Ruby master - Feature #6687

Enumerable#with

07/02/2012 10:38 PM - merborne (kyo endo)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

=begin
Let me propose Enumerable#with for an alias of Enumerable#each_with_object or replace of it.

Enumerable#each_with_objectのエイリアス、またはその置き換えとして、Enumerable#withを提案します。

##Reason

##理由

When you create a hash using Enumerable#inject, you should ensure that the block return the hash.

Enumerable#injectを使ってハッシュを生成するときには、ブロックの返り値としてハッシュが返ることを保証する必要があります。

```
words.inject(Hash.new(0)) { |h, word| h[word] += 1; h } # => {"You"=>3, "say"=>10, "Yes"=>1, "I"=>7, "No"=>1, "Stop"=>1, "and"=>2, "Go"=>1, "go"=>2, "Oh"=>1, "no"=>1, "Goodbye"=>2, "Hello"=>2, "hello"=>5, "don"=>2, "t"=>2, "know"=>2, "why"=>2, "you"=>2, "goodbye"=>1}
```

Many rubyists, however, hate this, and there are many discussions for it.

しかし、これを嫌うRubyistは多く、ネット上でその改善についての議論をしばしば見掛けます。

```
Feature #5662: inject-accumulate, or Haskell's mapAccum* - ruby-trunk - Ruby Issue Tracking System http://bugs.ruby-lang.org/issues/5662
```

```
Ruby inject with intial being a hash - Stack Overflow http://stackoverflow.com/questions/9434162/ruby-inject-with-intial-being-a-hash
```

Enumerable#each_with_object is often presented for one of the best solutions for it.

そしてその有力な解決策として、Enumerable#each_with_objectが提示されてきました。

```
words.each_with_object(Hash.new(0)) { |word, h| h[word] += 1 } # => {"You"=>3, "say"=>10, "Yes"=>1, "I"=>7, "No"=>1, "Stop"=>1, "and"=>2, "Go"=>1, "go"=>2, "Oh"=>1, "no"=>1, "Goodbye"=>2, "Hello"=>2, "hello"=>5, "don"=>2, "t"=>2, "know"=>2, "why"=>2, "you"=>2, "goodbye"=>1}
```

However, each_with_object is still unfamiliar and then not used frequently. The biggest reason, I think, is its lengthy name.

しかし、その有用性にも拘らず、依然としてeach_with_objectの知名度および利用頻度は低いと思われます。そして、その原因は、その名前の長さにあると考えます。

each_with_object is the 39th longest-name method among 754 at Ruby 1.9.3, based on following calculation;

以下の演算により、Ruby1.9.3の環境下でeach_with_objectは、754件中39番目に長い名前のメソッドであることが分かりました。

```
methods = Module.constants.flat_map do |c|
  next [] if c == :Gem
  k = Module.const_get(c)
  k.methods(false) + k.instance_methods(false) rescue []
end.uniq.reject { |m| "#{m}".start_with? '_deprecated' }.sort_by { |m| -m.size }
```

```
methods.size # => 754
methods.index(:each_with_object) # => 39
```

```
puts methods.take(100).group_by(&:size).to_a
```

The output is here.

□□□□□

```
26
protected_instance_methods
instance_variable_defined?
25
protected_method_defined?
24
private_instance_methods
23
class_variable_defined?
public_instance_methods
define_singleton_method
private_method_defined?
22
singleton_method_added
public_instance_method
public_method_defined?
21
instance_variable_get
instance_variable_set
remove_class_variable
20
private_class_method
repeated_combination
repeated_permutation
compare_by_identity?
19
respond_to_missing?
abort_on_exception=
public_class_method
compare_by_identity
18
undefine_finalizer
instance_variables
abort_on_exception
class_variable_get
class_variable_set
relative_path_from
17
internal_encoding
external_encoding
default_internal=
default_external=
protected_methods
singleton_methods
ascii_compatible?
16
global_variables
executable_real?
initialize_clone
each_with_object   # <= Here!
require_relative
private_constant
default_external
included_modules
instance_methods
define_finalizer
default_internal
15
private_methods
fixed_encoding?
class_variables
instance_method
each_with_index
```

```
public_constant
garbage_collect
source_location
valid_encoding?
singleton_class
world_writable?
local_variables
world_readable?
method_defined?
14
readable_real?
locale_charmap
const_defined?
collect_concat
initialize_dup
add_trace_func
close_on_exec=
close_on_exec?
named_captures
set_trace_func
write_nonblock
writable_real?
each_codepoint
force_encoding
public_methods
13
const_missing
each_filename
default_proc=
set_backtrace
public_method
read_nonblock
instance_exec
absolute_path
count_objects
instance_eval
12
marshal_load
reverse_each
exclude_end?
instance_of?
make_symlink
set_encoding
block_given?
default_proc
slice_before
marshal_dump
11
rationalize
realdirpath
each_object
expand_path
with_object
```

This result shows that methods which has 15+ name length is mostly for reflection or for special purpose. each_with_object is a general purpose method, the name should be shorter.

このリストから分かることは、長さ15を超えるメソッドはその大半がリフレクション用か特殊目的用のものであるという事実です。each_with_objectはより汎用的なメソッドですから、その名前はもっと短くあるべきと考えます。現状の長さは、そのメソッドを無きものにしています。

I propose Enumerable#with for it. a word object in each_with_object is obvious and not necessary to spcify the purpose, because all data in Ruby is object. Also, a word each in each_with_object is not essential, then omittable in view of the fact that it is called to Enumerable object. I think that a word with still works for describing the same of each_with_object.

そこでEnumerable#withを提案します。まず、Rubyで扱われるデータはすべてオブジェクトですから、each_with_objectにおけるobjectは自明であり不要と考えます。次に、Enumerableオブジェクトに対するメソッド呼び出しという点から見て、eachも必須のものとは言えず、削除可能と考えます。そして残ったwithで十分にその目的、つまりEnumerableな要素を任意のオブジェクトと共に操作する、を意図できていると考えます。

Lastly, following is examples with Enumerable#with

□□□□Enumerable#with□□□□□□□□□□□□

```
Enumerable.send(:alias_method, :with, :each_with_object)

words.with(Hash.new(0)) { | word, h| h[word] += 1 } # => {"You"=>3, "say"=>10, "Yes"=>1, "I"=>7, "No"=>1, "Stop"=>1, "and"=>2, "Go"=>1, "go"=>2, "Oh"=>1, "no"=>1, "Goodbye"=>2, "Hello"=>2, "hello"=>5, "don"=>2, "t"=>2, "know"=>2, "why"=>2, "you"=>2, "goodbye"=>1}

[*1..10].with(5).map(&:*) # => [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

['ruby', 'python', 'haskell'].with('ist').map(&:+) # => ["rubyist", "pythonist", "haskellist"]
```

Thank you for your consideration.

□□□□□□□□□□□□□□□□□□

=end

Related issues:

Related to Ruby master - Feature #7384: Rename #each_with_object to #each_with	**Open**
Related to Ruby master - Feature #7340: 'each_with' or 'into' alias for 'each...	**Open**

History

#1 - 07/03/2012 01:17 AM - Eregon (Benoit Daloze)

That's a beautiful demonstration.
I indeed find #each_with_object too long, and that makes me use a local variable instead most of the time.

The other reason I hesitate to use it is for cases with multiple arguments in the enumeration:

```
{a: 1, b: 2}.with({}) { |(k, v), h| h[k] = do_sth(v) }

{a: 1, b: 2}.each_with_object({}) { |(k, v), h| h[k] = do_sth(v) }

h = {}
{a: 1, b: 2}.each { |k, v| h[k] = do_sth(v) }
h
```

But it sounds already more reasonable with #with.

I also agree "object" is redundant, I'm less sure about "each", but anyway there would logically be #with in Enumerator if there was #each_with in Enumerable, so I think it's better to have only #with (and #each_with is still too long to me).

#2 - 07/04/2012 02:50 PM - knu (Akinori MUSHA)

> Also, a word each in each_with_object is not essential, then omittable in view of the fact that it is called to Enumerable object.

I doubt it. Enumerable is often just one probably minor aspect of an including class, and even when used with an array or hash it is not obvious from the name with or with_object that it would iterate over the contents. I'd say it's too bold.

However, just as Eregon says, I wouldn't deny it would make sense to alias Enumerable::Enumerator#with() to with_object() because polluting the name space limited to Enumerator would only do little harm.

P.S.
You may want to search for a past discussion:
http://blade.nagaokaut.ac.jp/cgi-bin/vframe.rb/ruby/ruby-core/17084?16896-17624

#3 - 07/04/2012 09:09 PM - merborne (kyo endo)

knu (Akinori MUSHA) wrote:

> I doubt it. Enumerable is often just one probably minor aspect of an including class, and even when used with an array or hash it is not obvious from the name with or with_object that it would iterate over the contents. I'd say it's too bold.

Without a block, each_with_object does not start iteration to the elements, just returns a Enumerator object. Operation to the elements is passed to next method.

```
[10,20,30].each_with_object(3).max_by { |i, o| i%o } # => [20, 3]

['ruby', 'python', 'haskell'].each_with_object('ist').map(&:+) # => ["rubyist", "pythonist", "haskellist"]
```

In this example, each_with_object only bind a passed object to each element. So I feel a word each is less appropriate for this, only with is more appropriate.

```
[10,20,30].with(3).max_by { |i, o| i%o } # => [20, 3]

['ruby', 'python', 'haskell'].with('ist').map(&:+) # => ["rubyist", "pythonist", "haskellist"]
```

When each_with_object takes a block, the iteration start. so I agree that the meaning is less clearer without a word each in this case. However, let me think about Array#each or Hash#each. Thease each methods, when they takes a block, return self, not block result. From this, I expect a method that have a word each returns self. or makes irregular iterations like each_cons, each_slice.

As long as it is used to enumerable object, with a block, I think a word with still works same as each_with_object, just like map or inject.

> P.S.
> You may want to search for a past discussion:
> http://blade.nagaokaut.ac.jp/cgi-bin/vframe.rb/ruby/ruby-core/17084?16896-17624

This is very helpful for me. thank you.

#4 - 07/25/2012 02:40 PM - merborne (kyo endo)

=begin
It is obvious that many rubyists have troubled with Enumerable#inject when they use it with a hash.

```
Feature #5662: inject-accumulate, or Haskell's mapAccum* - ruby-trunk - Ruby Issue Tracking System http://bugs
.ruby-lang.org/issues/5662

Ruby: inject issue when turning array into hash - Stack Overflow http://stackoverflow.com/questions/10575052/r
uby-inject-issue-when-turning-array-into-hash 'Ruby: inject issue when turning array into hash - Stack Overflo
w'

Ruby inject with intial being a hash - Stack Overflow http://stackoverflow.com/questions/9434162/ruby-inject-w
ith-intial-being-a-hash 'Ruby inject with intial being a hash - Stack Overflow'

Ruby (Rails) #inject on hashes - good style? - Stack Overflow http://stackoverflow.com/questions/3230863/ruby-
rails-inject-on-hashes-good-style 'Ruby (Rails) #inject on hashes - good style? - Stack Overflow'
```

It is obvious that one of the best solution to solve the above is using Enumerable#each_with_object.

However, no one use it because of its lengthy name.

In the source of ruby 1.9.3, #inject hits 99, but #each_with_object hits only 6. And 6 are tests for the mehtod..

```
source/ruby% grep 'inject' **/*.rb | wc -l
      99
source/ruby% grep 'each_with_object' **/*.rb | wc -l
       6
```

I still believe that shorten the name solves the problem.

As knu sited, there was a long discussion over the its naming.

((URL:http://blade.nagaokaut.ac.jp/cgi-bin/vframe.rb/ruby/ruby-core/17084?16896-17624))

I have found, in the discussion, David Flanagan suggested the name "with" for it.

```
[ruby-core:17191]
I'm partial to the name "with", but knu is worried that it might
conflict with a future reserved word.  If we can't come up with anything
more expressive I'd prefer with_object or with_value to with_memo.
```

It was 2008. "with" keyword have not been emerged in 2012 yet.

Any idea?
=end

#5 - 07/25/2012 09:01 PM - alexeymuranov (Alexey Muranov)

I think #each_with_object(obj) mostly makes sense for obj of container class, like Hash or Array, and less sense with Integer or fixed string:

['ruby', 'python', 'haskell'].with('ist').map(&:+) # => ["rubyist", "pythonist", "haskellist"]

vs

['ruby', 'python', 'haskell'].map { |x| "#{ x }ist" } # => ["rubyist", "pythonist", "haskellist"]

So, maybe instead of with, use memo?

hash = [1, 2, 3].memo({}).each { |e, m| m[e] = e*e } # => {1=>1, 2=>4, 3=>9}

vs

hash = {}.tap { |m| [1, 2, 3].each { |e| m[e] = e*e } } # => {1=>1, 2=>4, 3=>9}

#6 - 07/25/2012 10:13 PM - trans (Thomas Sawyer)

=begin
+1 for #with. Would be great if matz pulled a mikey on this one!

Don't like #memo, I have used that for memoization before.
=end

#7 - 07/26/2012 11:09 AM - nobu (Nobuyoshi Nakada)

I don't think "with" is nice as a method name.
What does it `with' the argument?
"each" should not be omitted, I guess.

#8 - 07/26/2012 01:37 PM - merborne (kyo endo)

nobu (Nobuyoshi Nakada) wrote:

> I don't think "with" is nice as a method name.
> What does it `with' the argument?
> "each" should not be omitted, I guess.

It iterate(or enumerate) with the argument. The receiver, by itself tells it, I think.
I think #each tells nothing. it suggests the returning value is self instead.
This method returns a passed object...

How about #return_with ?

#9 - 07/26/2012 03:07 PM - trans (Thomas Sawyer)

The name #return_with has good semantic, although it may confuse that it would return from method. But I fear it may still be too long. I think the utility of the method warrants very concise name. It is interesting about #each_with_object. We all see it's utility but avoid it b/c then name is too long. It may also be b/c it is poor semantics but definitely more b/c it is too long. For this reason I think #with is pretty good, but if not good enough then here are some concise alternatives that might work: #make, #give or #cons (perhaps short for #construct).

Btw, is ok if default argument is {}? I believe it is likely to be most common case.

#10 - 07/27/2012 07:51 PM - Eregon (Benoit Daloze)

nobu (Nobuyoshi Nakada) wrote:

> I don't think "with" is nice as a method name.
> What does it `with' the argument?
> "each" should not be omitted, I guess.

What about #each_with then? (and #with for Enumerator)
I think it's a reasonable alternative.
And "_object" really seems less than necessary (everything is an object!).

merborne (kyo endo) wrote:

> How about #return_with ?

#return_with does not imply iteration to me at all (and the receiver type can not always be easily known).

#11 - 10/28/2012 12:17 AM - yhara (Yutaka HARA)

- Category set to core

- Target version set to 2.6

#12 - 11/18/2012 12:32 PM - Anonymous

I think the word "object" is non-essential in both Enumerable#each_with_object and Enumerator#with_object; which should become respectively Enumerable#each_with and Enumerator#with.

#13 - 05/10/2014 03:23 PM - akr (Akira Tanaka)

- Related to Feature #7340: 'each_with' or 'into' alias for 'each_with_object' added

#14 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)