Ruby master - Feature #12157

Is the option hash necessary for future Rubys?

03/08/2016 05:16 AM - justcolin (Colin Fulton)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

Option hashes were great before Ruby had named arguments, but with the addition of named arguments and the double splat operator are they still needed?

One can convert named arguments into an option hash by using the double splat operator in the parameter list, so option hashes are redundant. More importantly, the existence of both option hashes and named arguments is creating odd or ugly edge cases. See #12022, #10708, and #11860. Legacy software can mostly be updated using a double splatted parameter, so it shouldn't be a hard shift to make.

What do you all think?

Apologies if this discussion has happened before. I searched for a bit but couldn't turn anything up.

History

#1 - 03/09/2016 09:25 PM - justcolin (Colin Fulton)

- Tracker changed from Bug to Feature

Sorry, I accidentally marked this as a bug. Moving it to features.

#2 - 03/09/2016 09:49 PM - jeremyevans0 (Jeremy Evans)

An option hash is just an optional argument with a default value of the empty hash. You can't really remove options hashes unless you remove optional arguments or disallow using a hash as a default value for an optional argument, neither of which seems like a good idea. Can you clarify what exactly you are proposing?

#3 - 03/09/2016 11:56 PM - justcolin (Colin Fulton)

Examples

Currently Ruby has syntactic sugar to make passing a Hash as a final argument prettier. Consider the following:

```
def foo(arg1, arg2, option_hash)
  option_hash
end
```

If someone doesn't know about the options hash syntactic sugar they can write:

```
foo("hello", "world", { color: "blue", pugs: "awesome" })
  # => { :color => "blue", :pugs => "awesome" }

foo("hello", "world", { "color" => "blue", "pugs" => "awesome" })
  # => { "color" => "blue", "pugs" => "awesome" }
```

However, the options hash syntactic sugar allows one to write:

```
foo("hello", "world", color: "blue", pugs: "awesome")
  # => { :color => "blue", :pugs => "awesome" }

foo("hello", "world", "color" => "blue", "pugs" => "awesome")
  # => { "color => "blue", "pugs" => "awesome" }
```

The curly braces get added to the arguments list implicitly as long as the hash is the last argument. This makes it even more possible to write a lot of really great looking DSLs in Ruby.

Ruby 2.0 and 2.1 added named parameters and the double splat syntax, which can look identical to an options hash when calling, but named arguments are more explicit in the method definition:

```
def bar(arg1, arg2, **option_hash)
  option_hash
end
```

Here is a call to #bar:

```
bar("hello", "world", color: "blue", pugs: "awesome")
  # => { :color => "blue", :pugs => "awesome" }
```

Unlike named arguments, options hashs allow you to have keys which are not symbols, but I haven't seen many places where that is needed. Named arguments have the big benefits that they give you the ability have required keys, and they make methods easier to read. You don't have to dig through the method's source to see what keys are expected, you only have to look at the parameters list. With options hashes you have to though the code to see what keys are required.

Discussion

The issue is that having both the options hash syntactic sugar AND named arguments at the same time causes some weird corner cases (see: #12022, #10708, and #11860).

The question I am putting forward is this: what do we value as more important?

Is the special syntax for options hashes so important that we we should allow for those edge cases to exist? This may be the case. If so, then what do we want to do with the edge cases? Currently the edge cases almost always assume that you meant to write an options hash, **but**#12022 shows that that makes things like decorators really ugly to write.

Personally I think that we should keep options hashes but we should fix those edge cases to assume that the author means to use named arguments instead of an options hash. Named hashes are often a much cleaner and more powerful solution, so we should favor people using them.

#4 - 03/10/2016 05:58 AM - sawa (Tsuyoshi Sawada)

Note that optional hash is not just for arguments in method call. It is also used for arrays, for example:

```
["a", "b", "c" => 1, d: 2]
```

Unlike in method calls, keyword arguments do not make sense in an array. Therefore, removing the optionality of braces for hashes in the syntax and reinterpreting them as keyword arguments is not realistic.

If you want some way to explicitly distinguish final hash arguments from keyword arguments, it is the keyword arguments that should have a new special syntax.

#5 - 03/10/2016 06:59 AM - justcolin (Colin Fulton)

Edited:
Tsuyoshi Sawada: how do you feel about those edge cases dealing with the double splat? Do you know of any real world code that uses the option hash in arrays? I have yet to see that in the wild.

A special syntax to distinguish keyword arguments from options hashes could be an interesting solution, but I am not sure how that could be done well.

#6 - 03/23/2016 09:42 PM - shevegen (Robert A. Heiler)

You forget that some people - like me - may use a hash because it is a lot simpler and more convenient too.

I don't have to wonder about any positional arguments ever and I don't want to have to think.

With a hash I don't have to think.

Removing the optionality of braces is not a good idea, because it simply is not necessary. Colin Fulton reasons that they can be removed now because keyword args are possible but that is not a "either or", ruby has an "and - and" philosophy.

I do not feel that hashes used for options are redundant in any way, they are synergistic and partially overlapping in their use case, just as many other parts of ruby also are.

I also do not see how your use of double splat has anything to do with option hashes at all since you do not need ** for a hash either.

> Unlike named arguments, options hashs allow you to have keys which are not symbols, but I haven't seen many places where that is needed.

Then perhaps you have not heard of HashWithIndifferentAccess or older discussions about whether to use 'key' or :key for any given hash. User input tends to be given in via a String input so you'd always have to convert into a symbol before storing in a Hash.

Option hashes were great before Ruby had named arguments

Option hashes are still great.

I am very much against removing hashes for options as per this proposal just because of some arbitrarily perceived keyword arguments being "superior" and any alternative means being "inferior".

#7 - 05/17/2016 07:23 AM - matz (Yukihiro Matsumoto)

Agreed. But we have to design migration path. That's a hard problem, though.

Matz.

#8 - 05/18/2016 10:07 AM - dsferreira (Daniel Ferreira)

It is not clear to me what will be the future behaviour.
Can we update the description to clearly state what we will have once the feature is implemented?

#9 - 09/02/2019 04:15 AM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed

With the acceptance of #14183, this can be closed. The behavior in Ruby 3 will be that **args in a method call will set keyword arguments if the method accepts keyword arguments, and will be a positional hash argument if the method does not (unless args is empty, in which case no argument will be added).