Ruby - Feature #13795

Hash#select return type does not match Hash#find_all

08/09/2017 04:08 PM - davidarnold (David Arnold)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Enumerable#select and Enumerable#find_all are aliases. Hash is_a Enumerable, yet only Hash#select was overridden to return a Hash, with Hash#find_all still returning an Array. This is confusing since the message is that you can use select and find_all interchangeably for Enumerable, yet when you get to Hash, there are warnings that it is no longer true.

Also any code that expects to call select on an Enumerable and get an array back (as documented) could break, but only for Hash#select.

Example:

```
def select_many(*enumerables, &block)
  result = []
  enumerables.each do |e|
    result.concat e.select(&block)
  end
  result
end

select_many([1, 2], [3, 4]) { |x| x % 2 == 0 } #=> [2, 4]

select_many({ 1 => 2 }, { 3 => 4 }) { |k, v| k < 2 }
#=> TypeError: no implicit conversion of Hash into Array
```

Should Hash#find_all also return a Hash for consistency? Or, given the fact that calling #to_h on the resulting Array is so easy, should Hash#select revert to the Enumerable behavior of returning an Array?

Proposal 1:

```
h = { "a" => 100, "b" => 200, "c" => 300 }
h.find_all {|k,v| k > "a"}  #=> {"b" => 200, "c" => 300}
```

Proposal 2:

```
h = { "a" => 100, "b" => 200, "c" => 300 }
h.select {|k,v| k > "a"} #=> [["b", 200], ["c", 300]]
h.select {|k,v| k > "a"}.to_h #=> {"b" => 200, "c" => 300}
```

History

#1 - 08/09/2017 04:25 PM - adp90 (Alexander Patrick)

This might make the next version of Ruby not backwards-compatible in some instances. It might be better to leave the methods as they are.

There are instances where aliases to the same C functions act differently, such as lazy_super (aliases: chunk, slice_before, slice_after, slice_when, chunk_while).

The solution might be to update the documentation for find_all and select so that their behavior is clear.

It should also be noted that while select has specs and tests for Enumerable, Array, Hash, Set, and SortedSet, find_all only has spec and test for Enumerable. If it is decided that the methods shouldn't be changed, it would be a good idea to better document and test find_all.

#2 - 08/09/2017 05:47 PM - davidarnold (David Arnold)

- ruby -v set to ruby 2.4.1p111 (2017-03-22 revision 58053) [x86_64-darwin16]

> This might make the next version of Ruby not backwards-compatible in some instances. It might be better to leave the methods as they are.

Ruby has been not backwards-compatible in several releases, so I don't see that as a reason per se to not entertain the two proposals. I do agree with you that the documentation and tests should be improved if the decision is made to leave things as-is.

My speculation is that the custom functionality of Hash#select was very useful when it was introduced in 1.9, as Array#to_h did not exist yet. However with 2.1, when to_h became an option, the disjointed behavior became less useful, and more of an inconsistency.

```
h.select {|k,v| k > "a"}.to_h

# vs

h.select {|k,v| k > "a"} # did this really save that much effort?
```

I feel like breaking the contract of Enumerable#select is bad, but I will fully admit that Hash#select returning a Hash is probably a very popular feature!

Either way, back to the topic of the bug, I am not sure why find_all was left behind through all this.

#3 - 08/09/2017 06:30 PM - adp90 (Alexander Patrick)

davidarnold (David Arnold) wrote:

> Ruby has been not backwards-compatible in several releases, so I don't see that as a reason per se to not entertain the two proposals.

You have a good point. This wouldn't be a big problem.

I would like to suggest a third possible option: if Hash#select returning a Hash is useful, why not expand on this so that one of them always returns the input data structure?
Select could be changed to only return arrays, and find_all can be changed to return the same type of data structure it was given.

Proposed behavior:

```
s = Set.new [1, 2, 3, 4, 5]
=> #<Set: {1, 2, 3, 4, 5}>
s.find_all { |x| x.even? }
=> #<Set: {2, 4}>
s.select { |x| x.even? }
=> [2, 4]
```

This would make them non-equivalent, but could be useful enough to warrant it.
Also, find_all's lack of existing spec and tests would make it easy to change it in this way without removing anything from the codebase.

#4 - 08/09/2017 08:37 PM - davidarnold (David Arnold)

> I would like to suggest a third possible option: if Hash#select returning a Hash is useful, why not expand on this so that one of them always returns the input data structure?

So, I think people do like having a way to get the same data type back that you started with, however, that substantially increases the burden of being an Enumerable implementor. The nice thing about Enumerable is that you are supposed to just have to implement an #each method and you get all the Enumerable functionality for "free" when you include the module.

If there is a convention where every class that includes Enumerable has to have a set of functions that returns an instance of the same class, you wouldn't just have to implement #each, you'd have to implement over half of the Enumerable interface yourself. The value of Enumerable as an includable module drops to almost nothing at that point.

> Select could be changed to only return arrays, and find_all can be changed to return the same type of data structure it was given.

I think if the inconsistency is being retained intentionally, most people would vote to leave things as-is with Hash and expand the ad hoc implementation to #select on the other Enumerable classes instead of #find_all.

Your example with Set is interesting because currently both methods return Array, but it doesn't matter too much because you can always call #to_set on the result.

```
s.select { |x| x.even? }.to_set #=> #<Set: {2, 4}>
```

Which comes back to my observation that Ruby 2.1's #to_h method can be used the same way to make the ad hoc implementation of Hash#select unnecessary.

#5 - 08/10/2017 06:31 AM - adp90 (Alexander Patrick)

davidarnold (David Arnold) wrote:

> If there is a convention where every class that includes Enumerable has to have a set of functions that returns an instance of the same class, you wouldn't just have to implement #each, you'd have to implement over half of the Enumerable interface yourself. The value of Enumerable as an includable module drops to almost nothing at that point.

You're right. While select/select! are re-implemented outside of Enumerable, this seems to be more to ensure Enumerable/Enumerator work than to provide any special functionality. Overriding other Enumerable methods elsewhere would cause problems.

> Which comes back to my observation that Ruby 2.1's #to_h method can be used the same way to make the ad hoc implementation of Hash#select unnecessary.

I looked for anything that might be broken if Hash#select returns an array, but was unable to find anything.

I also found that Hash#reject once returned a Hash, while Hash#select did not:
https://www.ruby-forum.com/topic/158100

This makes me think there's no good historical reason for this behavior, and no real reason to keep it.

When I have time I'll update Hash#select and see if it breaks anything other than its own spec/tests.

#6 - 08/10/2017 04:12 PM - davidarnold (David Arnold)

I did some research to explore the idea that *more* Enumerable classes should implement #select (and even #collect, #drop, etc) to return their own type, since that sounds appealing on the surface.

While classes like Array, Hash, and Set work pretty cleanly, other less thought about classes like Range or -- even worse -- IO, could not possibly return their own type. So, it seems more logical to me to just standardize on returning Array from Enumerable methods as documented than try to spread the behavior further.

Also, as a side note, it seems that Hash#select and #reject really are the only two methods that behave this way. And even inside Hash, methods like #collect, #collect_concat, #drop, #drop_while, #grep, #max, #max_by, #min, #min_by, #sort, #sort_by, #take, and #take_while are all still returning Array.

#7 - 08/11/2017 01:08 PM - adp90 (Alexander Patrick)

I made a pull request: https://github.com/ruby/ruby/pull/1674

#8 - 08/12/2017 06:30 AM - shevegen (Robert A. Heiler)

> Ruby has been not backwards-compatible in several releases, so I don't see that as a reason per se to not entertain the two proposals.

It all depends on matz, whether he agrees with the assessment or whether he does not. I won't comment on the proposal and discussion as such, that is for matz, the core team and you guys - but I want to mention at the least ~2 presentations by matz in the ... last 2 years or so (he does not do that many but you should be able to find them on youtube). In at the least one, he explained that a big reason as to why some "fixes" to ruby 2.x will not be done is because of backwards compatibility.

I forgot which examples he used, perhaps it was some legacy perl variables or perhaps it was @@class variables, I honestly do not remember (I think he brought some specific example by code that ... some ruby core maintainer used but I honestly do not remember right now).

In the worst case, ruby 3.x may accept backwards-incompatible changes so either way I think you can continue to reason in favour of your proposal just fine.

#9 - 08/12/2017 11:11 AM - duerst (Martin Dürst)

On 2017/08/11 01:12, david.n.arnold@gmail.com wrote:

> Also, as a side note, it seems that Hash#select and #reject really are the only two methods that behave this way. And even inside Hash, methods like #collect, #collect_concat, #drop, #drop_while, #grep, #max, #max_by, #min, #min_by, #sort, #sort_by, #take, and #take_while are all still returning Array.

Let's analyze this a bit.

#max, #max_by, #min, #min_by

These return a single key, value pair as an array. That's much easier to handle than a hash with a single key, value pair, and is also quite a bit more efficient internally (although we shouldn't care). That feels right to me.

#grep
In the case of a hash, the argument to #grep has to be a two-element array ([key, value]). Because there is only one entry with a given key in the hash, the result will also be a single two-element array,

#collect, #collect_concat
The block for collect can return anything. Putting these 'anythings' together again can only be done with an array, not with a hash. As an example, think about something like:
h.collect {|k, v| "#{k}-#{v}"}
We get back strings, and there is no such thing as a Hash of Strings.

#sort, #sort_by
Although Hashes now have order, that wasn't always the case. And Hashes are still not considered the prime class when dealing with order. As an example, they don't allow easy access to an element at a certain position in the order.

#drop, #drop_while, #take, #take_while
Similar considerations to those for #sort apply here, although maybe not that strongly.

Bug #13795: Hash#select return type does not match Hash#find_all
https://bugs.ruby-lang.org/issues/13795#change-66129

- Author: davidarnold (David Arnold)
- Status: Open
- Priority: Normal
- Assignee:
- Target version:
- ruby -v: ruby 2.4.1p111 (2017-03-22 revision 58053) [x86_64-darwin16]
- Backport: 2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

#10 - 08/12/2017 02:53 PM - adp90 (Alexander Patrick)

Changing the behavior of Hash#select/reject broke a lot of files, some in very strange ways.

This makes me think that such a change won't be popular. Consistency with previous versions of Ruby and with legacy code might outweigh being consistent within Enumerable.

Also, with select! and reject! not actually being Enumerable methods, changing select and reject to return Arrays makes them inconsistent with Hash#select! and Hash#reject!

The best solution to this might be to update the documentation to point out the difference between select and find_all.

I've closed the pull request for the time being, as this issue likely warrants further discussion.

#11 - 08/12/2017 04:00 PM - davidarnold (David Arnold)

> Also, as a side note, it seems that Hash#select and #reject really are the only two methods that behave this way. And even inside Hash, methods like #collect, #collect_concat, #drop, #drop_while, #grep, #max, #max_by, #min, #min_by, #sort, #sort_by, #take, and #take_while are all still returning Array.

> Let's analyze this a bit.
> [...]

I agree that all those methods are better in returning Array. My pointing out the others wasn't meant to imply that they should be converted to returning Hashes. My intention was to point out as a developer you have to remember that every Enumerable method in Hash returns an Array except select and reject.

Beyond the mental catch there, my strongest reason to offer for why those methods should also return Array is that any code receiving a Hash in its aspect of being an Enumerable will receive the *wrong type* per the Enumerable documentation. That means that anything that uses these methods

below will fail:

```
Array.instance_methods - Hash.instance_methods
#=> [:join, :rotate, :rotate!, :sort!, :sort_by!, :collect!, :map!, :delete_at, :transpose, :fill, :uniq!, :sh
uffle!, :shuffle, :sample, :combination, :repeated_permutation, :permutation, :repeated_combination, :product,
 :flatten!, :bsearch_index, :bsearch, :&, :*, :+, :-, :pack, :|, :insert, :rindex, :<<, :reverse, :reverse!, :
concat, :to_ary, :slice, :slice!, :at, :last, :push, :pop, :unshift, :each_index]
```

But, to my bug report, if there is some extremely strong reason why hashes should still be returned, then why was find_all left behind?

#12 - 08/12/2017 04:03 PM - davidarnold (David Arnold)

> In the worst case, ruby 3.x may accept backwards-incompatible changes so either way
> I think you can continue to reason in favour of your proposal just fine.

I fully agree that changing Hash#select and #reject (or even #find_all) should not be a 2.x change!

#13 - 06/24/2019 06:33 PM - jeremyevans0 (Jeremy Evans)

- Tracker changed from Bug to Feature

- ruby -v deleted (ruby 2.4.1p111 (2017-03-22 revision 58053) [x86_64-darwin16])

- Backport deleted (2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN)