Ruby trunk - Feature #9108

Hash sub-selections

11/14/2013 12:00 PM - wardrop (Tom Wardrop)

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

Description

Hi,

I seem to regularly have the requirement to work on a sub-set of key/value pairs within a hash. Ruby doesn't seem to provide a concise means of selecting a sub-set of keys from a hash. To give an example of what I mean, including how I currently achieve this:

```
    sounds = {dog: 'woof', cat: 'meow', mouse: 'squeak', horse: 'nay', cow: 'moo'}
    domestic_sounds = sounds.select { |k,v| [:dog, :cat].include? k }
#=> {dog: 'woof', cat: 'meow'}
```

I think a more concise and graceful solution to this would be to allow the Hash#[] method to take multiple arguments, returning a sub-hash, e.g.

```
    domestic_sounds = sounds[:dog, :cat] #=> {dog: 'woof', cat: 'meow'}
```

I had a requirement in the current project I'm working on to concatenate two values in a hash. If this proposed feature existed, I could of just done this...

```
    sounds[:dog, :cat].values.join #=> 'woofmeow'
```

You could do something similar for the setter also...

```
    sounds[:monkey, :bat] = 'screech'
    sounds
#=> {dog: 'woof', cat: 'meow', mouse: 'squeak', horse: 'nay', cow: 'moo', monkey: 'screech', bat: 'screech'}
```

Concise, convenient and readable. Thoughts?

Related issues:

Related to Ruby trunk - Feature #8499: Importing Hash#slice, Hash#slice!, Has...	**Closed**

History

#1 - 11/14/2013 12:46 PM - phluid61 (Matthew Kerwin)

In your proposal, what would happen with undefined keys? I see two reasonable options:

```
sounds = {dog: 'woof', cat: 'meow'}
# option 1:
sounds[:dog, :fish] #=> {dog: 'woof'}
# option 2:
sounds[:dog, :fish] #=> {dog: 'woof', fish: nil}
```

Of the two, I'd much prefer the first. A third option is to raise an exception, but that seems the least friendly of all.

If approved, I'd be +1 on the multiple-setter as well. I've had scenarios in which I'd have used it had it been available.

We should note the previous feature discussions (can't remember issue numbers) involving nested lookups, which also suggested a multiple-argument #[] semantic.

#2 - 11/14/2013 12:51 PM - phluid61 (Matthew Kerwin)

Apologies for immediately replying again, but I've just had a potential source of confusion occur to me:

```
hash = {a:1, b:2}

keys = [:a,:b]
```

```
hash[*keys] #=> {a:1, b:2}

keys = [:a]
hash[*keys] #=> 1, expected {a:1} ?

keys = []
hash[*keys] #=> ???
```

I'm not against the subset feature, but I think using #[] will cause more trouble than it's worth. Why not use #subset or a similar name?

#3 - 11/14/2013 01:44 PM - nobu (Nobuyoshi Nakada)

wardrop (Tom Wardrop) wrote:

> I think a more concise and graceful solution to this would be to allow the Hash#[] method to take multiple arguments, returning a sub-hash, e.g.
>
> ```
> domestic_sounds = sounds[:dog, :cat] #=> {dog: 'woof', cat: 'meow'}
> ```

As sounds[:dog] returns 'woof', it should return the values only, even if it were introduced.

> I had a requirement in the current project I'm working on to concatenate two values in a hash. If this proposed feature existed, I could of just done this...
>
> ```
> sounds[:dog, :cat].values.join #=> 'woofmeow'
> ```

Try:

```
sounds.values_at(:dog, :cat).join('')
```

> You could do something similar for the setter also...
>
> ```
> sounds[:monkey, :bat] = 'screech'
> sounds
> #=> {dog: 'woof', cat: 'meow', mouse: 'squeak', horse: 'nay', cow: 'moo', monkey: 'screech', bat: 'screech
> '}
> ```

It feels ambiguous, since it looks like a kind of mulitple assignment to me.

Rather it should be:

```
sounds[:monkey, :bat] = 'screech'
# sounds[:monkey] == 'screech'
# sounds[:bat]    == nil

sounds[:cock, :hen] = 'cock-a-doodle-doo', 'cluck'
# sounds[:cock]   == 'cock-a-doodle-doo'
# sounds[:hen]    == 'cluck'
```

shouldn't it?

#4 - 11/14/2013 04:12 PM - alexeymuranov (Alexey Muranov)

I think, in *Rails*, the proposed method (not the assignment) is called Hash#slice.

I think it is impossible to use #[] for that method:

```
h = {1 => 2}
h[1] # => {1 => 2}?
     # => [2]?
     # => Set[2]?
     # => 2?
```

#5 - 11/14/2013 07:50 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

related to #8499 and the rejected #6847.

Personally I'd love to see Hash#slice implemented in Ruby core, but I think #[] shouldn't work as #slice. It should instead return the values only for the requested keys.

#6 - 11/14/2013 09:48 PM - BertramScharpf (Bertram Scharpf)

```
[:dog, :cat].map { |k| sounds[ k] }
```

```
#=> ["woof", "meow"]

[:dog, :cat].inject Hash.new do |r,k| r[ k] = sounds[ k] ; r end
#=> {:dog=>"woof", :cat=>"meow"}
```

If you look at it this way, one should rather define an Array method than
a Hash method.

#7 - 11/15/2013 03:30 AM - prijutme4ty (Ilya Vorontsov)

I prefer such syntax for nested hash (as in http://bugs.ruby-lang.org/issues/5531)
Why not to use usual method like #key_values_at or #subhash or smth like that?

#8 - 11/15/2013 08:56 AM - wardrop (Tom Wardrop)

I suppose square-bracket syntax is too ambiguous as it collides with many other existing and potential behaviours I didn't consider. A normal method is fine.

I think an appropriate solution would be to amend Hash#select. It currently doesn't take any arguments, so could easily take a list of *args. You could use the optional block to further refine this selection if need be. The same for Hash#reject.

One potential issue though is that Hash#select without any arguments currently returns an enumerator. This could be problematic when doing something like hash.select *args with an empty array. I think calling Hash#select without arguments should return a copy of the full original Hash, this keeps it somewhat compatible with the existing behaviour of returning an enumerator (given a Hash is an enumerator), while at the same time making it consistent with the new Hash#select(*keys) functionality.

What do you think of changing #select and #reject to support this?

#9 - 11/15/2013 10:07 AM - nobu (Nobuyoshi Nakada)

Enumerator differs from Hash.

#10 - 11/15/2013 12:07 PM - wardrop (Tom Wardrop)

They do differ, yes, but in most cases an enumerator is interchangeable with a Hash. I can't imagine anyone would be using Hash#select to get an enumerator anyway. If anyone is, then their code deserves to break to some extent. You should use Hash#enum_for or Hash#each methods if you want an enumerator from a hash.

#11 - 11/15/2013 12:23 PM - phluid61 (Matthew Kerwin)

wardrop (Tom Wardrop) wrote:

> They do differ, yes, but in most cases an enumerator is interchangeable with a Hash. I can't imagine anyone would be using Hash#select to get an enumerator anyway. If anyone is, then their code deserves to break to some extent. You should use Hash#enum_for or Hash#each methods if you want an enumerator from a hash.

Do you mean Enumerator (the class returned by many functions when !block_given?) or Enumerable (the module that defines #sort, #reverse, etc.)?

#12 - 11/15/2013 01:06 PM - phluid61 (Matthew Kerwin)

phluid61 (Matthew Kerwin) wrote:

> wardrop (Tom Wardrop) wrote:
>
> > They do differ, yes, but in most cases an enumerator is interchangeable with a Hash. I can't imagine anyone would be using Hash#select to get an enumerator anyway. If anyone is, then their code deserves to break to some extent. You should use Hash#enum_for or Hash#each methods if you want an enumerator from a hash.
>
> Do you mean Enumerator (the class returned by many functions when !block_given?) or Enumerable (the module that defines #sort, #reverse, etc.)?

Sorry, I realise you do mean the right thing. I was put off by the fact that you say they're most often interchangeable; all they have in common is #each

#13 - 11/15/2013 02:48 PM - wardrop (Tom Wardrop)

Enumerator includes Enumerable, as does Hash. Enumerator introduces a few new methods that revolve around the concept of a cursor, but otherwise everything else comes from Enumerable.

My whole point is that for anyone using #reject or #select to retrieve an Enumerator from a Hash (which really no one should be doing), there's a good chance their code will still work, as long as they're not using the extra cursor functionality exclusive to Enumerators.

Hash#select(*keys) is such an appropriate interface for obtaining a subset of a hash, as that's exactly what the #select and #reject methods are intended for. It would be silly in my opinion to introduce a new method.

The question is, should Hash#select without arguments return a copy of the original hash, or an empty hash? Thinking about it, I'd say Hash#select should return an empty hash if no arguments are given, though this completely breaks compatibility for anyone using Hash#select (without arguments) as a means of obtaining an enumerator. Hash#reject without an argument should definitely return a full copy of the original hash.

#14 - 11/15/2013 03:28 PM - alexeymuranov (Alexey Muranov)

wardrop (Tom Wardrop) wrote:

> I think an appropriate solution would be to amend Hash#select. It currently doesn't take any arguments, so could easily take a list of *args. You could use the optional block to further refine this selection if need be. The same for Hash#reject.
>
> One potential issue though is that Hash#select without any arguments currently returns an enumerator. This could be problematic when doing something like hash.select *args with an empty array. I think calling Hash#select without arguments should return a copy of the full original Hash, this keeps it somewhat compatible with the existing behaviour of returning an enumerator (given a Hash is an enumerator), while at the same time making it consistent with the new Hash#select(*keys) functionality.

If Hash#select without arguments returns the original hash or an enumerator, it will contradict the proposal to use #select with args to "slice" a hash: it would have to return the empty hash.

#15 - 11/15/2013 03:32 PM - alexeymuranov (Alexey Muranov)

However, i do not see why it has to be used as select(*args) and not select select(ary) or select(enum).

#16 - 11/17/2013 02:39 PM - wardrop (Tom Wardrop)

select(*args) just seemed like a more natural interface, though I suppose select(enum) provides more flexibility and solves any compatibility problems with the current behaviour of select. If an empty enumerable is given, an empty hash is returned. If no argument is given, then the current behaviour of returning an enumerator is respected. That'll work well.

In summary, I'm in favour of the select(enum) implementation, likewise for #reject.

#17 - 10/03/2014 08:40 PM - Ajedi32 (Andrew M)

My personal preference would be for Hash#select(*args) and Hash#reject(*args), but if we *really* must maintain backwards compatibility for Hash#select/reject with no args or block then I guess Hash#select(enum) and Hash#reject(enum) would fine too.

#18 - 06/12/2015 07:27 AM - ko1 (Koichi Sasada)

- Description updated

- Assignee set to matz (Yukihiro Matsumoto)

#19 - 06/12/2015 07:32 AM - ko1 (Koichi Sasada)

- Related to Feature #8499: Importing Hash#slice, Hash#slice!, Hash#except, and Hash#except! from ActiveSupport added

#20 - 06/12/2015 07:33 AM - matz (Yukihiro Matsumoto)

I prefer use of Hash#select, but in form of hash.select([:foo, :bar]), since it may consume too much stack region, besides that hash.select(*arg) could work differently when arg=[]. It would cause confusion.

Matz.

#21 - 06/12/2015 03:46 PM - pabloh (Pablo Herrero)

Since Hash#slice wouldn't really play well polymorphically with Array#slice, and it feels (to me at least) a bit odd to have Hash#select returning an enumerator if the parameter is an array while Enumerable#select would fail on that scenario, why we don't go with a new selector altogether like Hash#only (and maybe Hash#except)?.

#22 - 06/15/2015 09:46 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I needed (once more) Hash#except today and I always wonder why it doesn't exist in Ruby yet. I ended up using the one implemented in ActiveSupport even though I usually avoid AS as much as possible. Hash really needs quicker ways to filter keys in and out. Maybe hash.except([:foo, :bar]) could be introduced altogether with hash.select([:foo, :bar]).

#23 - 06/16/2015 01:26 AM - akr (Akira Tanaka)

Rodrigo Rosenfeld Rosas wrote:

> I needed (once more) Hash#except today and I always wonder why it doesn't exist in Ruby yet. I ended up using the one implemented in ActiveSupport even though I usually avoid AS as much as possible. Hash really needs quicker ways to filter keys in and out. Maybe hash.except([:foo, :bar]) could be introduced altogether with hash.select([:foo, :bar]).

hash.reject([:foo, :bar]) would be easier to accept because Hash#reject is already exist and matz prefer similar form: https://bugs.ruby-lang.org/issues/8499#note-18

#24 - 06/16/2015 03:52 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I actually prefer the signature used by Matz two years ago in that note 18 of issue #8499, which is similar to how AS support implemented Hash#except. But since Matz seems to have changed his mind in note 20 of this ticket, I think Hash#except should have the same syntax as Hash#select.

Now, this will be a problem for lots of people using Rails since AS is not an opt-out gem in that case and it will override Hash#except changing its meaning from the Ruby bundled one.

This is the problem we get when we delegate such features to external libraries and wait for them to become popular. Once they are popular and you consider the feature and decide the behavior in the gem is not exactly the desired one we have a problem... It would be so much nicer if we could have decided on this before it was implemented in AS...

In AS except is implemented as dup.except!:

https://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/hash/except.rb

```
def except!(*keys)
  keys.each { |key| delete(key) }
  self
end
```

It uses the signature used in Matz' Note 18 of issue #8499, from 2 years ago...

#25 - 06/16/2015 06:26 PM - pabloh (Pablo Herrero)

Rodrigo Rosenfeld Rosas wrote:

> I actually prefer the signature used by Matz two years ago in that note 18 of issue #8499, which is similar to how AS support implemented Hash#except. But since Matz seems to have changed his mind in note 20 of this ticket, I think Hash#except should have the same syntax as Hash#select.
>
> Now, this will be a problem for lots of people using Rails since AS is not an opt-out gem in that case and it will override Hash#except changing its meaning from the Ruby bundled one.
>
> This is the problem we get when we delegate such features to external libraries and wait for them to become popular. Once they are popular and you consider the feature and decide the behavior in the gem is not exactly the desired one we have a problem... It would be so much nicer if we could have decided on this before it was implemented in AS...
>
> In AS except is implemented as dup.except!:
>
> https://github.com/rails/rails/blob/master/activesupport/lib/active_support/core_ext/hash/except.rb
>
> ```
> def except!(*keys)
>   keys.each { |key| delete(key) }
>   self
> end
> ```
>
> It uses the signature used in Matz' Note 18 of issue #8499, from 2 years ago...

Merb used to have Hash#only(*args) and Hash#except(*args). If we add them now, AS will only have to alias '#slice' with '#only', and simply don't define '#except' if is already there. And maybe the stack issue could be solve at the compiler/vm level? (I'm asking because I'm not really that familiar with CRuby code to know that much).

#26 - 07/23/2016 02:57 PM - pabloh (Pablo Herrero)

Any chance something like this could make it into 2.4?.

Is really cumbersome to require ActiveSupport as a dependency just to use a handful of methods and these two are quite common.

BTW: I think Hash#only(*args) / Hash#except(*args) or Hash#with(*args) / Hash#without(*args) are both good options. Hash#slice wouldn't really behave very similar to Array#slice, so I don't think is a very good option but is at least what AS already does.

#27 - 10/18/2016 03:55 AM - wardrop (Tom Wardrop)

Indeed, I'm still hanging out for this. Seems like such a common thing I run into, and I'm always surprised this functionality isn't built in. Hash#select { |k,v| {...}.include?(k) } is very verbose.

#28 - 12/11/2017 09:59 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Open to Closed