Ruby master - Feature #6596

New method for Arrays : Array#indexes

06/15/2012 06:05 PM - robin850 (Robin Dupret)

Status:	Assigned	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

Hello

5 days ago, I submitted a pull request on Github which provides a new method for the array objects which is Array#indexes. I have fist edit the Array#index method in order it to return an array of indexes and not a single index (which is the first occurrence it finds). I found it more logical but a user (trans) tells us that it could break the contract of Array#index so I decided to move it into Array#indexes. Eric (drbrain) tells me I should reasonning why I want to add this method ; it's just a point of view : I don't really understand why Array#index return a single index if the parameter is in the array several times.

Examples

a = [1, 2, 3, 1]
a.indexes(1)
Return : [0, 3]
a.index(1)
Return : 0
In my opinion, it's not really logical, 1 is in the array twice

Moreover, this pull request doesn't beak anything because we don't edit the Array#index method so programms which were created with previous version of Ruby will work.

I hope my post is complete. Have a nice day.

History

#1 - 06/15/2012 09:46 PM - knu (Akinori MUSHA)

> a = [1, 2, 3, 1]
> a.indexes(1)
> Return : [0, 3]

Is there any proof that such a new functionality has a certain amount of need?

> a.index(1)
> Return : 0
> In my opinion, it's not really logical, 1 is in the array twice

We also have Array#rindex and it looks completely logical for us with some C background.

#2 - 06/16/2012 12:09 AM - robin850 (Robin Dupret)

Hello Knu,

Array#rindex return the last occurrence of the parameter. Array#indexes return all of the indexes.

> Is there any proof that such a new functionality has a certain amount of need?

Not really but I think it doesn't revoke anything, it's a good adding in my opinion.

#3 - 06/16/2012 03:28 AM - trans (Thomas Sawyer)

@kyu Try doing the equivalent of #indexes without it. Not that it's especially hard, but you have to stop and work out a solutuon. When you need it, that's when you wish there were already a method for it.

#4 - 06/16/2012 04:42 PM - knu (Akinori MUSHA)

- Status changed from Open to Feedback

You haven't shown any real use case yet.
I don't deny that the functionality alone would be a good shortcut itself just like any other proposal we see every day.

But for example, if you just want to use it to delete the matching elements then it will be nothing better than Array#delete.
If I am to replace each element then I'll write a map!.with_index loop, or just use each_with_index in general.

It's all about how you would use it and what the supposed context would look like.

#5 - 06/17/2012 06:27 AM - trans (Thomas Sawyer)

=begin
knu (Akinori MUSHA) I think you have already pointed out the usecase. Wherever someone is using #each_with_index which contains a conditional selection would be a potential use case.

even_indexes = []
array_of_numbers.each_with_index do |c, i|
next if c % 2 == 1
even_indexes << i
end

Basically #indexes is to #index as #select is to #find. We could do without #select too, but why would we?
=end

#6 - 06/17/2012 11:24 AM - knu (Akinori MUSHA)

trans (Thomas Sawyer) That's what it does, not a use case. I am questioning you how it is useful to get an array of indices.

#7 - 06/17/2012 10:39 PM - trans (Thomas Sawyer)

=begin
https://github.com/apillet/invaders/blob/ab32d1704e20cfffe472458b04c16ce82353dabe/classes/enemy_grid.rb

```
@table.each_with_index do |row, rindex|
  row.each_with_index do |column, cindex|
    if column.nil? then
      @table[rindex][cindex] = enemy
      return
    end
  end
end
```

Could have been written:

```
@table.each_with_index do |row, rindex|
  row.indexes(nil).each do |cindex|
    @table[rindex][cindex] = enemy
    return
  end
end
```

=end

#8 - 06/17/2012 11:02 PM - nobu (Nobuyoshi Nakada)

=begin
Assuming the return value has no meanings.

@table.any? do |row|
if cindex = row.index(nil)
row[cindex] = enemy
true
end
end
=end

#9 - 06/18/2012 01:47 AM - trans (Thomas Sawyer)

Yea, I would have written is completely different myself (even from yours). All I did was search GitHub for #each_with_index for code in which one could use #indexes. This was the first one I found. Unfortunately, as you can imagine, such a search is time consuming. Worse still GitHub's search turned out to have a bug and I could not go beyond page 100 of search results. There in lies one of the problem with "real" use case though --there's always dozens, if not more, way to do something. When you know you don't have a method at your disposal obviously you are going to think of ways to do it that won't require it.

So it can be very difficult to find a case until you are the one that realizes you could use it. And even when you do, lacking the method at hand, before any traction is made in any discussion like this you've already refactored in some fashion or anther to get things to work and have moved on. I know I've used a form of #indexes before but on a String, not an Array. And I remember being frustrated/surprised there was no method for it. But that was years ago now and I don't recall where it was.

The fact is, sometimes a method makes sense not b/c you have a shining example at hand, but b/c it fills an obvious hole in functionality. We have #index to find the first index. We have #rindex to find the last index. So it becomes obvious to ask about all the other indexes in between. Why do we have to fallback to writing our own loops to get at those? While I agree, it won't find a great deal of use, there are many methods like that. And yet it's nice that they are around when you do need them.

In considering the nature of this method further, I think the most quintessential use would be when you have two arrays corresponding by index and you want to locate the elements of one array based on the positions of a select elements of another. e.g.

```
a1 = [:a, :b, :c, :b, :c]
a2 = [:x, :z, :z, :y, :y]

g[:b] = a2.indicies(*a1.select_indexes(:b))
g[:b] #=> [:z, :y]
```

Yea, yea. Not a "real" use case. We all know code like that exists (and it pretty common back in the days of procedural coding). But I am not going to spend days searching through source code for it.

BTW, I just notice Enumerable#find_index was added to the language (in 1.9.2?). So by analogy this could be Enumerable#select_index, or #select_indexes, right? So that why I used that in the last example.

#10 - 06/18/2012 04:07 AM - knu (Akinori MUSHA)

Since we introduced Enumerator (and Enumerable::Lazy recently) we are skeptic about adding a new method returning a complete array rather than one generating an enumeration, especially when a result array is unlikely ever to be used as a final result.
As I already said, generating an array just for iterating with it is nonsense.

If you start to say that #select_indexes may be what you want by now, then you haven't analyzed the desired feature enough.

I'd like to hear something from the originator before closing this issue.

#11 - 06/18/2012 05:28 AM - trans (Thomas Sawyer)

I don't see any reason it can't return an Enumerator.

My last example doesn't iterate over it.

#select_indexes is the exact same thing, just generalized to Enumerable instead of Array. You have to forgive me for not being omniscient, as I just spent the last two days "analyzing" this (in addition to the time I spent a few years back for String#index_all).

I doubt you will.

#12 - 06/18/2012 12:25 PM - knu (Akinori MUSHA)

=begin
You are changing the subject. This issue is about Array#indexes, and if you talk about generalization, you should work out a concrete proposal with convincing examples to open another issue.

For that matter, I don't think "indexes" could or should be generalized to Enumerable because Enumerable currently does not have the sense of index.

Lastly, even if you sometimes need to collect indices you can simply use #select with ease:

```
indexes = (0...a.size).select { |i| a[i] % 2 == 1 }

#enumurator = (0...a.size).lazy.select { |i| a[i] % 2 == 1 }
```

Considering that indices are only useful when you access the originating array with them, the array should have a name and there should be no problem if you can't do this in a method chain.
=end

#13 - 06/18/2012 07:23 PM - regularfry (Alex Young)

On 18/06/12 04:25, knu (Akinori MUSHA) wrote:

> Issue #6596 has been updated by knu (Akinori MUSHA).
>
> =begin
> You are changing the subject. This issue is about Array#indexes, and if you talk about generalization, you should work out a concrete proposal

with convincing examples to open another issue.

For that matter, I don't think "indexes" could or should be generalized to Enumerable because Enumerable currently does not have the sense of index.

Lastly, even if you sometimes need to collect indices you can simply use #select with ease:

```
indexes = (0...a.size).select { |i| a[i] % 2 == 1 }
```

```
#enumurator = (0...a.size).lazy.select { |i| a[i] % 2 == 1 }
```

Considering that indices are only useful when you access the originating array with them, the array should have a name and there should be no problem if you can't do this in a method chain.
=end

Index lists *can* be useful without the original array. A couple of
years ago I had the job of turning a big CSV into a database schema.
The first step in this was to work out which columns in the CSV are
correlated. Here's a simplified example:

```
$ cat example.csv
```

NAME,TYPE,LABEL
aleph,a,foo
aleph,b,foo
beth,a,bar

See how NAME and LABEL vary together? I want them to end up in the same
table, with two rows in it. You can group columns into tables by
generating an Array#indexes list for each unique entry in each column in
order, then seeing if the columns match by checking *just the index
lists* for for equality, ignoring the original column data.

--
Alex

Bug #6596: New method for Arrays : Array#index
https://bugs.ruby-lang.org/issues/6596#change-27286

Author: robin850 (Robin Dupret)
Status: Feedback
Priority: Normal
Assignee:
Category: core
Target version: 2.0.0
ruby -v: 2.0.0

Hello

5 days ago, I submitted a pull request on Github which provides a new method for the array objects which is Array#indexes. I have fist edit the Array#index method in order it to return an array of indexes and not a single index (which is the first occurrence it finds). I found it more logical but a user (trans) tells us that it could break the contract of Array#index so I decided to move it into Array#indexes. Eric (drbrain) tells me I should reasonning why I want to add this method ; it's just a point of view : I don't really understand why Array#index return a single index if the parameter is in the array several times.

Examples

a = [1, 2, 3, 1]
a.indexes(1)
Return : [0, 3]
a.index(1)
Return : 0
In my opinion, it's not really logical, 1 is in the array twice

Moreover, this pull request doesn't beak anything because we don't edit the Array#index method so programms which were created with previous version of Ruby will work.

I hope my post is complete. Have a nice day.

#14 - 06/18/2012 11:53 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 17-06-2012 13:47, trans (Thomas Sawyer) escreveu:

> Yea, I would have written is completely different myself (even from yours). All I did was search GitHub for #each_with_index for code in which one could use #indexes.

I guess you didn't get the idea of the policy here... Usually people
will come with suggestions on improvements to Ruby syntax or core
library through real issues taken from real programming on existing
projects.

We usually don't try to anticipate issues and use cases. We start with
them, so usually it is not hard to have a concrete example to
demonstrate how that feature would be useful...

If you're having to look at GitHub, it seems like you didn't find a
situation yourself where the requested feature would make you happier.

I would advice you not to attempt to find use cases beforehand, just let
them find you.

#15 - 08/07/2012 03:11 PM - trans (Thomas Sawyer)

=begin
Given a ordered list, swap items relative to those that match.

list.index_all{ |x| match?(x) }.each do |i|
list[i], list[i+1] = list[i+1], list[i]
end

vs. what?

list.each_with_index do |x, i|
if match?(x)
list[i], list[i+1] = list[i+1], list[i]
end
end

Nope, that doesn't work. Could clone list but that would be rather inefficient. Better idea? A built-in #index_all is going to be a lot faster then a Ruby code implementation of the same.

=end

#16 - 10/27/2012 09:24 AM - ko1 (Koichi Sasada)

- Assignee set to mame (Yusuke Endoh)

Is it a bug or a feature?

#17 - 10/27/2012 06:58 PM - mame (Yusuke Endoh)

- Status changed from Feedback to Assigned

- Assignee changed from mame (Yusuke Endoh) to matz (Yukihiro Matsumoto)

- Target version changed from 2.0.0 to 2.6

ko1 (Koichi Sasada) wrote:

> Is it a bug or a feature?

I think it is a feature, in every respect.

--
Yusuke Endoh mame@tsg.ne.jp

#18 - 10/27/2012 09:57 PM - nobu (Nobuyoshi Nakada)

- Tracker changed from Bug to Feature

#19 - 11/19/2012 09:35 AM - zzak (Zachary Scott)

- % Done changed from 100 to 0

#20 - 11/20/2012 03:15 AM - robin850 (Robin Dupret)

Any new on this please ? I see that this ticket is assigned to Matz. Is there any chance to see it implemented in ruby ?

#21 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#22 - 10/31/2018 07:11 PM - TylerRick (Tyler Rick)

- Subject changed from New method for Arrays : Array#index to New method for Arrays : Array#indexes