Ruby master - Feature #7341

Enumerable#associate

11/13/2012 08:29 AM - nathan.f77 (Nathan Broadbent)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Jeremy Kemper proposed Enumerable#associate during the discussion in #7297, with the following details:

Some background:

#4151 proposes an Enumerable#categorize API, but it's complex and hard to understand its behavior at a glance.
#7292 proposes an Enumerable#to_h == Hash[...] API, but I don't think of association/pairing as explicit coercion, so #to_h feels misfit.

Associate is a simple verb with unsurprising results. It doesn't introduce ambiguous "map" naming. You associate an enumerable of keys with yielded values.

Some before/after examples:

Before: Hash[filenames.map { |filename| [filename, download_url(filename)]}]
After: filenames.associate { |filename| download_url filename }

=> {"foo.jpg"=>"http://...", ...}

Before: alphabet.each_with_index.each_with_object({}) { |(letter, index), hash| hash[letter] = index }
After: alphabet.each_with_index.associate

=> {"a"=>0, "b"=>1, "c"=>2, "d"=>3, "e"=>4, "f"=>5, ...}

Before: keys.each_with_object({}) { |k, hash| hash[k] = self[k] } # a simple Hash#slice
After: keys.associate { |key| self[key] }

It's worth noting that this would compliment ActiveSupport's Enumerable#index_by method:
http://api.rubyonrails.org/classes/Enumerable.html#method-i-index_by
#index_by produces '{ => el, ...}', while #associate would produce '{el => , ...}'.

For cases where you need to control both keys and values, you could use '[1,2,3].map{|i| [i, i * 2] }.associate', or continue to use 'each_with_object({})'.

History

#1 - 11/14/2012 12:03 AM - bitsweat (Jeremy Daer)

Thanks for posting, Nathan. See https://gist.github.com/4035286 for the full pitch and a demonstration implementation.

In short: associating a collection of keys with calculated values should be easy to do and the code should reflect the programmer's intent. But it's hard for a programmer to discover which API is appropriate to achieve this. Hash[] and each_with_object({}) seem unrelated. And using these API requires boilerplate code that obscures the programmer's intent.

Must write code to build a Hash[] argument in the format it expects: an array of [key, value] pairs. The intent is hidden by unrelated code needed to operate the Hash[] method.

Hash[*collection.map { |elem| [elem, calculate(elem)] }]

This is better. Much less boilerplate code. But the programmer is reimplementing association every time: providing a hash and setting the value for each key in the collection. This is what an *implementation* of association looks like. It shouldn't be repeated in our code.

collection.each_with_object({}) { |elem, hash| hash[elem] = calculate(elem) }

Most Rubyists just use this instead. It uses simple, easy-to-discover API. But it suffers the same issues: it's an *implementation* of association that's now repeated in our code, blurring its intent. And it forces us to disrupt chains of enumerable methods and write boilerplate code.

hash = {}
collection.each { |element| hash[element] = calculate(element) }

Now the code is stating precisely what the programmer wants to achieve. Associate is easy to find in docs and uses a verb that "rings a bell" to programmers who need to associate keys with yielded values.

collection.associate { |element| calculate element }

Marc-André Lafortune proposed a similar Enumerable#associate in #4151. The basic behavior is the same, so I consider that a point in favor of this method name. It associates values with the enumerated keys. He introduces additional collision handling that I consider out of scope. For more complex scenarios, using more verbose, powerful API like #inject, #each_with_object, or #map + #associate feels appropriate.

#2 - 11/14/2012 01:12 AM - marcandre (Marc-Andre Lafortune)

- Category changed from lib to core

- Priority changed from 3 to Normal

Hi,

bitsweat (Jeremy Kemper) wrote:

> In short: associating a collection of keys with calculated values should be easy to do and the code should reflect the programmer's intent.

A strong +1 from me

> See https://gist.github.com/4035286

A good start. I'd make one important change: return an enumerator when no block is given. Here's why:

1) The form you suggest would be redundant with Enumerable#to_h

2) It would be more powerful, for example to associate things that need an index...

```
rng.each_with_index.associate {|elem, index| ....} # => { [elem, index] => ... }, not what you want
# Easy this form:
rng.associate.with_index {|elem, index| ... }  # => { elem => ... }
```

3) Consistency with modern methods dealing with enumerable.

#3 - 11/14/2012 04:53 AM - nathan.f77 (Nathan Broadbent)

1) The form you suggest would be redundant with Enumerable#to_h

I agree that 'Enumerable#to_h' would seem more appropriate than the block-less version of 'associate'. To me, the 'associate' verb implies that the programmer will provide some logic to determine how the elements will be associated. So I also feel that invocation without a block should return an enumerator.

However, if 'to_h' is rejected and 'associate' is all we have to work with, then it would probably be more useful to make 'associate' 'multi-purpose' in the way that is currently proposed.

#4 - 11/18/2012 01:34 PM - Anonymous

Agree with Marc-Andre.

#5 - 11/18/2012 08:35 PM - trans (Thomas Sawyer)

=begin
One problem I have with this is the terminology. The term "associate" already applies to arrays. ((*Associative arrays*)) are arrays of arrays where the first element of an inner array acts a key for the rest.

```
[[:a,1],[:b,2]].assoc(:a)  #=> [:a,1]
```

For this reason I would expect an #associate method to take a flat array and group the elements together.

```
[:a,1,:b,2].associate  #=> [[:a,1],[:b,2]]
```

An argument could determine the number elements in each group, the default being 2.

Since Hash#to_a returns an associative array, to me it makes sense that Array#to_h would reverse the process.

```
{:a=>1,:b=>2}.to_a    #=> [[:a,1],[:b,2]]
[[:a,1],[:b,2]].to_h  #=> {:a=>1,:b=>2}
```

Putting the two together, your version of associate is easy enough to achieve:

```
[:a,1,:b,2].associate.to_h
```

As it turns out, with the exception of the default argument, #associate is same as #each_slice. But I think it would be nice to have #associate around for it's default and the fact that it reads better in these cases.

=end

#6 - 11/20/2012 06:55 AM - Anonymous

@Tom: Associative arrays are nice, but they are just arrays. No need to pamper them too much in the core.

#7 - 06/16/2014 03:35 PM - Ajedi32 (Andrew M)

This is related to #6669

#8 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)