Propose to make Struct subclass constructors which accept keyword arguments. Not sure, if it's reasonable to allow .new accept kwargs, so may be should use different method named like .create:

```ruby
Point = Struct.new(:x, :y, :color)
pt_1 = Point.create(x: 1, y: 2) # => Point<x: 1, y: 2, color: nil>
pt_2 = Point.create!(x: 1, y: 2) # => ArgumentError, color not specified.
```

It will greatly simplify work with big structures, especially in cases when struct layout changes and for cases when structure can have lots of non-significant values. It also allows simpler ways to use implement default values for struct members.

Associated revisions

Revision 02015974 - 12/12/2017 08:12 AM - k0kubun (Takashi Kokubun)
struct.c: add keyword_init option to Struct.new
to initialize struct with keyword arguments.

[Feature #11925] [close GH-1771]
git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@61137 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 61137 - 12/12/2017 08:12 AM - k0kubun (Takashi Kokubun)
struct.c: add keyword_init option to Struct.new
to initialize struct with keyword arguments.

[Feature #11925] [close GH-1771]

Revision 61137 - 12/12/2017 08:12 AM - k0kubun (Takashi Kokubun)
struct.c: add keyword_init option to Struct.new
to initialize struct with keyword arguments.

[Feature #11925] [close GH-1771]

Revision 61137 - 12/12/2017 08:12 AM - k0kubun (Takashi Kokubun)
struct.c: add keyword_init option to Struct.new
to initialize struct with keyword arguments.

[Feature #11925] [close GH-1771]

History

#1 - 12/30/2015 07:44 AM - ksss (Yuki Kurihara)
Hi.
I have thought the same thing that want to use kwargs in Struct class.
But I can't come up with a good API.

Finally, I made a gem.  https://github.com/ksss/type_struct

#2 - 05/17/2016 06:37 AM - naruse (Yui NARUSE)
- Assignee deleted (ruby-core)

#3 - 05/18/2016 12:23 AM - shyouhei (Shyouhei Urabe)
We looked at this issue in yesterday's developer meeting. Nobody there was against the functionality -- but the name. create! doesn't sound appropriate at all. create also not that obvious for non-English speakers like us that it expects keywords.

#4 - 10/25/2016 06:58 AM - herwinw (Herwin Quarantainenet)
What about new_from_kwargs(**kwargs) ? It's a bit long, but it describes the functionality exactly.

#5 - 10/25/2016 06:37 PM - herwin (Herwin W)
https://github.com/ruby/ruby/pull/1468
A proposal for an implementation.

```
irb(main):001> MyClass = Struct.new(a, b, c)
 => MyClass
irb(main):002> MyClass.new_from_kwargs(a:1, c: 3)
 => #<struct MyClass a=1, b=nil, c=3>
irb(main):003> MyClass.new_from_kwargs(1, 2, 3, b: 3)
 => #<struct MyClass a=1, b=3, c=3>
irb(main):004> MyClass.new_from_kwargs(d: 4)
NameError: no member 'd' in struct
  from (irb):4:in 'new_from_kwargs'
from (irb):4
from ./irb:11:in `<main>'

irb(main):005> MyClass.new_from_kwargs(1, 2, 4, 5, b: 3)
IndexError: offset 3 too large for struct(size:3)
  from (irb):5:in 'new_from_kwargs'
from (irb):5
from ./irb:11:in `<main>'
irb(main):006>
```

#6 - 10/25/2016 06:54 PM - nobu (Nobuyoshi Nakada)
Herwin W wrote:
```
irb(main):003:0> MyClass.new_from_kwargs(1, 2, 3, b: 3)
 => #<struct MyClass a=1, b=3, c=3>
```

Why does new_from_kwargs accept other than keyword arguments?

#7 - 10/25/2016 07:05 PM - herwin (Herwin W)
To be prepared for "the great unification of constructors" of course.

It looked like a pretty logical step to support while I was coding this. It's also pretty easy to remove again

#8 - 12/21/2016 06:52 AM - shyouhei (Shyouhei Urabe)
- Has duplicate Feature #9209: Struct instances creatable with named args added

#9 - 01/20/2017 04:41 AM - ko1 (Koichi Sasada)
Another idea is introducing another method to define own struct, such as T = Struct.define(:a, :b); T.new(a: 1, b: 2) and so on. (just idea) Moreover we can extend Struct with some properties, like: Struct.define(:a, :b: :read_only).

These ideas are provided by another person.

#10 - 03/03/2017 04:58 AM - nobu (Nobuyoshi Nakada)
- Has duplicate Feature #13272: Keyword argument to instantiate a subclass of Struct added

#11 - 12/06/2017 01:57 PM - k0kubun (Takashi Kokubun)
Similar to one commented by ko1, how about this interface?
T = Struct.new(:a, :b, keyword_argument: true)
T.new(a: 1, b: 2)

As keyword_argument is long, another option is:

Struct.new(:a, :b, keyword_args: true)

#12 - 12/10/2017 06:47 AM - k0kubun (Takashi Kokubun)
In case that my suggestion of the name is accepted, I wrote a patch for Struct.new(:a, :b, keyword_args: true).
https://github.com/ruby/ruby/pull/1771

#13 - 12/12/2017 07:42 AM - knu (Akinori MUSHA)
What if Struct.new([[:a, :b]]) created a class with the desired constructor?

#14 - 12/12/2017 07:54 AM - matz (Yukihiro Matsumoto)
I vote for the keyword argument (e.g. keyword_init:) to Struct#new.
Matz.

#15 - 12/12/2017 08:06 AM - herwin (Herwin W)

knu (Akinori MUSHA) wrote:

What if Struct.new([[:a, :b]]) created a class with the desired constructor?

If you'd compare the two possible constructors:

Struct.new(:a, :b)
Struct.new([[:a, :b]])

There is nothing in the second one that would indicate the second creates a keyword constructor. If I hadn't read this discussion, I would just expect them to behave the same.

#16 - 12/12/2017 08:12 AM - k0kubun (Takashi Kokubun)
- Status changed from Open to Closed

Applied in changeset trunk|61137.

struct.c: add keyword_init option to Struct.new
to initialize struct with keyword arguments.

[Feature #11925] [close GH-1771]

#17 - 12/12/2017 08:16 AM - k0kubun (Takashi Kokubun)

As Matz approved, I committed only keyword_init option which is equivalent to "Point.create(x: 1, y: 2)" in original suggestion. If you still want "Point.create!" version which raises ArgumentError (keyword_init initializes unspecified fields with nil), please file another ticket.

#18 - 09/05/2018 10:21 AM - nobu (Nobuyoshi Nakada)
- Related to Feature #15076: Struct to raise error when keyword arguments used but not enabled added

#19 - 10/11/2018 01:10 PM - k0kubun (Takashi Kokubun)
- Related to Feature #15222: Add a way to distinguish between Struct classes with and without keyword initializer added