Ruby master - Feature #5008

Equal rights for Hash (like Array, String, Integer, Float)

07/10/2011 05:57 AM - sunaku (Suraj Kurapati)

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

Description

=begin
Hello,

I am using ruby 1.9.2p180 (2011-02-18 revision 30909) [x86_64-linux].

Although Ruby has a rich set of primitive data types and structures,
the Hash seems neglected in the Ruby API in comparison to its peers:

- String: Object#to_s by API
- Integer: Kernel#Integer by API and Object#to_i by convention
- Float: Kernel#Float by API and Object#to_f by convention
- Array: Kernel#Array by API and Object#to_a by convention
- Hash: Kernel#Hash (issue #3131) and Object#to_hash by convention

In particular, the Hash seems neglected by the Ruby API because:

- Its convention method (#to_hash) is longer than one character (#to_h).
- It did not have a Kernel-level method until recently (see issue #3131).
- It has no methods for conversion from NilClass, unlike #to_s, a, i, f.

Please rectify this un-orthogonality and grant Hash equal rights by:

- Establish #to_h as the convention method for converting objects into Hash.
- Add Kernel#Hash method for converting objects into Hash strictly (see issue #3131).
- Define NilClass#to_h so that we can convert nil into an empty Hash.

Thanks for your consideration.
=end

Related issues:

Related to Ruby master - Feature #4151: Enumerable#categorize	**Rejected**		
Related to Ruby master - Feature #7241: Enumerable#to_h proposal	**Rejected**	**10/30/2012**	
Related to Ruby master - Feature #6276: to_h as explicit conversion to Hash	**Closed**	**04/10/2012**	
Is duplicate of Ruby master - Feature #4862: Struct#to_hash	**Rejected**	**06/10/2011**	
Is duplicate of Ruby master - Feature #1400: Please add a method to enumerate...	**Closed**	**04/23/2009**	
Precedes Ruby master - Feature #3131: add Kernel#Hash() method like Kernel#Ar...	**Closed**	**07/11/2011**	**07/11/2011**

Associated revisions

Revision 33399fe8 - 04/16/2012 03:15 AM - marcandre (Marc-Andre Lafortune)

- object.c: Add NilClass#to_h [Feature #6276] [ref #5008] [rubyspec:dc5ecddbd608]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@35340 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 35340 - 04/16/2012 03:15 AM - marcandre (Marc-Andre Lafortune)

- object.c: Add NilClass#to_h [Feature #6276] [ref #5008] [rubyspec:dc5ecddbd608]

Revision 35340 - 04/16/2012 03:15 AM - marcandre (Marc-Andre Lafortune)

- object.c: Add NilClass#to_h [Feature #6276] [ref #5008] [rubyspec:dc5ecddbd608]

Revision 35340 - 04/16/2012 03:15 AM - marcandre (Marc-Andre Lafortune)

- object.c: Add NilClass#to_h [Feature #6276] [ref #5008] [rubyspec:dc5ecddbd608]

Revision 35340 - 04/16/2012 03:15 AM - marcandre (Marc-Andre Lafortune)

- object.c: Add NilClass#to_h [Feature #6276] [ref #5008] [rubyspec:dc5ecddbd608]

Revision 35340 - 04/16/2012 03:15 AM - marcandre (Marc-Andre Lafortune)

- object.c: Add NilClass#to_h [Feature #6276] [ref #5008] [rubyspec:dc5ecddbd608]

Revision 35340 - 04/16/2012 03:15 AM - marcandre (Marc-Andre Lafortune)

- object.c: Add NilClass#to_h [Feature #6276] [ref #5008] [rubyspec:dc5ecddbd608]

History

#1 - 07/16/2011 12:23 AM - naruse (Yui NARUSE)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

First of all, Ruby has two way of the type conversion; implicit and explicit.
to_i, to_f, to_s, to_a and so on are explicit conversion.
to_int, to_str, to_ary and so on are implicit conversion.

> Establish #to_h as the convention method for converting objects into Hash.

If to_h is introduced, it should be a implicit conversion.
But what it is?

> Add Kernel#Hash method for converting objects into Hash strictly (see issue #3131).

Why don't you discuss in #3131?

> Define NilClass#to_h so that we can convert nil into an empty Hash.

You should show the use case: what is the benefit of the function.

#2 - 07/27/2011 02:52 AM - sunaku (Suraj Kurapati)

Yui NARUSE wrote:

> First of all, Ruby has two way of the type conversion; implicit
> and explicit. to_i, to_f, to_s, to_a and so on are explicit
> conversion. to_int, to_str, to_ary and so on are implicit
> conversion.

I see, then for Hash:

- to_h should be explicit conversion
- to_hash should be implicit conversion.

> > Add Kernel#Hash method for converting objects into Hash strictly
> > (see issue #3131).
>
> Why don't you discuss in #3131?

You're right. Sorry for bringing that up.

> Define NilClass#to_h so that we can convert nil into an empty
> Hash.

You should show the use case: what is the benefit of the function.

The benefit of NilClass#to_h is convenience. It is the same reason why NilClass#to_a and NilClass#to_s exist.

For example, we might process an array that (1) can be empty or (2) have a hash as the first element:

```
some_array.first.to_h.each_pair do |key, value|
# do some processing ...
end
```

Without NilClass#to_h, we need to do some extra work:

```
if hash = some_array.first
hash.each_pair do |key, value|
# do some processing ...
end
end
```

Furthermore, this extra work seems unfair to Hash because all of the other primitive data structures (strings, arrays) have NilClass#to_* conversion methods. :-(

Thanks for your consideration.

#3 - 07/27/2011 03:30 AM - steveklabnik (Steve Klabnik)

NilClass#to_h would be useful. Especially for Enumerable types, it's common convention to return an empty container of that type, so that you can chain things together.

The proliferation of nil is often considered a code smell, there's tons of cases where converting nils into values that still mean 'nothing' but have some amount of meaning is useful. See Avdi's talk at RailsConf about Confident Code, for example: http://avdi.org/talks/confident-code-railsconf-2011/

#4 - 09/26/2011 05:45 AM - sunaku (Suraj Kurapati)

Any chance of this getting into Ruby 1.9.3? Thanks.

#5 - 09/26/2011 10:10 AM - trans (Thomas Sawyer)

+1 I use #to_h often and always have to implement myself when needed. It's annoying.

#6 - 09/27/2011 10:04 AM - sunaku (Suraj Kurapati)

I have implemented these features in this repo:

https://github.com/sunaku/equal_rights_for_hash

And I have released the code as a nice RubyGem:

http://rubygems.org/gems/equal_rights_for_hash

This will serve as a workaround until these "equal rights" are granted to Hash in Ruby's core itself.

Cheers.

#7 - 09/27/2011 11:50 AM - trans (Thomas Sawyer)

See facets/to_hash.rb

#8 - 09/28/2011 03:47 AM - sunaku (Suraj Kurapati)

Very nice! I had to dig around for the source code, so here it is for reference:

https://github.com/rubyworks/facets/blob/master/lib/core/facets/to_hash.rb

My implementation only supports the array-to-hash conversions documented in Hash.[]:

http://rubydoc.info/stdlib/core/1.9.2/Hash.[]

If the Ruby developers want to implement facets/to_hash.rb in Ruby core, then I would be even happier. But for now, the basic equal rights for hash are my goal.

#9 - 09/28/2011 04:32 AM - sunaku (Suraj Kurapati)

I have simplified my implementation to reflect the MRI implementation of Hash.[]:

https://github.com/sunaku/equal_rights_for_hash/blob/master/lib/equal_rights_for_hash.rb#L11

In the case where the object being converted is an array: If all of its items are arrays, then it is passed directly to Hash.[]. Otherwise, it is passed to Hash.[] in splatted form.

#10 - 09/28/2011 04:49 AM - sunaku (Suraj Kurapati)

Sorry for the frequent updates. I have added support for Enumerable#to_h now.

https://github.com/sunaku/equal_rights_for_hash/commit/a146f66594caffd6c1b68d864d729a870e516975

The line count is still the same and the implementation remains simple.

#11 - 09/28/2011 05:43 AM - sunaku (Suraj Kurapati)

Final update (hopefully): I made Kernel#Hash() raise ArgumentError just
like the other Kernel-level methods: Kernel#Integer() and Kernel#Float().

https://github.com/sunaku/equal_rights_for_hash/blob/ab10c5232452f54aa3c88d7520e9169327f92824/lib/equal_rights_for_hash.rb#L5

I have released all these changes as equal_rights_for_hash 0.0.4 gem.

Thanks for your consideration.

#12 - 10/05/2011 02:23 AM - sunaku (Suraj Kurapati)

=begin

Here is a comparison of core data structure API for your reference:

| | Kernel | Implicit | Explicit | NilClass |
Class	method	conversion	conversion	conversion
String	String()	to_str	to_s	nil.to_s
Integer	Integer()	to_int	to_i	nil.to_i
Float	Float()	MISSING	to_f	nil.to_f
Array	Array()	to_ary	to_a	nil.to_a
Hash	MISSING	to_hash	MISSING	MISSING

=end

#13 - 12/29/2011 02:22 AM - sunaku (Suraj Kurapati)

Any update on the status of this request being accepted into Ruby trunk? Thanks.

#14 - 03/16/2012 02:58 AM - sunaku (Suraj Kurapati)

=begin

Request #3131 was fulfilled, so here is the updated feature matrix:

| | Kernel | Implicit | Explicit | NilClass |
Class	method	conversion	conversion	conversion
String	String()	to_str	to_s	nil.to_s
Integer	Integer()	to_int	to_i	nil.to_i
Float	Float()	MISSING	to_f	nil.to_f
Array	Array()	to_ary	to_a	nil.to_a
Hash	Hash()	to_hash	MISSING	MISSING

=end

#15 - 03/16/2012 12:18 PM - marcandre (Marc-Andre Lafortune)

I approve to_h as explicit conversion to a Hash.

It should be created and implemented for NilClass, Hash, Enumerable, Struct, OpenStruct.

It would also pave the way for more generic hash splat if the proposal is accepted (see [ruby-core:40675])

The implementation proposed by Suraj Kurapati can not be accepted in its current form, though. In particular, there should be no Object/Kernel#to_h (like there is no Object#to_a/to_i), and Hash#to_h should return a new copy for subclasses (like to_a, to_s).

I realize that Matz doesn't like Enumerable#to_h because there is no perfect "natural" mapping. I feel like any choice here is better than no choice. I think it would be fine if Enumerable#to_h was equivalent to the key-value pair form of Hash[enum.to_a], but I'll agree with any other sane definition, as long as it's simple.

For more involved cases, I'm still hoping for Enumerable#associate/categorize.

Thanks.

Marc-André

#16 - 03/16/2012 01:23 PM - matz (Yukihiro Matsumoto)

Hi,

In message "Re: [ruby-core:43310] [ruby-trunk - Feature #5008] Equal rights for Hash (like Array, String, Integer, Float)"
on Fri, 16 Mar 2012 02:58:04 +0900, Suraj Kurapati sunaku@gmail.com writes:

|Request #3131 was fulfilled, so here is the updated feature matrix:
|
		Kernel	Implicit	Explicit	NilClass
	Class	method	conversion	conversion	conversion
	---	---	---	---	---
	String	String()	to_str	to_s	nil.to_s
	Integer	Integer()	to_int	to_i	nil.to_i
	Float	Float()	MISSING	to_f	nil.to_f
	Array	Array()	to_ary	to_a	nil.to_a
	Hash	Hash()	to_hash	MISSING	MISSING

This table means nothing. Are you going to expand the table for every class Ruby would provide, e.g. Complex, Rational, Range, etc?

Repeating myself, unlike other classes in the table, Hash does not have "natural" conversion from set of values, so that I don't think it's worth provide to_h method.

In [ruby-core:43321], Marc-Andre claims "any choice here is better than no choice". But I disagree. For "any choice", we already have Hash[value].

```
                          matz.
```

#17 - 03/16/2012 02:19 PM - trans (Thomas Sawyer)

"Repeating myself, unlike other classes in the table, Hash does not have "natural" conversion from set of values, so that I don't think it's worth provide to_h method."

What does that even mean?

From a practical vantage, I use #to_h all the time and constantly have to define it in my projects. I just did it a few minutes ago, in fact. The need stems from polymorphic behaviour on data mappings.

```
def some_method(data)
  data = data.to_h
  # ... now I have a known data object to work with ...
end
```

Anything that responds to #to_h can be passed. Without that we would have to account for every possible type, which is very limiting and basically not practical.

While #to_hash could be used, as with the other conversion methods, it conveys the object is a Hash in it's very essence --not simply an object that can fashion a hash from itself regardless of how. An object that responds to #to_hash, otoh, would almost certainly respond to #[], #[]= and #each and probably #key?, #keys and #values, too. Although obviously there is no necessary set of public methods it must provide. But is a much stronger indication of the object's nature.

Another use case is simple serialization. #to_h is hand-dandy for converting representation of objects to JSON. Again, something like Contact#to_h is perfectly useful and sensible, where as Contact#to_hash just doesn't jive.

#18 - 03/16/2012 02:23 PM - sunaku (Suraj Kurapati)

On Friday, 16 Mar 2012 at 1:13 PM, Yukihiro Matsumoto wrote:

> Hi,
>
> In message "Re: [ruby-core:43310] [ruby-trunk - Feature #5008] Equal rights for Hash (like Array, String, Integer, Float)" on Fri, 16 Mar

> 2012 02:58:04 +0900, Suraj Kurapati sunaku@gmail.com writes:
>
> |Request #3131 was fulfilled, so here is the updated feature matrix:
> |
> | | | Kernel | Implicit | Explicit | NilClass |
> | | Class | method | conversion | conversion | conversion |
> | | --- | --- | --- | --- | --- |
> | | String | String() | to_str | to_s | nil.to_s |
> | | Integer | Integer() | to_int | to_i | nil.to_i |
> | | Float | Float() | MISSING | to_f | nil.to_f |
> | | Array | Array() | to_ary | to_a | nil.to_a |
> | | Hash | Hash() | to_hash | MISSING | MISSING |
>
> This table means nothing. Are you going to expand the table for every class Ruby would provide, e.g. Complex, Rational, Range, etc?

No, this proposal is just for Hash because it is one of the two most commonly used data structures in Ruby/Perl/JSON/YAML: Array and Hash.

Amdahl's Law says "make the common case fast"; in this case, since Hash is so frequently used, we would all benefit by making it s/fast/easy/.

That is why I wrote that feature comparison matrix: to highlight the second-class treatment of Hash (in comparison to Array) in Ruby's API.

> Repeating myself, unlike other classes in the table, Hash does not have "natural" conversion from set of values, so that I don't think it's worth provide to_h method.
>
> In [ruby-core:43321], Marc-Andre claims "any choice here is better than no choice". But I disagree. For "any choice", we already have Hash[value].

Fair enough. I agree with Marc-Andre, but you're the boss. I will just have to live a gem that provides #to_h. Thanks for your consideration.

#19 - 03/17/2012 07:13 AM - marcandre (Marc-Andre Lafortune)

Hi,

Yukihiro Matsumoto wrote:

> Repeating myself, unlike other classes in the table, Hash does not have "natural" conversion from set of values, so that I don't think it's worth provide to_h method.

Thanks for taking the time to reply.

Even if you will not accept Enumerable#to_h, how about to_h for Hash, Struct, OpenStruct and NilClass, which do have a natural conversion to a hash?

Thanks

Marc-André

#20 - 03/17/2012 10:08 AM - trans (Thomas Sawyer)

> "Repeating myself, unlike other classes in the table, Hash does not have "natural" conversion from set of values, so that I don't think it's worth provide to_h method."
>
> What does that even mean?

I see, you were referring to Enumerable#to_h and meant there is no "single definitive" method to conversion. True, and to that end, I, with the help of other developers, worked through the general permutations of these:

https://github.com/rubyworks/facets/blob/master/lib/core/facets/to_hash.rb

So leave Enumerable#to_h out if you don't like it, but #to_h still has a clear definition for a number of other classes, does it not?

#21 - 03/17/2012 10:23 AM - matz (Yukihiro Matsumoto)

Hi,

In message "Re: [ruby-core:43358] [ruby-trunk - Feature #5008] Equal rights for Hash (like Array, String, Integer, Float)"
on Sat, 17 Mar 2012 07:13:17 +0900, Marc-Andre Lafortune ruby-core@marc-andre.ca writes:

|Even if you will not accept Enumerable#to_h, how about to_h for Hash, Struct, OpenStruct and NilClass, which do have a natural conversion to a hash?

It seems a better idea than adding Enumerable#to_h, except that I am
not sure nil.to_h is a good one.

```
                          matz.
```

#22 - 03/29/2012 08:36 AM - sunaku (Suraj Kurapati)

In my mind, nil.to_h should exist for the same reason that
nil.to_a, nil.to_s, nil.to_i, and nil.to_f exist: convenience.

It allows me to write:

some_complicated_method.to_h.each { ... }

Instead of being forced to write:

if some_hash = some_complicated_method
some_hash.to_hash.each { ... }
end

Note that I am only forced to write the above in the specific
case of hash and not for the other abstract data types because,
unlike hash, they all have nil.to_*() conversion methods.

Thanks for your consideration.

#23 - 03/29/2012 09:25 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Or for convenience we could allow nil to respond to each, like in Groovy:

null.each {} // or eachWithIndex - doesn't throw an exception

#24 - 03/29/2012 10:29 PM - aprescott (Adam Prescott)

On Thu, Mar 29, 2012 at 13:25, rosenfeld (Rodrigo Rosenfeld Rosas) <
rr.rosas@gmail.com> wrote:

> Or for convenience we could allow nil to respond to each, like in Groovy:
>
> null.each {} // or eachWithIndex - doesn't throw an exception

Why go down the road of adding this to nil instead of just relying on ||?

foo = some_method || {}
foo.each { ... }

If some_method is controlled, it can be made to return foo || {} if it
really does make sense for it to always return something that client code
can always use as a hash.

#25 - 03/29/2012 10:53 PM - regularfry (Alex Young)

On 29/03/12 14:28, Adam Prescott wrote:

> On Thu, Mar 29, 2012 at 13:25, rosenfeld (Rodrigo Rosenfeld Rosas)
> > wrote:
>
> ```
> Or for convenience we could allow nil to respond to each, like in
> Groovy:
>
> null.each {} // or eachWithIndex - doesn't throw an exception
> ```
>
> Why go down the road of adding this to nil instead of just relying on ||?
>
> foo = some_method || {}

> foo.each { ... }
>
> If some_method is controlled, it can be made to return foo || {} if it really does make sense for it to always return something that client code can always use as a hash.

Would you argue that this is wrong and should be removed?

1.9.3p125 :001 > nil.to_a
=> []

If not, why not? The same argument you've just made about hashes applies here.

--
Alex

#26 - 03/29/2012 11:23 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Em 29-03-2012 10:28, Adam Prescott escreveu:

> On Thu, Mar 29, 2012 at 13:25, rosenfeld (Rodrigo Rosenfeld Rosas)
> > wrote:
>
> ```
> Or for convenience we could allow nil to respond to each, like in
> Groovy:
>
> null.each {} // or eachWithIndex - doesn't throw an exception
> ```
>
> Why go down the road of adding this to nil instead of just relying on ||?
>
> foo = some_method || {}
> foo.each { ... }

What if some_method returned false instead of nil with this pattern? It would must probably be some kind of bug hard to track... Unfortunately Ruby doesn't have a similar operator that will only operate on nil objects.

I'm not saying that we should copy Groovy behavior as I still don't have a strong opinion on this subject. But Groovy has added "each" and "collect" to Object and not only to NullObject.
This way, it allows 10.collect{it} == [10] and null.collect{it} == [].
But on the other hand, 'abc'.collect{it} == ['a', 'b', 'c'] and I don't like this behavior.

While I find that adding each and collect to NilClass might be valid, I wouldn't like them to be added to Object.

In fact I would prefer to be able to use each and collect with nil objects rather than adding a "to_h" to it if we're talking about convenience.

Rodrigo.

#27 - 03/29/2012 11:53 PM - aprescott (Adam Prescott)

On Thu, Mar 29, 2012 at 14:39, Alex Young alex@blackkettle.org wrote:

> Would you argue that this is wrong and should be removed?
>
> 1.9.3p125 :001 > nil.to_a
> => []
>
> If not, why not?

Perhaps I wasn't clear (including to myself).

I think nil.to_h is probably fine, for the same reason nil.to_a is fine, and it would nice to have it. But, I think adding #each to NilClass is the wrong way to go to get around being able to call #each on the return value of some method call. If you really need to_h behaviour, I think you can go for || before sticking #each on NilClass.

#28 - 04/10/2012 06:48 PM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

#to_hash protocol expects the object to be hash-compatible. Struct is not the case.

Matz.

#29 - 04/10/2012 10:07 PM - marcandre (Marc-Andre Lafortune)

- Status changed from Rejected to Open

Hi,

matz (Yukihiro Matsumoto) wrote:

> #to_hash protocol expects the object to be hash-compatible. Struct is not the case.

I'm reopening this issue, as the request is not for #to_hash but for #to_h.

I believe you are positive for Hash#to_h, Struct#to_h and OpenStruct#to_h. [ruby-core:43363]

You are hesitant about NilClass#to_h, and we were wondering why. I think the translation from nil to {} would be consistent with the transitions to the other basic types, like nil.to_s, nil.to_i, nil.to_f, nil.to_c, nil.to_r. It's the only one missing!

#30 - 04/10/2012 10:19 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

Submit new issue for new proposal, e.g. adding hash conversion method to Struct etc.

The reason I hesitate to add to_h to nil is because I am not fully satisfied with nil.to_a etc.
They are sometimes useful, but often hides bugs and hiders crash-early principle.

Matz.

#31 - 02/09/2013 03:30 PM - sunaku (Suraj Kurapati)

Hi Matz,

matz (Yukihiro Matsumoto) wrote:

> The reason I hesitate to add to_h to nil is because I am not fully satisfied with nil.to_a etc.
> They are sometimes useful, but often hides bugs and hiders crash-early principle.

Thank you very much for accepting nil.to_h in issue #6276. ^_^

I just read about this feature in Ruby 2.0.0 release notes.

Cheers!