```

I'm proposing an Enumerable#to_h method that would let me write:

```
[1,2,3,4,5].to_h {|h, el| h[el] = el * 2 }
```

I saw the proposal at http://bugs.ruby-lang.org/issues/666, but I would not be in favor of his implementation.
I believe the implementation should be similar to inject, so that the hash object and next element are passed to the block. The main difference to the inject method is that we would be modifying the hash in place, instead of relying on the block's return value.

As well as providing support for the case above, I have also considered other cases where the to_h method would be useful.
I thought it would be useful if symmetry were provided for the Hash#to_a method, such that:

```
hash.to_a.to_h == hash  # => true
```

(See example 2)

I've allowed developers to provide a symbol instead of a block, so that each element in the collection will be passed to that named method. (See example 3)

Finally, hashes can be given a default value, or a Proc that returns the default value. (See examples 4 & 5)

Heres an example implementation that I would be happy to rewrite in C if necessary:

```
module Enumerable
  def to_h(default_or_sym = nil)
    if block_given?
      hash = if Proc === default_or_sym
        Hash.new(&default_or_sym)
      else
        Hash.new(default_or_sym)
      end
      self.each do |el|
        yield hash, el
      end
    elsif !default_or_sym.nil?
      hash = {}
      self.each do |el|
        hash[el] = el.send(default_or_sym)
      end
    else
      return Hash[*self.to_a.flatten(1)]
    end
    hash
  end
end
```

Examples

1) Build a hash from array elements

```
[1,2,3,4,5].to_h {|h, el| h[el] = el * 2 }
```

=> {1=>2, 2=>4, 3=>6, 4=>8, 5=>10}

2) Provides symmetry for Hash.to_a (i.e. you can call hash.to_a.to_h)

```
[[1, 2], [3, 4], [5, 6]].to_h
```

=> {1=>2, 3=>4, 5=>6}

3) Build a hash by calling a method on each array element

```
["String", "Another String"].to_h(:size)
```

=> {"String"=>6, "Another String"=>14}

4) Hash with default value

```
[4,5,6,5].to_h(0) {|h, el| h[el] += el }
```

=> {4=>4, 5=>10, 6=>6}

5) Hash with default value returned from Proc

```
default_proc = -> hash, key { hash[key] = "go fish: #{key}" }
[4,5,6].to_h(default_proc) {|h, el| h[el].upcase! }
```

=> {4=>"GO FISH: 4", 5=>"GO FISH: 5", 6=>"GO FISH: 6"}

Thanks for your time, and please let me know your thoughts!

Best,
Nathan Broadbent

Related issues:

Related to Ruby - Feature #5008: Equal rights for Hash (like Array, String, I...	**Rejected**	**07/10/2011**
Related to Ruby - Feature #4151: Enumerable#categorize	**Rejected**	
Is duplicate of Ruby - Feature #666: Enumerable::to_hash	**Rejected**	**10/20/2008**
Has duplicate Ruby - Feature #7292: Enumerable#to_h	**Closed**	**11/07/2012**

History

#1 - 10/30/2012 08:23 AM - Anonymous

On Tue, Oct 30, 2012 at 07:23:29AM +0900, nathan.f77 (Nathan Broadbent) wrote:

> Issue #7241 has been reported by nathan.f77 (Nathan Broadbent).
>
> ---
>
> Feature #7241: Enumerable#to_h proposal
> https://bugs.ruby-lang.org/issues/7241
>
> Author: nathan.f77 (Nathan Broadbent)
> Status: Open
> Priority: Normal
> Assignee:
> Category: core
> Target version:
>
> I often use the inject method to build a hash, but I always find it annoying when I need to return the hash at the end of the block.
> This means that I often write code like:
>
> ```
> [1,2,3,4,5].inject({}) {|hash, el| hash[el] = el * 2; hash }
> ```

1.9.3p194 :001 > [1,2,3,4].each_with_object({}) { |x,o| o[x] = x ** 2 }
=> {1=>1, 2=>4, 3=>9, 4=>16}
1.9.3p194 :002 >

--
Aaron Patterson
http://tenderlovemaking.com/

#2 - 10/30/2012 08:27 AM - v_krishna (Vijay Ramesh)

Or

1.9.3-p0 :001 > Hash[[1,2,3,4,5].map{|el| [el, el*2]}]
=> {1=>2, 2=>4, 3=>6, 4=>8, 5=>10}

#3 - 10/30/2012 08:37 AM - matz (Yukihiro Matsumoto)

Your idea of to_h is interesting, but it adds too much behavior in one method.
Besides that, since to_s, to_a, to_i etc. are used for implicit conversion, to_h is not a proper name for the method.

Nice try, we will wait for next one.

Matz.

#4 - 10/30/2012 08:37 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

#5 - 10/30/2012 08:53 AM - nathan.f77 (Nathan Broadbent)

Thanks! Sorry, I didn't know about each_with_object.

Do you think it would still be worth shortening
each_with_object(Hash.new([])) { ... } to to_h([]) { ... }, and are any
of the other cases worth supporting?

Best,
Nathan

On Tue, Oct 30, 2012 at 12:18 PM, Aaron Patterson
tenderlove@ruby-lang.orgwrote:

> On Tue, Oct 30, 2012 at 07:23:29AM +0900, nathan.f77 (Nathan Broadbent)
> wrote:
>
> > Issue #7241 has been reported by nathan.f77 (Nathan Broadbent).
> >
> > ---
> >
> > Feature #7241: Enumerable#to_h proposal
> > https://bugs.ruby-lang.org/issues/7241
> >
> > Author: nathan.f77 (Nathan Broadbent)
> > Status: Open
> > Priority: Normal
> > Assignee:
> > Category: core
> > Target version:
> >
> > I often use the inject method to build a hash, but I always find it
> > annoying when I need to return the hash at the end of the block.
> > This means that I often write code like:
> >
> > ```
> > [1,2,3,4,5].inject({}) {|hash, el| hash[el] = el * 2; hash }
> > ```
>
> 1.9.3p194 :001 > [1,2,3,4].each_with_object({}) { |x,o| o[x] = x ** 2 }
> => {1=>1, 2=>4, 3=>9, 4=>16}
> 1.9.3p194 :002 >
>
> --
> Aaron Patterson
> http://tenderlovemaking.com/

#6 - 10/30/2012 08:53 AM - nathan.f77 (Nathan Broadbent)

OK, no problem! Thanks for your response!

A bit unrelated, but is it strange that each_with_object and inject have a different order for the block params?

```
[1,2,3].inject({}) {|obj, el| obj[el] = el * 2; obj }        #=> {1=>2,
```

2=>4, 3=>6}

```
[1,2,3].each_with_object({}) {|obj, el| obj[el] = el * 2 }  #=>
```

NoMethodError: undefined method `*' for {}:Hash

```
[1,2,3].each_with_object({}) {|el, obj| obj[el] = el * 2 }  #=> {1=>2,
```

2=>4, 3=>6}

On Tue, Oct 30, 2012 at 12:37 PM, matz (Yukihiro Matsumoto) <matz@ruby-lang.org> wrote:

> Issue #7241 has been updated by matz (Yukihiro Matsumoto).
>
> Status changed from Open to Rejected
>
> ---
>
> Feature #7241: Enumerable#to_h proposal
> https://bugs.ruby-lang.org/issues/7241#change-31937
>
> Author: nathan.f77 (Nathan Broadbent)
> Status: Rejected
> Priority: Normal
> Assignee:
> Category: core
> Target version:
>
> I often use the inject method to build a hash, but I always find it annoying when I need to return the hash at the end of the block.
> This means that I often write code like:
>
> ```
> [1,2,3,4,5].inject({}) {|hash, el| hash[el] = el * 2; hash }
> ```
>
> I'm proposing an Enumerable#to_h method that would let me write:
>
> ```
> [1,2,3,4,5].to_h {|h, el| h[el] = el * 2 }
> ```
>
> I saw the proposal at http://bugs.ruby-lang.org/issues/666, but I would not be in favor of his implementation.
> I believe the implementation should be similar to inject, so that the hash object and next element are passed to the block. The main difference to the inject method is that we would be modifying the hash in place, instead of relying on the block's return value.
>
> As well as providing support for the case above, I have also considered other cases where the to_h method would be useful.
> I thought it would be useful if symmetry were provided for the Hash#to_a method, such that:
>
> ```
> hash.to_a.to_h == hash  # => true
> ```
>
> (See example 2)
>
> I've allowed developers to provide a symbol instead of a block, so that each element in the collection will be passed to that named method. (See example 3)
>
> Finally, hashes can be given a default value, or a Proc that returns the default value. (See examples 4 & 5)
>
> Heres an example implementation that I would be happy to rewrite in C if necessary:
>
> ```
> module Enumerable
>   def to_h(default_or_sym = nil)
> ```

```
    if block_given?
      hash = if Proc === default_or_sym
        Hash.new(&default_or_sym)
      else
        Hash.new(default_or_sym)
      end
      self.each do |el|
        yield hash, el
      end
    elsif !default_or_sym.nil?
      hash = {}
      self.each do |el|
        hash[el] = el.send(default_or_sym)
      end
    else
      return Hash[*self.to_a.flatten(1)]
    end
    hash
  end
end
```

Examples

1) Build a hash from array elements

```
[1,2,3,4,5].to_h {|h, el| h[el] = el * 2 }
```

=> {1=>2, 2=>4, 3=>6, 4=>8, 5=>10}

2) Provides symmetry for Hash.to_a (i.e. you can call hash.to_a.to_h)

```
[[1, 2], [3, 4], [5, 6]].to_h
```

=> {1=>2, 3=>4, 5=>6}

3) Build a hash by calling a method on each array element

```
["String", "Another String"].to_h(:size)
```

=> {"String"=>6, "Another String"=>14}

4) Hash with default value

```
[4,5,6,5].to_h(0) {|h, el| h[el] += el }
```

=> {4=>4, 5=>10, 6=>6}

5) Hash with default value returned from Proc

```
default_proc = -> hash, key { hash[key] = "go fish: #{key}" }
[4,5,6].to_h(default_proc) {|h, el| h[el].upcase! }
```

=> {4=>"GO FISH: 4", 5=>"GO FISH: 5", 6=>"GO FISH: 6"}

Thanks for your time, and please let me know your thoughts!

Best,
Nathan Broadbent

--
http://bugs.ruby-lang.org/

#7 - 10/30/2012 07:58 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Maybe .hash_map? each_with_object is a too long name for a very common needed method. Many have asked for a method like it (including me) because they couldn't find "each_with_object" and they ended up learning here after asking for such a method.

Maybe "hash_map" could be a better name for this.

matz (Yukihiro Matsumoto) wrote:

> Your idea of to_h is interesting, but it adds too much behavior in one method.
> Besides that, since to_s, to_a, to_i etc. are used for implicit conversion, to_h is not a proper name for the method.
>
> Nice try, we will wait for next one.
>
> Matz.

#8 - 10/31/2012 03:29 AM - Anonymous

On Tue, Oct 30, 2012 at 07:58:33PM +0900, rosenfeld (Rodrigo Rosenfeld Rosas) wrote:

> Issue #7241 has been updated by rosenfeld (Rodrigo Rosenfeld Rosas).
>
> Maybe .hash_map? each_with_object is a too long name for a very common needed method. Many have asked for a method like it (including me) because they couldn't find "each_with_object" and they ended up learning here after asking for such a method.
>
> Maybe "hash_map" could be a better name for this.

each_with_object isn't specific to hashes, and isn't doing list
translation like map does.

IOW, it sounds perfect for ActiveSupport. ;-)

--
Aaron Patterson
http://tenderlovemaking.com/

#9 - 10/31/2012 06:23 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 30-10-2012 16:23, Aaron Patterson escreveu:

> On Tue, Oct 30, 2012 at 07:58:33PM +0900, rosenfeld (Rodrigo Rosenfeld Rosas) wrote:
>
> > Issue #7241 has been updated by rosenfeld (Rodrigo Rosenfeld Rosas).
> >
> > Maybe .hash_map? each_with_object is a too long name for a very common needed method. Many have asked for a method like it (including me) because they couldn't find "each_with_object" and they ended up learning here after asking for such a method.
> >
> > Maybe "hash_map" could be a better name for this.
> each_with_object isn't specific to hashes, and isn't doing list
> translation like map does.
>
> IOW, it sounds perfect for ActiveSupport. ;-)

I often have this requirement and I guess others have it as well. There
are two problems with each_with_object in my opinion:

1 - you can't find it easily in the docs when you're looking for some
way to "inject" a Hash without worrying about the result of the block;
hash_map would be easier to find in the docs for newcomers (to
each_with_object I mean, like I was less then an year ago if I remember
correctly);
2 - it is a too long name. See examples below:

hash =
a_long_array_name_as_I_usually_use_for_my_variables.each_with_object({}){|(name,
url), h| h[name] = url }
h = {}; a_long_array_name_as_I_usually_use_for_my_variables.each{|(name,
url)| h[name] = url }; hash = h

Often in my methods I don't really need that extra (; hash = h) so it is
usually much shorter when I don't use each_with_object.

With proposed method:

hash = a_long_array_name_as_I_usually_use_for_my_variables.hash_map{|h,
(name, url)| h[name] = url }

Notice that I changed the order of the arguments for the block. It makes
more sense to me this way, just like inject.

I know this is subjective but I find the last example better to read ;)

Cheers,
Rodrigo.

#10 - 11/01/2012 11:53 AM - trans (Thomas Sawyer)

Almost no one uses #each_with_object as it is. #each_with_hash is hardly
better. We need a short method name. Moreover I don't think this method's
behavior is really the best approach to the real use case.

On Wed, Oct 31, 2012 at 7:07 PM, Nathan Broadbent nathan.f77@gmail.comwrote:

> Hi everyone,
>
> Please see the pull request that I've opened on Rails ActiveSupport, to
> add an each_with_hash method: https://github.com/rails/rails/pull/8088
>
> @matz (Yukihiro Matsumoto): Do you think this each_with_hash implementation could be added to
> Ruby, or is it better as a Rails ActiveSupport extension?
>
> Best,
> Nathan

--
Sorry, says the barman, we don't serve neutrinos. A neutrino walks into a
bar.

Trans transfire@gmail.com
7r4n5.com http://7r4n5.com

#11 - 11/01/2012 12:23 PM - nathan.f77 (Nathan Broadbent)

> Almost no one uses #each_with_object as it is. #each_with_hash is hardly
> better. We need a short method name. Moreover I don't think this method's
> behavior is really the best approach to the real use case.

It's true that each_with_object doesn't seem to be used too much, but when
it is used, the object is usually a hash (for 90% of the cases in Rails, at
least.)

I think that each_with_hash should be provided for when you want to map an
enumerable onto a Hash, but I think that there should also be a 'to_h'
method on Array for when you just want to *convert* an Array into a hash.

I think 'to_h' would be most useful if it supported the behaviour of both
Hash[arr], and 'Hash[*arr]'. I'm on my phone at the moment, but
here's how I could see that working:

```
def to_h
if self.all? {|el| el.respond_to? :each && el.size == 2 }
Hash[self]
else
Hash[*self]
end
end
```

We could just let Hash[] handle any invalid input.

#12 - 11/01/2012 12:23 PM - Anonymous

Hi,

In message "Re: [ruby-core:48690] Re: [ruby-trunk - Feature #7241] Enumerable#to_h proposal"
on Thu, 1 Nov 2012 08:07:11 +0900, Nathan Broadbent nathan.f77@gmail.com writes:

|@matz: Do you think this each_with_hash implementation could be added to
|Ruby, or is it better as a Rails ActiveSupport extension?

I think it should go in to ActiveSupport first.

```
matz.
```

#13 - 11/01/2012 12:29 PM - nathan.f77 (Nathan Broadbent)

> I think it should go in to ActiveSupport first.
>
> ```
> matz.
> ```

Thanks for your reply! The pull request has just been rejected on ActiveSupport, so I guess that's the end of this discussion :)

Thank you for Ruby, by the way, it's a beautiful language!

Best,
Nathan

#14 - 11/01/2012 05:10 PM - alexeymuranov (Alexey Muranov)

Just in case, here is some relevant discussion on StackOverflow with benchmarks:

http://stackoverflow.com/questions/3230863/ruby-rails-inject-on-hashes-good-style

#15 - 11/01/2012 05:49 PM - trans (Thomas Sawyer)

=begin
I wouldn't say it is over. See #4151.

I still like:

module Enumerable
def each_with(x={})
each{ |e| yield(x,e) }
x
end
end

Is #each_with a better name?
=end

#16 - 11/01/2012 07:23 PM - nathan.f77 (Nathan Broadbent)

> I wouldn't say it is over. See #4151. ...

Is #each_with a better name?

Has anyone suggested map_to? I think map_to has a clearer intention
than each_with, because you're mapping the collection onto something, and
then returning it.
I don't really like the each part of each_with_object, because
array.each just returns the array. Since we usually use each to
iterate, and map to build an array, I think map_to(<object>) might make
sense.

How does this look:

[1, 2, 3].map_to({}) { |e, hash| hash[e] = e ** 2 }

I'd also propose a map_to_hash method. It's longer than map_to({}), but
I think it's nicer to read:

[1, 2, 3].map_to_hash { |e, hash| hash[e] = e ** 2 }

map_to_hash(0) would also be nicer than map_to(Hash.new(0)).

What do you think?

#17 - 11/11/2012 12:47 AM - jballanc (Joshua Ballanco)

=begin
Clojure has a function (({into})) that might fit the bill. An equivalent Ruby implementation might look something like the following:

```
class Hash
  alias :<< :merge!
end

module Enumerable
  def into(coll)
    coll = coll.dup
    each do |elem|
      coll << yield(elem)
    end
    coll
  end
end

chars = (97..107).into({}) { |i| { i => i.chr } }
p chars

require 'prime'
prime_chars = chars.into([]) { |k, v| k.prime? ? v : nil }
p prime_chars.compact

char_string = chars.into("") { |k, v| "#{k}=>#{v}, " }
p char_string
```

=end

#18 - 11/11/2012 05:59 PM - duerst (Martin Dürst)

On 2012/11/11 0:47, jballanc (Joshua Ballanco) wrote:

> Issue #7241 has been updated by jballanc (Joshua Ballanco).
>
> =begin
> Clojure has a function (({into})) that might fit the bill.

This indeed looks very promising.

> An equivalent Ruby implementation might look something like the following:
>
> ```
> class Hash
>   alias :<<  :merge!
> end
> ```

I might be wrong, but my guess is that constructing lots of
one-key/value hashes isn't very efficient. Two-element arrays should be
quite a bit more efficient. So we could define this as follows (in the
end in C, but here just in Ruby):

class Hash
def << (other)
case other.class
when Array
store(other[0], other[1])
when Hash
merge! other
end
self
end
end

(some additional tweaks may be needed for Array-like and Hash-like objects).

```
module Enumerable
  def into(coll)
    coll = coll.dup
    each do |elem|
      coll<<  yield(elem)
    end
```

```
    coll
  end
end

chars = (97..107).into({}) { |i| { i =>  i.chr } }
p chars

require 'prime'
prime_chars = chars.into([]) { |k, v| k.prime? ? v : nil }
p prime_chars.compact
```

It would be great to have a version that avoided "compact". Or maybe only that version would be okay? This would use "concat" instead of merge! (with Hash#concat an alias for Hash#merge!). Because neither Hashes nor Strings can be nested, there would actually not be any difference for those, but for Array, the preceeding code could be simplified to:

```
require 'prime'
prime_chars = chars.into___([]) { |k, v| k.prime? ? [v] : [] }
```

I often want a "collect" method where I'm not forced to collect exactly one item per item of the original collection. If collect weren't an alias to map, I think it would even make a lot of sense to use the word "collect" for this (map: one-to-one, collect: one-to-many).

Regards, Martin.

```
char_string = chars.into("") { |k, v| "#{k}=>#{v}, " }
p char_string
```

=end

Feature #7241: Enumerable#to_h proposal
https://bugs.ruby-lang.org/issues/7241#change-32755

Author: nathan.f77 (Nathan Broadbent)
Status: Rejected
Priority: Normal
Assignee:
Category: core
Target version:

I often use the inject method to build a hash, but I always find it annoying when I need to return the hash at the end of the block.
This means that I often write code like:

```
[1,2,3,4,5].inject({}) {|hash, el| hash[el] = el * 2; hash }
```

I'm proposing an Enumerable#to_h method that would let me write:

```
[1,2,3,4,5].to_h {|h, el| h[el] = el * 2 }
```

I saw the proposal at http://bugs.ruby-lang.org/issues/666, but I would not be in favor of his implementation.
I believe the implementation should be similar to inject, so that the hash object and next element are passed to the block. The main difference to the inject method is that we would be modifying the hash in place, instead of relying on the block's return value.

As well as providing support for the case above, I have also considered other cases where the to_h method would be useful.
I thought it would be useful if symmetry were provided for the Hash#to_a method, such that:

```
hash.to_a.to_h == hash  # =>  true
```

(See example 2)

I've allowed developers to provide a symbol instead of a block, so that each element in the collection will be passed to that named method. (See example 3)

Finally, hashes can be given a default value, or a Proc that returns the default value. (See examples 4& 5)

Heres an example implementation that I would be happy to rewrite in C if necessary:

```
module Enumerable
  def to_h(default_or_sym = nil)
    if block_given?
```

```
      hash = if Proc === default_or_sym
        Hash.new(&default_or_sym)
      else
        Hash.new(default_or_sym)
      end
      self.each do |el|
        yield hash, el
      end
    elsif !default_or_sym.nil?
      hash = {}
      self.each do |el|
        hash[el] = el.send(default_or_sym)
      end
    else
      return Hash[*self.to_a.flatten(1)]
    end
    hash
  end
end
```

Examples

1) Build a hash from array elements

```
[1,2,3,4,5].to_h {|h, el| h[el] = el * 2 }
```

=> {1=>2, 2=>4, 3=>6, 4=>8, 5=>10}

2) Provides symmetry for Hash.to_a (i.e. you can call hash.to_a.to_h)

```
[[1, 2], [3, 4], [5, 6]].to_h
```

=> {1=>2, 3=>4, 5=>6}

3) Build a hash by calling a method on each array element

```
["String", "Another String"].to_h(:size)
```

=> {"String"=>6, "Another String"=>14}

4) Hash with default value

```
[4,5,6,5].to_h(0) {|h, el| h[el] += el }
```

=> {4=>4, 5=>10, 6=>6}

5) Hash with default value returned from Proc

```
default_proc = ->  hash, key { hash[key] = "go fish: #{key}" }
[4,5,6].to_h(default_proc) {|h, el| h[el].upcase! }
```

=> {4=>"GO FISH: 4", 5=>"GO FISH: 5", 6=>"GO FISH: 6"}

Thanks for your time, and please let me know your thoughts!

Best,
Nathan Broadbent

#19 - 11/11/2012 06:53 PM - nathan.f77 (Nathan Broadbent)

Clojure has a function (({into})) that might fit the bill.

This indeed looks very promising.

I like the sound of 'into', but am not sure about appending results with the '<<' operator. If Hash had '<<' and '+' aliases for 'update' and 'merge' (respectively), we might as well give 'map' an optional argument, and call:

```
[1,2,3].map({}) {|i| { i => i ** 2 } }
```

And if Hash#update accepted a two-element array, we could do:

```
[1,2,3].map({}) {|i| [i, i ** 2] }
```

So I like the 'into' name, but I think it would be more useful as an alias for 'each_with_object', instead of just 'map' with an argument for the base object.

> I often want a "collect" method where I'm not forced to collect exactly one item per item of the original collection. If collect weren't an alias to map, I think it would even make a lot of sense to use the word "collect" for this (map: one-to-one, collect: one-to-many).

Ruby has a 'flat_map' method (aliased as 'collect_concat') that flattens the first level of a returned array, so you can append multiple results, and don't need to use compact. See http://ruby-doc.org/core-1.9.3/Enumerable.html#method-i-flat_map

```
[1,nil,2].flat_map {|i| i ? [i] : [] }    #=> [1, 2]
```

Best,
Nathan

#20 - 11/12/2012 10:07 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I like "into". But I'd vote it to be an alias to "each_of_object" as I even prefer "into" instead of "each_with" or "map_with". I'd also vote for the order of the closure arguments to be changed.

I read "doubles = numbers.into({}){|h, n| h[n] = 2 * n }" as "assign to double the numbers into a hash indexed by each number having the double as value".