Ruby - Bug #14909

Method call with object that has to_hash method crashes (method with splat and keyword arguments)

07/12/2018 07:37 AM - johannes_luedke (Johannes Lüdke)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.4.1p111 (2017-03-22 revision 58053) [x86_64-darwin17]	**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

In a method with a splat method argument followed by a keyword argument, it leads to an ArgumentError when calling the method with an object that reacts to to_hash

```
def my_func(*objects, error_code: 400)
  objects.inspect
end

class Test
  def to_hash
    # an example hash
    { to_hash_key: "to_hash" }
  end
end

my_func(Test.new)
```

Observed result: an exception is raised: in my_func: unknown keyword: to_hash_key (ArgumentError)
Expected result: [#<Test:0x007fc8c9825318>] is returned by the my_func call

It should behave the same when calling with objects that have a to_hash method and objects that don't, shouldn't it?

Related issues:

Related to Ruby - Feature #14930: sample/trick2018	**Closed**

History

#1 - 07/12/2018 11:21 AM - mame (Yusuke Endoh)

- Status changed from Open to Feedback

In the plan of Ruby 3 (#14183), keyword arguments will be separated from other kinds of arguments. If this plan is implemented successfully, you can use my_func(Test.new) and my_func(**Test.new) for each purpose. If you call my_func(Test.new), the argument will be passed as a part of the rest parameter objects. If you call my_func(**Test.new), the argument will be handled as a keyword parameter.

So, I'd like to propose keeping the current behavior as is, because changing the semantics will bring extra complexity. Instead, just wait for Ruby 3.

#2 - 07/12/2018 12:29 PM - Eregon (Benoit Daloze)

At least, it behaves the same if passing a keyword arguments directly:

```
def my_func(*objects, error_code: 400)
  objects.inspect
end
my_func(to_hash_key: "to_hash") # => unknown keyword: to_hash_key (ArgumentError)
```

So this has nothing to do with the to_hash conversion.

One way to workaround this is:

```
def my_func(*objects, error_code: 400, **kwargs)
  kwargs
end
p my_func(to_hash_key: "to_hash") # => {:to_hash_key=>"to_hash"}
```

So adding a keyrest argument (**kwargs), because there is already a keyword argument which means keywords have to fit in the declared keyword args, unless there is a keyrest arg.

#3 - 07/13/2018 04:52 AM - funny_falcon (Yura Sokolov)

Why your object has to_hash method?

Ruby uses long named methods for implicit conversion: to_str - if your object should act as a string, to_int - if your object should act as an integer, to_ary - if your object should act as an array. Looks like same for to_hash.

For explicit conversion short names are used: to_s, to_i, to_a, to_h.

#4 - 07/13/2018 03:18 PM - johannes_luedke (Johannes Lüdke)

https://bugs.ruby-lang.org/issues/14909#note-2 doesn't resolve the issue for me

> Why your object has to_hash method?

the objects in question are instances of Dry::Validation::Result (dry-validation gem)

> So, I'd like to propose keeping the current behavior as is, because changing the semantics will bring extra complexity. Instead, just wait for Ruby 3.

Could this maybe be highlighted in the docs -- to be careful when passing objects that respond to_hash when there are keyword arguments?

In order to make it work both ways, my_func(obj1, obj2, error_code: 422) as well as my_func(obj1, obj2) with a default value for error_code, I ended up doing this workaround:

```
def my_func(*args)
  opts, objects = args.partition { |el| el.is_a? Hash }
  error_code = opts&.first&.fetch(:error_code, nil) || 400
```

It would be cool if ruby would support that out of the box though.

#5 - 07/22/2018 06:35 AM - znz (Kazuhiro NISHIYAMA)

- Related to Feature #14930: sample/trick2018 added

#6 - 09/02/2019 04:48 AM - jeremyevans0 (Jeremy Evans)

- Status changed from Feedback to Closed

The changes in #14183 solve this issue. You will now get warnings:

```
my_func(Test.new)
# (irb):101: warning: The last argument is used as the keyword parameter
# (irb):92: warning: for `my_func' defined here
# ArgumentError (unknown keyword: :to_hash_key)
```

In Ruby 3, this will be passed as a positional argument. To get the Ruby 3 behavior with the master branch:

```
my_func(Test.new, **(;{}))
```

#7 - 02/06/2020 01:25 PM - AlexWayfer (Alexander Popov)

There are problems with Sequel Models, which have #to_hash method (implicit conversion), and Memery gem, which defines methods with *args, **kwargs, &block (**kwargs was added after warning from Ruby 2.7).

So, Sequel Models as arguments for memoized methods becomes Hashes and it breaks everything.

The **(;{}) approach is:

- unclear for me (what it does in Ruby?);
- additional changes at every call;
- offenses RuboCop (should be resolved there, yes).

I'll try to make dynamic methods definition in Memery with exactly expected parameters instead of *args, **kwargs, &block, but it's harder.

And, probably, impossible at all, like define_method(:bar) do |*instance_method(:foo).parameters|. Only eval, I guess.

#8 - 02/06/2020 02:22 PM - jeremyevans0 (Jeremy Evans)

AlexWayfer (Alexander Popov) wrote in #note-7:

There are problems with Sequel Models, which have #to_hash method (implicit conversion), and Memery gem, which defines methods with *args, **kwargs, &block (**kwargs was added after warning from Ruby 2.7).

Memery may want to switch to using ruby2_keywords instead.

So, Sequel Models as arguments for memoized methods becomes Hashes and it breaks everything.

The **(;{}) approach is:

- unclear for me (what it does in Ruby?);

It passes empty keywords, so that the final positional object is not treated as a hash.

- additional changes at every call;

Should not be necessary with the ruby2_keywords approach.

- offenses RuboCop (should be resolved there, yes).

I'll try to make dynamic methods definition in Memery with exactly expected parameters instead of *args, **kwargs, &block, but it's harder.

And, probably, impossible at all, like define_method(:bar) do |*instance_method(:foo).parameters|. Only eval, I guess.

*args, &block as arguments and ruby2_keywords :bar after defining the method should work, and that is the approach I would recommend.