

Ruby master - Feature #16511

Staged warnings and better compatibility for keyword arguments in 2.7.1

01/16/2020 04:30 AM - Dan0042 (Daniel DeLorme)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	
Description	
<p>As an alternative to #16463 and #16494 I'd like to propose this approach, which I believe allows a much more flexible path for migration of keyword arguments.</p> <p>The idea is to indicate for every Hash object if it's intended to represent a <u>keyword</u> hash or a <u>data</u> hash. This extra information is then used to generate more granular warnings depending on a user's compatibility needs.</p> <p>The "keywordness" of a hash would be indicated by a flag on the Hash object; this is already implemented in 2.7 and is the approach favored by Matz. Let's call this flagged hash a "KwHash", and a non-flagged hash is just a "Hash". Note: this could also be implemented via a subclass of Hash (I personally favor this object-oriented approach) which was the original idea in this proposal.</p> <p>I'll try to describe the idea in detail by breaking it down into figurative steps. (Skip to "Putting it all together" for the TL;DR version.) Imagine starting with ruby 2.6 and then:</p>	
Step 1	
<p>When a double-splat or a brace-less hash is used, instead of a Hash it creates a KwHash.</p> <pre>def foo(x) x end foo(k:1).class #=> KwHash foo(**hash).class #=> KwHash [k:1].last.class #=> KwHash [**hash].last.class #=> KwHash {**hash}.class #=> Hash</pre>	
<p>At this point we haven't introduced any real change. Everything that worked before is still working the same way. (With a minor exception if using the subclass approach: unusual code like <code>kw.class == Hash</code> would now return false.)</p>	
Step 2	
<p>When there is ambiguity due to optional vs keyword argument, we rely on the last argument being Hash or KwHash to disambiguate.</p> <pre>def foo(x=nil, **kw) [x, kw] end foo({k:1}) #=> [{k:1}, {}] foo(k:1) #=> [nil, {k:1}]</pre>	
<p>This is the <u>minimum</u> amount of incompatibility required to solve ALL bugs previously reported with keyword arguments. (#8040, #8316, #9898, #10856, #11236, #11967, #12104, #12717, #12821, #13336, #13647, #14130, etc.)</p>	
<p>The warnings for this would be about an impending <u>change of behavior</u> in the <u>next ruby version</u>, where <code>foo({k:1})</code> is no longer interpreted as keyword argument.</p>	
Step 3	
<p>Introduce additional incompatibility to improve clarity of design. Here we deprecate the automatic conversion of Hash to keyword argument; only KwHash is accepted. With a deprecation/warning phase, of course. The "automatic" promotion of a KwHash to a keyword argument follows the same rules as a Hash in 2.6; since the KwHash is conceptually intended to represent keyword arguments, this conversion makes sense in a way that a normal data Hash doesn't. We've taken the "last positional hash" concept and split it into "conceptually a hash" and "conceptually keyword arguments". <u>Most importantly</u>, all the changes required to silence these warnings are <u>compatible with 2.6</u>.</p> <pre>def foo(x, **kw); end</pre>	

```
foo(k:1)      # ArgumentError because x not specified
foo(1, {k:1}) # ArgumentError because too many arguments; Hash cannot be converted to Kwargs
opts = [k:1].first
foo(opts)
# opts is a Kwargs therefore used as keyword argument; ArgumentError because x not specified
foo(1, opts) # opts is a Kwargs therefore used as keyword argument
```

The warnings for this would be about upcoming errors for positional arguments: `foo(x:1)` will be "given 0, expected 1" and `foo(1,{x:2})` will be "given 2, expected 1". Such errors are useful when developing, but there is no new functionality per se, just a stricter syntax. So it's less important to escalate to an error and we can keep the warnings for longer than Step 2.

At this point we have achieved almost-fulldynamic keyword separation, as opposed to the current almost-fullstatic approach. I want to make the point here that, yes, keyword arguments **are** separated, it's just a different paradigm. With static separation, a keyword argument is defined lexically by a double-splat. With dynamic separation, a keyword argument is when the last argument is a Kwargs. {{Note: I'm saying "almost-full" because Kwargs is not promoted to keywords in `def foo(a,**kw);end;foo(x:1)` and because static keywords are auto-demoted to positional in `def foo(a);end;foo(x:1)`}}

Any form of delegation works with no change required. This preserves the behavior of 2.6 but only for Kwargs objects. This is similar to having 2.7 with `ruby2_keywords` enabled by default. But also different in some ways; most notably it allows the case shown in [#16494](#) to work by default:

```
array = [x:1]
array.push(x:2)
array.map{ |x:| x } #=> [1,2]
[{:x:3}].map{ |x:| x } #=> but this warns, as it should
```

The current approach does not allow this to work at all. The solution proposed in [#16494](#) has all the same flaws as Hash-based keyword arguments; what happens to each `{x:nil,**kw}` ? This solution allows a Kwargs to be converted to... keywords. Very unsurprising.

Given that ruby is a dynamically-typed language I feel that dynamic typing of keywords is a more natural fit than static typing. But I realize that many disagree with that, which is why we continue to...

Step 4

Introduce additional incompatibility to reach static/lexical separation of keyword arguments. Here we require that even a Kwargs should be passed with a double-splat in order to qualify as a keyword argument.

```
def bar(**kw)
end
def foo(**kw)
  bar(kw)    #=> error; Kwargs passed without **
  bar(**kw) #=> ok
end
```

At this point we've reached the same behavior as 2.7. Delegation needs to be fixed, but as we know the changes required to silence these warnings are **not** compatible with 2.6 or 2.7. The warnings for this are fundamentally not fixable as long as Step 2 has not been fixed. This is the core reason why `ruby2_keywords` is currently necessary in 2.7. So in the version after 2.7 we can enable these warnings by default since it's now possible to fix delegation to use static keywords. Except that gem authors who need to stay compatible with ≤ 2.7 cannot yet make these changes, so we introduce a way to silence only these "Step 4" warnings, for people who need to remain compatible with ≤ 2.7 . And we keep them as warnings instead of errors until ruby 2.7 is EOL.

So instead of having to update a bunch of places with `ruby2_keywords` just to temporarily silence warnings, it's a single flag like `Warning[:ruby3_keywords]`. Once ruby 2.7 is EOL these become controlled by `Warning[:deprecated]` which tells people they **have** to fix their code. Which is just like the eventual deprecation of `ruby2_keywords`, just without the busy work of adding `ruby2_keywords` statements in the first place. But again, this introduces no new functionality, just a stricter syntax. So we can play nice and leave the warnings for a few years before changing to errors.

The question remains of how to handle [#16494](#) here. Either disallow it entirely, but I think that would be a shame. Or just like [#16494](#) suggests, allow hash unpacking in non-lambda Proc. Except that now it can be a Kwargs instead of a Hash, which at least preserves dynamic keyword separation.

Putting it all together (TL;DR)

The idea is not to reimplement keyword argument separation; all that is needed is to implement the things above that are not in 2.7:

- Create a KwHash object for brace-less and double-splatted hashes.
- Differentiate the various types of warnings and allow to toggle on/off separately
 - Step 2 warnings must be fixed now; cannot toggle off
 - Step 3 warnings should be fixed now but you don't absolutely need to upgrade your gems just for that
 - Step 4 warnings should be fixed in next version unless you need to support ≤ 2.7

I think that's all, really...

Pros

- Cleaner way to solve [#16494](#)
- Better compatibility (at least until 2.6 is EOL)
 - delegation
 - storing an argument list that ends with a KwHash
 - destructuring iteration ([#16494](#))
- We can avoid the "unfortunate corner case" as described in the [release notes](#)
 - in 2.7 only do not output "Step 4" warnings, leave delegation like it was
 - in 2.8 the "Step 3" warnings have been fixed and a Hash will not be converted to keyword arguments
 - delegation can now safely be fixed to use the `**` syntax
- `ruby2_keywords` is not required, which is desirable because
 - it's a hidden flag [hack](#)
 - it requires to change the code now, and change it again when `ruby2_keywords` is deprecated; twice the work; twice the gem upgrades
 - it was supposed to be used only for people who need to support 2.6 or below, but it's being misunderstood as an acceptable way to fix delegation in general
 - there's the non-zero risk that `ruby2_keywords` will never be removed, leaving us with a permanent "hack mode"
 - dynamic keywords are by far preferable to supporting `ruby2_keywords` forever
- Likely better performance, as the KwHash class can be optimized specifically for the characteristics of keyword arguments.
- More flexible migration
 - Allow more time to upgrade the hard stuff in Step 4
 - Can reach the same goal as the current static approach
 - Larger "support zone" <https://xkcd.com/2224/>
 - Instead of wide-ranging incompatibilities all at once, there's the possibility of making it finer-grained and more gradual
 - rubyists can choose to migrate all at once or in smaller chunks
 - It hedges the risks by keeping more possibilities open for now.
 - It allows to cop-out at Step 3 if Step 4 turns out too hard because it breaks too much stuff

Cons

- It allows to cop-out at Step 3 if Step 4 turns out too hard because it breaks too much stuff

Related issues:

Related to Ruby master - Feature #16463: Fixing `*args`-delegation in Ruby 2.7:...

Closed

History

#1 - 01/16/2020 07:33 AM - Eregon (Benoit Daloze)

Thank you for filing this and explaining your idea.

My understanding is this is basically a different way to represent a flagged Hash, i.e., flagged Hash becomes KwHash in this proposal. Compared to [#16463](#), what are the differences except the representation of the flagged Hash?

Step 2 is not done in Ruby 2.7.0, I think it's too incompatible and needs warnings before changing that behavior. It will work like that in 3.0 though.

Most importantly, it allows the case shown in [#16494](#) to work by default:
`array = [x:1]; array.push(x:2); array.map{ |x| x } #=> [1,2]`

But `array = [{x:1}]` would warn, right? And `array << {x:2}` would also warn. In such a case I consider the hashes in `array` as data Hashes, e.g., coming from `JSON.parse`, and so IMHO should not be KwHash. This idea doesn't really solve that, just makes some cases easier.

#2 - 01/16/2020 07:41 AM - Eregon (Benoit Daloze)

- Related to Feature #16463: Fixing `*args`-delegation in Ruby 2.7: `ruby2_keywords` semantics by default in 2.7.1 added

#3 - 01/16/2020 08:47 AM - matz (Yukihiro Matsumoto)

I was going to use an internal flag instead of making them subclass. Subclassing standard class has often made trouble in the past. I am negative.

Matz.

#4 - 01/16/2020 08:57 AM - shevegen (Robert A. Heiler)

Matz already commented. :)

I will briefly give my opinion. I think aside from special cases, one issue is that ruby users have to understand why different/specialized hashes are used, be this `KwHash` or `HashWithIndifferentAccess` or any other variant. IMO, from that point of view, I'd personally prefer to not have special names and special subclasses be used if it could be avoided and keep (core) ruby simple(r). (A tiny issue may also be the name; `KwHash` reads somewhat strangely.)

#5 - 01/16/2020 03:55 PM - Dan0042 (Daniel DeLorme)

matz (Yukihiro Matsumoto) wrote:

I was going to use an internal flag instead of making them subclass.

Eregon (Benoit Daloze) wrote:

My understanding is this is basically a different way to represent a flagged Hash, i.e., flagged Hash becomes `KwHash` in this proposal. Compared to [#16463](#), what are the differences except the representation of the flagged Hash?

Well, I guess everything I described above could be achieved with a flag as well. After all when you get down to it an object's class is also just a flag that defines which set of behavior the object has. It's just that naturally I tend to consider an object-oriented design preferable to a flag-oriented design.

The difference with [#16463](#) is that every instance of a double-splat or brace-less hash would have the flag, instead of just the ones passed to a method with a `*rest` argument.

But I guess expanding the use of the flag in this way would be possible even in 2.7.x, and after that all that's needed is to add a separate set of warnings for hashes with the flag.

Step 2 is not done in Ruby 2.7.0, I think it's too incompatible and needs warnings before changing that behavior. It will work like that in 3.0 though.

Ah yes, of course there's a warning phase, I neglected to say it explicitly. I just wanted to illustrate that fixing the bugs of keyword arguments required few changes, while fixing the design requires more.

Most importantly, it allows the case shown in [#16494](#) to work by default:
`array = [x:1]; array.push(x:2); array.map{ |x| x } #=> [1,2]`

But `array = [{x:1}]` would warn, right? And `array << {x:2}` would also warn. In such a case I consider the hashes in array as data Hashes, e.g., coming from `JSON.parse`, and so IMHO should not be `KwHash`.

Yes, exactly. That way it's possible to opt in or out of the behavior.

This idea doesn't really solve that, just makes some cases easier.

????

Yes, it does solve that, it's the entire point of this proposal.

Subclassing standard class has often made trouble in the past.

I'd really like to know what kind of trouble. From what I know/remember there have been issues with e.g. `Subclass.select{}` returning an instance of `BaseClass`, but in this case it's exactly what we'd want, just like `flaggedhash.select{}` would return a non-flagged hash.

#6 - 01/16/2020 05:36 PM - jeremyevans0 (Jeremy Evans)

I believe this approach would break the following code:

```
def debug_log(arg, output: $stderr)
  output.print(arg.inspect)
  output.print("\n")
end
```

```
def bar(*args, **opts)
  debug_log(args)
  debug_log(opts)
  # do something
end
bar(:baz, quux:1)
```

With the master branch, {quux:1} is considered a keyword argument by bar, but passed as a positional argument to debug_log. This works fine as you would expect it to. With the KwHash approach, {quux:1} ends up being passed as a keyword argument to debug_log, resulting in ArgumentError. This results in even greater backwards compatibility issues than the ruby2_keywords by default approach.

In general, whether something is a keyword argument or a positional argument is something that should be decided on a per-call site basis, it shouldn't be a property of an object. ruby2_keywords is strictly a work around to support older Ruby versions without requiring separate code paths, to ease supporting multiple Ruby versions at once. When targeting only Ruby 3+, there would be no reason to use it (except performance when using CRuby).

The KwHash approach would result in the problems that keyword argument separation was designed to avoid, since you would have cases where positional arguments to methods are treated as keyword arguments depending on how they were created. With the current master branch, that can only happen in limited cases on an opt-in basis when using ruby2_keywords. Presumably, because it is opt-in, there will not be cases where positional arguments are ever mistreated as keyword arguments, without the deliberate misuse of ruby2_keywords.

#7 - 01/16/2020 07:27 PM - Dan0042 (Daniel DeLorme)

jeremylvans0 (Jeremy Evans) wrote:

I believe this approach would break the following code:

```
def debug_log(arg, output: $stderr)
  output.print(arg.inspect)
  output.print("\n")
end
def bar(*args, **opts)
  debug_log(args)
  debug_log(opts)
  # do something
end
bar(:baz, quux:1)
```

You're right! Thank you for finding an actual specific case (and a plausible one at that!) In this case, in ruby 2.7.KwHash, you'd get a warning to the effect that the KwHash will eventually be used for keyword arguments instead of positional, just as if you wrote debug_log(**opts) in 2.7; and you'd need to opt-in to the desired behavior with debug_log(opts.to_h) or debug_log({**opts}). Whereas in 2.6 and 2.7 it works with no warning. With the "Step 4" warnings you'd also get a warning to the effect that opts should be passed with a double-splat. Just like 2.7 the basic rule is still to warn for any change in behavior first.

But still, I believe that cases like this where you'd want a keyword hash to be interpreted as a regular hash are quite rare (although I could be wrong on that point). So overall I still believe in the KwHash approach.

This results in even greater backwards compatibility issues than the ruby2_keywords by default approach.

I don't understand your notion of backward compatibility. It seems to me like you're conflating "backward compatibility" with "desired behavior in 3.0". Backward compatibility is when things that worked in 2.6 still work the same way with no change. The example you showed demonstrates a backward compatibility breakage, yes. But many more other cases are not backward compatible in 2.7 and become backward compatible with the KwHash or ruby2_keywords by default approach.

In general, whether something is a keyword argument or a positional argument is something that should be decided on a per-call site basis, it shouldn't be a property of an object.

I know this is your opinion, but I hope you can realize it is an opinion about what the design should be like. My opinion differs, on the basis of putting more importance on backward compatibility.

ruby2_keywords is strictly a work around to support older Ruby versions without requiring separate code paths, to ease supporting multiple Ruby versions at once. When targeting only Ruby 3+, there would be no reason to use it (except performance when using CRuby).

But "supporting multiple Ruby versions at once" is not some rare thing, it's what all gems should do (until 2.6 is EOL).

The KwHash approach would result in the problems that keyword argument separation was designed to avoid

No, this is incorrect. If you take into account the "Step 4" I described above, the result would be exactly the same as the current master branch, with the exception that passing a KwHash as last argument must be disambiguated as either debug_log(**opts) or debug_log({**opts}). The biggest difference is how much pain and inconvenience we cause to other ruby programmers along the way. I really think that's worth minimizing.

#8 - 01/17/2020 04:04 AM - Dan0042 (Daniel DeLorme)

- Description updated

I'm rethinking the issue pointed out by Jeremy. It makes no sense to introduce a new kind of incompatibility for the sake of a compatibility layer. The behavior of KwHash must be identical to the behavior of Hash in 2.6, meaning that a positional KwHash is only promoted to keyword argument if there are enough other positional arguments to satisfy the minimum arity. So the example given by Jeremy would work. Conceptually it's not ideal, but the important thing in this case is backward compatibility.

#9 - 01/17/2020 04:00 PM - jeremyevans0 (Jeremy Evans)

Dan0042 (Daniel DeLorme) wrote:

I'm rethinking the issue pointed out by Jeremy. It makes no sense to introduce a new kind of incompatibility for the sake of a compatibility layer. The behavior of KwHash must be identical to the behavior of Hash in 2.6, meaning that a positional KwHash is only promoted to keyword argument if there are enough other positional arguments to satisfy the minimum arity. So the example given by Jeremy would work. Conceptually it's not ideal, but the important thing in this case is backward compatibility.

That still results in the issue that keyword argument separation was meant to fix, with a slightly modified example (which works correctly in the master branch, and fails in Ruby 2.*):

```
def debug_log(arg=nil, output: $stderr)
  output.print(arg.inspect)
  output.print("\n")
end
def bar(*args, **opts)
  debug_log(args)
  debug_log(opts)
  # do something
end
bar(:baz, quux:1)
```

In this case the minimum arity is met, but you would still end up with ArgumentError because it would treat the debug_log(opts) call as passing keywords instead of a positional hash, and debug_log doesn't support a quux keyword.

#10 - 01/17/2020 04:20 PM - Dan0042 (Daniel DeLorme)

jeremyevans0 (Jeremy Evans) wrote:

In this case the minimum arity is met, but you would still end up with ArgumentError because it would treat the debug_log(opts) call as passing keywords instead of a positional hash, and debug_log doesn't support a quux keyword.

But most importantly it's compatible with 2.6. In other words since it breaks in 2.6 this case "doesn't exist" in real production code; there's no compatibility issue, and no "real" problem. It's also rational; since opts is a KwHash it conceptually makes sense to interpret it as keyword arguments. Maybe not in your static paradigm, but dynamically yes. So we just keep this backward compatibility with warnings for a while longer until 2.6 (or 2.7?) is EOL. Keyword arguments have worked mostly fine since 2.0, there's no reason to be in such a rush to break things right now now.

If you really want the new "fixed" but backward incompatible behavior, you have to opt in to it with debug_log(**opts) or such. Unlike 2.6 this would finally be possible. It's far "better" to opt in to the incompatible behavior for a few cases like this than to force everyone to opt out of the incompatible behavior for a much larger number of cases. And by "better" here I mean more respectful of people's time and trouble.

#11 - 01/20/2020 12:51 AM - Dan0042 (Daniel DeLorme)

- Description updated

- Subject changed from Subclass of Hash for keyword arguments to Staged warnings for keyword arguments

I'm adjusting the proposal to focus on the goal (flexible deprecation) rather than the means (subclass or flag)

#12 - 01/29/2020 03:37 AM - Dan0042 (Daniel DeLorme)

Ok, it took a while to play with the vm_args code enough to really understand it, but I finally have my own **proof of concept** ready! □□□□□□□□
<https://github.com/dan42/ruby/tree/kwarg-warnings-16511>

Please check it out. I'm sure there are still some bugs to work out, though, so don't take that to mean that the entire idea is flawed.

Note: I heavily refactored the vm_args code in order to understand it, but please don't focus on that; this is intended as proof of concept to demonstrate the benefits of this approach, not a polished pull request. (Although I am pretty happy about how those refactorings turned out.)

Below I refer to 2.7.e for proposal [#16463](#); and 2.7.d / 3.x.d for this proposal.

Benefits:

Three levels of urgency in warnings

[FIX NOW]: must be fixed now since behavior will be different in 3.0, for example Hash is no longer converted to keywords

[fix now]: will eventually raise error; should be fixed now but there's no hurry or benefit to break people's stuff in 3.0

[fix in 3.x]: will eventually raise error; fix is not compatible with 2.x; therefore message is not displayed in 2.7; fix once you're ready to drop support for ruby 2.x

This is because there are different needs:

- app developer cares about his current version of ruby only
- gem developer cares about remaining compatible with a few versions
- gem user doesn't want to upgrade to a new major version of the gem with an incompatible API

In this proof of concept the three labels above are simply appended to the warnings, but in a final version they would instead change the visibility of the warnings.

Delegation is backward compatible

```
def foo(*args); p args; end
def bar(**kw); p kw; end
def deleg(m,*args); send(m,*args); end
ruby2_keywords def deleg2(m,*args); send(m,*args); end
def delegV(m,a); send(m,a); end

# 2.7      2.7.e   2.7.d
deleg(:bar, x:1) #=> {:x=>1} # warning - -
deleg2(:bar, x:1) #=> {:x=>1} # - - -
delegV(:bar, x:1) #=> {:x=>1} # warning warning -
deleg(:foo, 2) #=> [2]
deleg2(:foo, 2) #=> [2]
delegV(:foo, 2) #=> [2]
```

This example demonstrates that delegation can break even if it doesn't use *args. In this case you'd have to change delegV to use *args delegation with ruby2_keywords. I acknowledge this is unlikely to happen in practice, but since it's theoretically possible... all else being equal why not choose the approach that handles even this remote possibility?

Keywords-to-positional is compatible with 2.6

```
def foo(*args); p args; end
h0 = {}
h1 = {k: 42}
foo() #=> []
foo(**h0) #=> [{}] (like 2.6, unlike 2.7)
foo(**h1) #=> [{:k=>42}]

def bar(opts); p opts; end
h = {}
h[:k] = 1 if condition
bar(**h) #=> {} if condition is false; unlike warning in 2.7 and error in master
#=> {:k=>1} if condition is true
```

Since it's been decided that the conversion from keywords to positional will remain forever for the sake of compatibility, I strongly believe it makes no sense to introduce an incompatibility in this compatibility behavior.

**kw delegation doesn't introduce additional hash

```
def foo(*args); p args; end
def deleg(*a,**k); foo(*a,**k); end
h0 = {}
h1 = {k: 42}
p foo() ==deleg() #=> [] == [] #=> true, unlike [] == [{}] in 2.6
p foo(**h0)==deleg(**h0) #=> [{}] == [{}] #=> true ([] == [] in 2.7)
p foo(**h1)==deleg(**h1) #=> [{:k=>42}] == [{:k=>42}] #=> true
p foo({}) ==deleg({})
#=> [{}] == [{}] #=> true, unlike [{}] == [] in 2.7 (both versions display a warning)
```

I introduced a distinction between an empty kwsplat and a "nonexistent" kwsplat; the later is not converted to an empty positional hash. I've come to believe this is necessary in order to have both keywords-to-positional compatibility and **kw delegation work at the same time.

Destructuring iteration is compatible

```
p [x:1].push(x:2).map{ |x:| x } #=> [1, 2] with warning in 2.7 and 2.7.e and 3.x.d
p [{x:1}].push({x:2}).map{ |x:| x } #=> [1, 2] with warning in 2.7 and 2.7.e and 2.7.d
```

It may not be a common pattern, but at least 2 people have reported this being an issue. Since there's a solution that allows this to work, why not take

it? In the end maybe matz will decide to kill this feature but I would prefer to have an actual discussion on this rather than rush the "kill" decision just because the current approach doesn't allow this flexibility.

(But non-symbol keywords are still incompatible)

This approach doesn't change anything to this 2.7 change in behavior:

```
def foo(attr={}, **opt) [attr,opt] end
foo("a">1) #=> [{"a">1}, {}] in 2.6
           #=> [{} , {"a">1}] in 2.7.*
```

#13 - 01/29/2020 10:42 AM - Eregon (Benoit Daloze)

FWIW, I also discussed to keep `**empty_hash` in the case it's passed by the user in <https://bugs.ruby-lang.org/issues/16519#note-5> Jeremy seems clearly against it, and I get that `**empty_hash` always passing nothing is simpler to understand.

I'm not sure what's the design of your proposal.
What do you change to keep `*args`-delegation working?

It seems unlikely Ruby core would accept to backport anything more complicated than [#16463](#) to 2.7.

#14 - 01/29/2020 02:16 PM - Dan0042 (Daniel DeLorme)

FWIW, I also discussed to keep `**empty_hash` in the case it's passed by the user in <https://bugs.ruby-lang.org/issues/16519#note-5>

Ah, I missed that one. I was actually inspired to keep the behavior by your PR in [#16463](#). I haven't posted that much there but that's because I very much agree with what you're saying. I can't thank you enough for the efforts you've been putting in this.

Jeremy seems clearly against it, and I get that `**empty_hash` always passing nothing is simpler to understand.

Now that I understand the `vm_args` code I've gained a better appreciation for Jeremy's viewpoint. The splatting of an empty hash interacts in a really nasty way with `*rest` and `**kwrest` delegation. But I don't think that always passing nothing is simpler...

```
def foo(opts); p opts; end
opts = {}
opts[:k] = 1 if some_condition
foo(**opts)
```

In 2.7 this results in a warning depending on `some_condition`, that's really not simple to understand. :-/

I'm not sure what's the design of your proposal.
What do you change to keep `*args`-delegation working?

Well, the design is described in extremely thorough detail above, although I realize it's a massive wall of text to read. I guess you can boil it down to having the keyword flag set on every splatted hash instead of just when accepting a `*rest` argument. That way is much more logical; it's a coherent part of the design instead of a special-case hack (as explained in the proposal description).

It seems unlikely Ruby core would accept to backport anything more complicated than [#16463](#) to 2.7.

Despite having demonstrable benefits? That would make me sad. Enabling `ruby2_keywords` by default gets 90% of the benefit for 10% of the complexity, but I think it's worth going to 100%, and I've done it. The changes are not even that complicated.

#15 - 01/29/2020 04:57 PM - Dan0042 (Daniel DeLorme)

gem user doesn't want to upgrade to a new major version of the gem with an incompatible API

I think this part deserves explanation because so far I haven't seen much discussion of what happens to people who use gems.
Please consider this scenario:

- author of gem "foobar" fixes the keyword warnings in ruby 2.7 and publishes a new version 3.1.0 (only v3 is supported)
- gem user finds keyword warnings coming from foobar 1.4.7
- he upgrades teeny version to 1.4.12 but the warnings are still there
- he upgrades minor version to 1.6.3 but the warnings are still there
- he must upgrade to 3.1.0 but the API has changed and requires various fixes to the app
- he can't upgrade to ruby 3.0 as long as this isn't done
- so he has to fix the app to use foobar 3.1.0

- **3 years pass...** ruby2_keywords are being deprecated
- author of gem "foobar" changes ruby2_keywords to **kwrest delegation and publishes a new version 4.2.0 (only v4 is supported)
- user of gem "foobar" finds keyword warnings coming from the gem
- he upgrades teeny version to 3.1.7 but the warnings are still there
- he upgrades minor version to 3.2.2 but the warnings are still there
- he must upgrade to 4.2.0 but the API has changed and requires various fixes to the app
- he can't upgrade to ruby 3.3 as long as this isn't done
- so he has to fix the app AGAIN, to use foobar 4.2.0

This is a worst-case scenario but it doesn't sound so farfetched to me.

What I would like to see is this:

- author of gem "foobar" fixes the keyword warnings (except [fix in 3.x]) and publishes a new version 3.1.0 (only v3 is supported)
- only [FIX NOW] warnings are displayed in the case of gems in RUBYGEMS_DIR, and they are pretty rare
- user of gem "foobar" finds no keyword warnings coming from the gem, nothing to upgrade except his own code
- **3 years pass...** [fix now] and [fix in 3.x] warnings are enabled by default even in RUBYGEMS_DIR
- author of gem "foobar" fixes the warnings to use ** syntax and publishes a new version 4.2.0 (only v4 is supported)
- gem user finds keyword warnings coming from foobar 1.4.7
- he upgrades teeny version to 1.4.12 but the warnings are still there
- he upgrades minor version to 1.6.3 but the warnings are still there
- he must upgrade to 4.2.0 but the API has changed and requires various fixes to the app
- he can't upgrade to ruby 3.4 as long as this isn't done, but that's in two years
- so he has some time to fix the app, ONCE, to use foobar 4.2.0

Isn't that better? Note that the worst case would still involve two upgrades, but it would be significantly less likely: only if a gem has warnings with both [FIX NOW] and [fix in 3.x] warnings.

#16 - 01/31/2020 03:30 PM - Dan0042 (Daniel DeLorme)

- Subject changed from Staged warnings for keyword arguments to Staged warnings and better compatibility for keyword arguments in 2.7.1

[matz \(Yukihiko Matsumoto\)](#),

I removed the subclassing part that you were negative about, and I believe I've sufficiently proven many clear real-world benefits. The only "downside" is that some things that could be fixed now will be fixed later instead; personally I don't even consider that a downside. I hope you'll give this proposal serious consideration, based on the demonstrated benefits, despite me being a relative outsider here.

□□□□□□□□□□

#17 - 02/02/2020 08:31 PM - Eregon (Benoit Daloze)

Dan0042 (Daniel DeLorme) wrote in [#note-14](#):

In 2.7 this results in a warning depending on some_condition, that's really not simple to understand. :-/

I'd guess it's fairly uncommon to call a method not accepting keyword arguments with **.

I'm not sure what's the design of your proposal.
What do you change to keep *args-delegation working?

Well, the design is described in extremely thorough detail above, although I realize it's a massive wall of text to read. I guess you can boil it down to having the keyword flag set on every splatted hash instead of just when accepting a *rest argument. That way is much more logical; it's a coherent part of the design instead of a special-case hack (as explained in the proposal description).

I read the first post multiple times but I can't quite understand what it precisely means. <https://bugs.ruby-lang.org/issues/16511#note-12> is more precise, but still quite abstract to me.

Could you show a tentative timeline for your proposal?
I.e., when each deprecation is introduced and when behavior is changed.

Treating as keywords even for foo(h) if h is flagged seems like it would significantly increase the chance to not behave as intended, as Jeremy showed. Considering minimum arity like in <https://bugs.ruby-lang.org/issues/16511#note-8> would IMHO breaks the separation and add a new exceptional case. It's likely to also be very brittle if there are non-required arguments. Would such behavior be eventually removed in your proposal? Performance-wise it likely has some cost as it causes extra checks.

The ruby2_keywords rule to only consider *args is a very precise way to detect delegation in Ruby. For that reason I think it's valuable to only have special behavior for that case, and only until we can safely migrate to (*args, **kwargs)-delegation.

If I understand correctly from the first post, your proposal has more steps than mine. My proposal has 2 steps: deprecate passing keyword arguments without **/foo: in 2.7 (already done in 2.7.0) and deprecate delegation in Ruby 3.warn (e.g., Ruby 3.4, can use (*args, **kwargs)-delegation then).

For simplicity let's assume each release after the deprecation changes to the new behavior. I'm afraid more steps will bring more confusion and slow down even more adoption.

Destructuring iteration is compatible

That would only work in very few cases.

For example if the Hash is created by JSON.load, then it won't be marked as keyword, and .map{ |x:| x } won't work.

Also if you wanted to make this work on an Array of multiple elements you'd need something like `[**h1, **h2].map{ |x:| x }` which is both weird and doesn't work (hashes are merged).

That's an abuse of the automatic conversion, let's kill it, there is no way to preserve it without breaking the separation.

#18 - 02/13/2020 03:48 AM - Dan0042 (Daniel DeLorme)

I'd guess it's fairly uncommon to call a method not accepting keyword arguments with `**`.

Maybe, who knows. I'd rather not make assumptions on what is "uncommon".

Could you show a tentative timeline for your proposal?

I.e., when each deprecation is introduced and when behavior is changed.

Ok, fair enough. The methods and table below should cover all possible warnings and errors.

```
def foo(lead, opt=nil, k:nil)
  {lead: lead, opt: opt, k: k}
end
def bar(lead, opt=nil, **kw)
  {lead: lead, opt: opt, kw: kw}
end
def deleg(m, *rest)
  send(m, *rest)
end
```

method call	2.6 / 2.7 result	2.7	3.0 to 3.2	3.3 to 3.4 (2.7 is eol)	3.5	warning message
foo(k:1)	{:lead=>{:k=>1}, :opt=>nil, :k=>nil}	warning*	warning*	warning	error: wrong number of arguments	Passing the keyword argument as the last hash parameter is deprecated
bar(k:1)	{:lead=>{:k=>1}, :opt=>nil, :kw=>{}}	warning*	warning*	warning	error: wrong number of arguments	Passing the keyword argument as the last hash parameter is deprecated
foo(1, "a"=>2)	{:lead=>1, :opt=>{"a"=>2}, :k=>nil}	warning*	warning*	warning	error: wrong number of arguments	Passing the keyword argument as the last hash parameter is deprecated
bar(1, "a"=>2)	{:lead=>1, :opt=>nil, :kw=>{"a"=>2}}					
foo(1, "a"=>2, k:3)	{:lead=>1, :opt=>{"a"=>2}, :k=>3}	warning	error: wrong number of arguments			Splitting the last argument into positional and keyword parameters is deprecated
bar(1, "a"=>2, k:3)	{:lead=>1, :opt=>nil, :kw=>{"a"=>2, :k=>3}}					
foo(1, {"a"=>2, k:3})	{:lead=>1, :opt=>{"a"=>2}, :k=>3}	warning	{:lead=>1, :opt=>{"a"=>2, :k=>3}, :k=>nil}			Splitting the last argument into positional and keyword

method call	2.6 / 2.7 result	2.7	3.0 to 3.2	3.3 to 3.4 (2.7 is eol)	3.5	warning message
						parameters is deprecated
bar(1, {"a"=>2, k:3})	{:lead=>1, :opt=>{"a"=>2}, :kw=>{:k=>3}}	warning	{:lead=>1, :opt=>{"a"=>2, :k=>3}, :k=>nil}			Splitting the last argument into positional and keyword parameters is deprecated
foo(1, {k:2})	{:lead=>1, :opt=>nil, :k=>2}	warning	{:lead=>1, :opt=>{:k=>2}, :k=>nil}			Using the last argument as keyword parameters is deprecated
foo(1, 2, {k:3})	{:lead=>1, :opt=>2, :k=>3}	warning*	warning*	warning	error: wrong number of arguments	Using the last argument as keyword parameters is deprecated
deleg(:foo, 1, k:2)	{:lead=>1, :opt=>nil, :k=>2}		opt-in warning*	warning	{:lead=>1, :opt=>{:k=>2}, :k=>nil}	Using the last argument as keyword parameters is deprecated
deleg(:foo, 1, 2, k:3)	{:lead=>1, :opt=>2, :k=>3}		opt-in warning*	warning	error: wrong number of arguments	Using the last argument as keyword parameters is deprecated

Blank cells mean there's no change from the previous version.

warning* means there's a warning except if it comes from a gem (from a file in Gem::RUBYGEMS_DIR)

opt-in warning is for people who want to stop supporting ruby 2.x before 2.7 eol.

Treating as keywords even for foo(h) if h is flagged seems like it would significantly increase the chance to not behave as intended, as Jeremy showed.

Considering minimum arity like in <https://bugs.ruby-lang.org/issues/16511#note-8> would IMHO breaks the separation and add a new exceptional case.

It's likely to also be very brittle if there are non-required arguments.

I'm not sure what you consider as "intended" behavior, but for me it means that

- 1) positional hash should not be converted to keywords;
- 2) keywords should be not be converted to positional hash (except for positional-only methods);
- 3) during the transition period, delegation should work for keywords even if not using ****kwrest**;
- 4) everything else should continue working as it worked in the past.

Since the behavior of a flagged hash in 2.7 is exactly like any hash in 2.6, the compatibility for that case is 100%. At least for now, but it would still warn in 3.3. And considering minimum arity is something that was already being done in 2.6, so it's not some new exceptional case. And it would still show a warning, so it's not breaking separation. There's nothing brittle in there; would you care to elaborate?

Would such behavior be eventually removed in your proposal? Performance-wise it likely has some cost as it causes extra checks.

Yes, it would be removed. It's somewhat equivalent to ruby2_keywords by default but more comprehensive. Performance-wise it's very unlikely to have a measurable cost, based on the code I've seen in vm_args.c

The ruby2_keywords rule to only consider *args is a very precise way to detect delegation in Ruby.

For that reason I think it's valuable to only have special behavior for that case, and only until we can safely migrate to (*args, **kwargs)-delegation.

It's actually not that precise. I think this is Jeremy's main worry about ruby2_keywords by default, that *args is not always used only for delegation.

If I understand correctly from the first post, your proposal has more steps than mine.

No, it has the same number of steps, it's just more lenient about things that we can afford to be lenient about. The "Steps" in the proposal are only conceptual steps, in order to explain the logic. It's still one round of deprecations in 2.7 (Steps 1-3) and another in 3.warn (Step 4). And for end-users

it boils down to the same thing: if you see a warning, fix it. There's just fewer warnings to fix at once.

Destructuring iteration is compatible

That would only work in very few cases.

That's an abuse of the automatic conversion, let's kill it, there is no way to preserve it without breaking the separation.

I disagree this is abusive or hacky code. Although I never used it before, my first impression was of a very clean and powerful idiom. [#16494](#) only showed a words array so I don't know how it was built. Perhaps neither of your `JSON.load` or `[**h1, **h2]` examples is relevant. What's important though is that it can be made to work quite easily, without having to restructure everything. And since it behaves differently based on having a positional hash or a keyword (flagged) hash, that means separation is preserved, at least to some extent.

#19 - 02/23/2020 02:24 PM - Eregon (Benoit Daloze)

Thanks for writing that out.

I think in general tracking whether a Hash is keywords or positional dynamically makes everything complicated.

Long-term, I think it's clear we want to know syntactically whether a Hash is keyword arguments or positional.

Anything that goes against that long term IMHO breaks the separation and makes the separation pretty moot.

I think it's very important to have that syntactical separation, otherwise one cannot understand the behavior of calling a method just by looking at caller and callee, which would be a very large issue for non-trivial codebases.

For that reason, I'm against anything more dynamic than in 2.7.0 and `ruby2_keywords`.

From my understanding of this proposal, it's more dynamic, so I'm against it.

Destruction iteration as you propose relies on dynamic tracking of keywords vs positional, so for me that's a big no-no, as it will have to break to have a syntactical separation.

That's also why I have been against `ruby2_keywords` from the start, and wish we would only use syntactic ways to delegate (e.g., only `*args` in the same methods passes the `kwargs` as-is, or ...).

However, since `ruby2_keywords` is there and it seems to be the only reasonable way now to preserve delegation in 2.7 (in terms of amount of changes and risk),

I think it makes sense to be the default, so `*args`-delegation still works and there is still an intuitive syntax to do delegation in a compatible way with earlier versions.

`*args` is not perfect but it's the best pattern we have to capture delegation AFAIK.

If nothing breaks in Rails due to adding `ruby2_keywords` semantics to `*args` it's a good sign it's either used for delegation, or in a way `ruby2_keywords` doesn't break anything.

Minimum arity might be something to help transition, but long-term for syntactical separation it has to be removed as well (e.g., syntactic `kwargs` passed to a method accepting `kwargs` should either be passed as `kwargs` or `ArgumentError`, no "treat as positional because not enough arguments"), so I think it shouldn't stay any longer than needed.

Rules related to minimum arity is brought this whole ambiguity between positional and keyword arguments in the first place.

In general, I doubt such a large change would even be considered for 2.7 or later by most MRI committers, because it would be a big risk.

#20 - 02/25/2020 06:54 PM - Dan0042 (Daniel DeLorme)

[Eregon \(Benoit Daloze\)](#) thanks for your reply.

I won't bother making a point-by-point counterargument because it would be redundant and boring at this point. Suffice it to say that I find almost every single line you wrote to be either flawed or factually incorrect or something I disagree with. We both know where we stand, and we've expounded those arguments in great detail above, so let's say we agree to disagree. Except for this:

In general, I doubt such a large change would even be considered for 2.7 or later by most MRI committers

Indeed I agree. I'm not expecting to have any impact. I only added this proposal to the dev meeting list because, after spending so much time and effort on this (digging into the VM internals was surprisingly addictive), it would be absurd not to see it through to the bitter end. At least my conscience will be clear and I'll be able to say I did everything I could if things turn sour. Sincerely hoping that they don't.