Ruby master - Feature #16463

Fixing *args-delegation in Ruby 2.7: ruby2_keywords semantics by default in 2.7.1

12/28/2019 12:34 PM - Eregon (Benoit Daloze)

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

Description

Ruby 2.7.0 is out.
It aims to warn for every keyword argument change that will happen in Ruby 3.0.
Most warnings are useful: adding **, etc is needed to not break code when migrating to 3.0.

Ruby 2.7 also aims at remaining compatible with 2.6.
However there is a big breaking change here: ***args-delegation broke in Ruby 2.7 for keyword arguments**.
The workaround is adding ruby2_keywords to the method/block receiving the keywords arguments to delegate later on.

But is it needed or useful at all to require everyone to add ruby2_keywords in many places of their codebase?
And for rubyists to get major headaches as to why *args-delegation broke and instead has strange semantics in Ruby 2.7?
Was it useful to break delegation in Ruby 2.7?

I think not, and here I propose a solution to keep delegation in 2.7 compatible with 2.6 (just use *args as before).

First I'll introduce some context.
The end goal is to have separation of positional and keyword arguments.
However, this will not happen in 3.0, because as long as ruby2_keyword exist, the separation will only be partial.
For example, foo(*args) should only pass positional arguments, never keyword arguments, but this can only be guaranteed once ruby2_keyword is removed.

The plan to get there, as far as I heard and imagine it is:

- In Ruby release 3.warn (around Ruby 2.7 EOL, maybe 3.3?), warn for every usage of ruby2_keywords, mentioning it should be replaced by *args, **kwargs-delegation (or ..., but that's severely restricted currently: #16378). *args, **kwargs-delegation is only correct in Ruby 3.0+ so at that point Ruby 2.x support needs to be dropped, or a version check be used.
- In Ruby release 3.clean (that is 3.(warn+1), maybe 3.4?), remove ruby2_keywords. At that point, the separation of positional and keyword arguments is finally achieved. foo(*args) will always mean "pass only positional arguments". Everytime keyword arguments are passed it will be explicit (foo(**kwargs) or foo(key: value)), no more magic and a clean separation.

So no matter what, to get the clean separation we'll have to wait many (5?) years for Ruby 3.clean, and delegation code will need to change in 3.warn.

But right now, we broke delegation in 2.7 and require to add ruby2_keywords (which means **changing twice delegation code** in this period) for seemingly little to no benefit.

My proposition is to simply use ruby2_keywords semantics for all methods and blocks in Ruby 2.7 (and until version 3.warn). This would be compatible with Ruby 2.6 and before.
This means, no explicit ruby2_keywords anywhere, no need to change anything for delegation to work in Ruby 2.7, 3.0, ... until Ruby 3.clean.

Importantly, it means **only change delegation code once** and **Ruby 2.0 until Ruby.warn keep *args-delegation compatible and working**.

The semantics of that are (same as if ruby2_keywords was applied to all methods):

- When passing keyword arguments syntactically (using either foo(**kwargs) or foo(key: value)) to a method not accepting keyword arguments (e.g., def m(*args)), flag the keyword arguments Hash as "keyword arguments".
- Whenever calling a method with a *rest argument and no keyword arguments (e.g., foo(*args)), if the last argument is flagged as "keyword arguments", pass them as keyword arguments. If the called method doesn't accept keyword arguments, pass the Hash as positional argument and keep the "keyword arguments" flag.

That way, code like this just keeps working:

```
def target(*args, **kwargs)
  [args, kwargs]
end

def delegate(*args, &block)
  target(*args, &block)
end

target(1, b: 2) # => [[1], {b: 2}] in Ruby 2 & 3
delegate(1, b: 2)
# => [[1], {b: 2}] in Ruby 2 & 3, no warning in 2.7 because {b: 2} is passed as keyword arguments
to target
```

And also if args is stored somewhere or delegated multiple levels down.

Do we lose anything by not marking delegation methods with ruby2_keywords?
I think we lose nothing, and we gain a lot (compatibility and avoiding needless ugly changes).
In Ruby 3.warn we can easily warn for every case that passes keyword arguments using foo(*args) and even have a debug mode telling where the Hash was flagged as a "keyword Hash".

Thoughts?
Should we fix delegation in Ruby 2.7 .. Ruby 3.warn so it works again and not needlessly break Ruby code? I believe YES!

PR: https://github.com/ruby/ruby/pull/2853

P.S.: I actually proposed this idea on the ruby-core Slack on 13th December, but got just one response from jeremyevans0 (Jeremy Evans):

> Me: If we applied ruby2_keywords automatically on all methods, would *args-delegation just keep working in 2.7 and later? I think the fundamental issue with kwargs changes is that we break *args by changing its meaning, in a way it no longer works to delegate "all arguments except block". Probably almost every method that takes (*args) and then call some methods with *args intents to pass positional and kwargs as-is, no matter the Ruby version. If we could save this pattern we'd make the transition much nicer.
> Jeremy: I worked on a branch with ruby2_keywords behavior by default (for all methods taking *args, not just those that delegate *args inside the method: https://github.com/jeremyevans/ruby/tree/ruby2_keywords-by-default . I don't recommend that approach, as it is much more likely to result in a keyword-flag hashed being created to a method where the hash should be treated as positional.
> Me: Does it matter if the Hash is flagged and passed to a method not taking kwargs? It would still be the same behavior, no?
> Jeremy: You can end up with the hash being passed as keywords when you expect it to be passed as non-keywords. It's not safe in general unless you know the method will be used for argument delegation.

Jeremy's concern is sometimes you might want foo(*args), with args[-1] a Hash with a "keyword arguments" flag, to pass as positional to def foo(*args, **kwargs).
However, that seems extremely unlikely to me, and not worth breaking delegation in Ruby 2.7.
To have the "keyword arguments" flag, the Hash must have been passed originally as keyword arguments. It sounds unlikely you would then want to pass it as positional to a method taking keyword arguments.
If you do want that, it's always possible to do foo(*args, **{}), which also works in Ruby 2.6 (and before).

Related issues:

Related to Ruby master - Bug #16473: New deprecated warning disallows keyword...	**Rejected**
Related to Ruby master - Feature #16511: Staged warnings and better compatibi...	**Open**
Related to Ruby master - Bug #16466: `*args -> *args` delegation should be wa...	**Closed**
Related to Ruby master - Feature #16378: Support leading arguments together w...	**Closed**
Related to Ruby master - Misc #16188: What are the performance implications o...	**Open**
Related to Ruby master - Feature #16897: General purpose memoizer in Ruby 3 w...	**Open**

History

#1 - 12/28/2019 12:35 PM - Eregon (Benoit Daloze)

- Description updated

#2 - 12/28/2019 12:35 PM - Eregon (Benoit Daloze)

*- Subject changed from Fixing delegation in Ruby 2.7: ruby2_keywords semantics by default in 2.7.1 to Fixing *args-delegation in Ruby 2.7: ruby2_keywords semantics by default in 2.7.1*

#3 - 12/28/2019 12:42 PM - Eregon (Benoit Daloze)

Another way to look at this, do we really want the Ruby language between versions 2.7 and 3.warn to have no natural (syntactic) way to do delegation?
(and instead have to use the ugly ruby2_keywords workaround)
That seems so lame to me, let's fix it.

#4 - 12/28/2019 12:47 PM - Eregon (Benoit Daloze)

- Description updated

#5 - 12/28/2019 01:23 PM - Eregon (Benoit Daloze)

- Description updated

#6 - 12/28/2019 01:26 PM - shevegen (Robert A. Heiler)

- Description updated

Sorry for the semi off-topic comment. Personally I switched back to ruby 2.6.5p114 for the time
being, mostly due to my own laziness. I do not use keyword arguments in my own code base, but
I had several warnings; a few in FileUtils, but more importantly some in other (external) gems
over which I have no control (though of course I could silence them all via $VERBOSE and
re-enabling it after loading). The change in keywords was actually the only one that was
frustrating me a bit - this may be due to psychological effects only, since warnings are
not the same as errors, but still.

I'll wait until other folks fix their ruby code first. :D (I have 2.8 available too, to
slowly test stuff at a later point, but right now I am just in waiting mode. So this comment,
while not that helpful to benoit's issue request, may be indirectly helpful to assess a
tiny bit of the impact. Note that I still agree with making keyword behaviour consistent,
but it was indeed a bit frustrating to see lots of warnings emerge when using the latest
ruby. I will most likely re-evaluate in some weeks or a few months of course. I also read
that the rails code is also being adjusted to the latest ruby - it may be interesting to
hear what they have to say, if they have to say something that is)

#7 - 12/28/2019 01:30 PM - Eregon (Benoit Daloze)

- Description updated

#8 - 12/28/2019 03:59 PM - rafaelfranca (Rafael França)

This solution would be my first choice if for some reason Rails needs to ask users to change their code to use ruby2_keywords.

In our current problem, users define Mailers and those mailers have actions, that are public methods in the mailer class. The way Rails dispatch the actions uses a few levels of delegations and all those delegations can pass keyword arguments. If it was not possible to remove the warnings without asking users to mark their actions using ruby2_keywords I was going to use the method_added hook on the mailer base class to automatically call ruby2_keywords in all public methods of a mailer.

I'm not sure yet if that solution is needed at all, but I think that by marking all methods with ruby2_keywords in 2.7 we are delaying the warnings and all code changes to Ruby 3. Maybe that is a good thing and more inline with the stability of Ruby releases but, as a library maintainer, I don't mind a rocky path in order to arrive to a good destination. And, I prefer to do that sooner than later.

We are being pushing this style of changes in Rails for a while. If the Rails 4 to Rails 5 upgrade path taught me something is that the more you delay those breaking change, the more people will delay fixing them. It took us more than 1 year to upgrade the Shopify application from Rails 4.2 to Rails 5.0 because we deprecated behavior in Rails 4.0 but only removed it in Rails 5.0. Instead of fixing the deprecated behavior through the 3 years between Rails 4.0 and 5.0 we left everything to be done only when it was needed and that had a huge price.

I think by delaying the warnings we will fall in the same trap we did with Rails 5.0. People will only fix the warnings after Ruby 3 is released. If we think that Ruby 3 will have more changes that will require user to change their code, we are just piling up more work to the users at the same time, what will delay their upgrades and give the impression that upgrading to Ruby 3 is too hard. If we split the path to upgrade to Ruby 3 in many bumps in our road, it will be still bumpy, but it will at least be passable.

I'm being very vocal against this change not because it is annoying, but because it is without precedents, at least while I'm in the community, and incoherent with other decisions in the same release. I don't remember any Ruby upgrade since Ruby 1.9 that caused so much trouble to application and library developers, and I didn't expected this kind of decision from the Ruby Core team, given their track record of stability. But, I'm used to this kind of decision, since I'm taking decisions like this in the Rails Core team for years, and I'm happy we are choosing to go by this path now.

In my opinion, if a bumpy road to a amazing destination is what we want, we should be coherent. In the same release we pushed the community to change a lot of code to add ruby2_keywords for the sake of consistency we also reverted a performance related change because a few gems would need to be updated. Not to say about the frozen_string_literal comments that we are pushing down to the community for a while and now that we were arriving closer to the date where finally we could drop that comment we just made it obligatory for maybe more 10 years.

In summary, I prefer if we make all changes necessary to the keyword arguments split sooner than later but that is coming from someone with a different opinion about stability when compared with the Ruby Core.

#9 - 12/28/2019 04:15 PM - Eregon (Benoit Daloze)

I should be more precise about "*args-delegation broke in Ruby 2.7 for keyword arguments.".

For example, let's take this simple example from https://eregon.me/blog/2019/11/10/the-delegation-challenge-of-ruby27.html:

```
def target(*args, **kwargs)
  [args, kwargs]
end

def delegate(*args, &block)
  target(*args, &block)
end

target(1, b: 2) # => [[1], {b: 2}] in Ruby 2 & 3

delegate(1, b: 2) # => [[1], {b: 2}] in Ruby <= 2.6

# Ruby 2.7:
# kwargs.rb:6: warning: Using the last argument as keyword parameters is deprecated; maybe ** should be added
to the call
# kwargs.rb:1: warning: The called method `target' is defined here
# => [[1], {:b=>2}]

# Ruby 3.0-3.warn: [[1], {b: 2}] if using `ruby2_keywords` (explicitly or by default); Otherwise [[1, {:b=>2}]
, {}] (breaking)

# Ruby 3.clean+: [[1, {:b=>2}], {}] in Ruby 3
```

If we use ruby2_keywords semantics by default, it works fine until Ruby 3.warn, at which points it warns to change to *args, **kwargs-delegation, which works for Ruby 3.0+.
Without ruby2_keywords (explicit or by default) it would break in 3.0.
Without ruby2_keywords it returns the correct result but produces a confusing warning in Ruby 2.7.

Based on this confusing warning in Ruby 2.7, people are likely to think they need to pass **kwargs explicitly like:

```
def delegate(*args, **kwargs, &block)
  target(*args, **kwargs, &block)
end
```

Which would be the natural and intuitive thing to do (who can blame them?), and would actually work in Ruby 3.0+.
At that point though, delegation actually breaks when delegating to a method not taking keyword arguments such as def target(*args); args; end.
In Ruby 2.6, **kwargs inside delegate passes an extra positional {} if no keyword arguments are passed: delegate() returns [{}] while target() returns [].
In Ruby 2.7, delegate({}) returns [], dropping the positional {}, while a direct target({}) call would correctly return [{}].

If we use ruby2_keywords semantics by default, we avoid this very complicated corner cases, **avoid these confusing warnings**, and we **only require people to change their delegation code when they can actually migrate** (i.e., drop Ruby 2.x support) to use *args, **kwargs-delegation.
If they want to support both Ruby 3.clean and Ruby 2.x (e.g., RSpec might) it's possible with a simple version check (like the first solution in the blog post).

In a simplified way, *args-delegation would work just fine in Ruby 2.0 - 3.warn, and then change to the intuitive *args, **kwargs delegation in Ruby 3.clean.
No confusing and full of weird corner cases state in between.

#10 - 12/28/2019 04:40 PM - Eregon (Benoit Daloze)

rafaelfranca (Rafael França) wrote:

> This solution would be my first choice if for some reason Rails needs to ask users to change their code to use ruby2_keywords.

Nice to hear.
I wonder if Rails and other gems really need to go through adding ruby2_keywords which obviously is not trivial as seen in that example.
My impression in this issue is explicit ruby2_keywords gains little to nothing, but requires lots of changes in gems to fix the warnings.

> I'm not sure yet if that solution is needed at all, but I think that by marking all methods with ruby2_keywords in 2.7 we are delaying the warnings and all code changes to Ruby 3.

The tricky bit here is the way recommended in the official blog post, that is adding ruby2_keywords for all delegating methods, will break in Ruby 3.clean.
And at the same time, there is no easy way to prepare for it, for instance with a version check because neither *args-delegation nor *args, **kwargs-delegation works in Ruby 2.7.0.
*args, **kwargs-delegation will work in Ruby 3.0+ but it doesn't work correctly in Ruby <= 2.7.

With my proposition, we could simplify this whole mess and keep delegation code as it is until Ruby 3.warn.
And then change to *args, **kwargs-delegation if Ruby 2.x support is not needed (should be very likely at that time, that's why the plan is to do this after Ruby 2.7 EOL), or to a version check approach otherwise, like:

```
if RUBY_VERSION < "3"
  # To work with Ruby 2.7.0, this needs an explicit ruby2_keywords
  def delegate(*args, &block)
    target(*args, &block)
  end
else
  def delegate(*args, **kwargs, &block)
    target(*args, **kwargs, &block)
  end
end
```

> I think by delaying the warnings we will fall in the same trap we did with Rails 5.0. People will only fix the warnings after Ruby 3 is released. If we think that Ruby 3 will have more changes that will require user to change their code, we are just piling up more work to the users at the same time, what will delay their upgrades and give the impression that upgrading to Ruby 3 is too hard. If we split the path to upgrade to Ruby 3 in many bumps in our road, it will be still bumpy, but it will at least be passable.

The problem here is *args, **kwargs-delegation cannot be used alone as long as Ruby 2.x needs to be supported.
So it's needed to be a 2 phases deprecation if we want to avoid many version checks in library code:

1. Warnings about calling methods accepting keyword arguments without ** or key: value, in Ruby 2.7.
2. Warnings about delegation changing from *args to *args, **kwargs in Ruby 3.warn.

But with the 2.7.0 release we actually get unhelpful warnings for delegation, so basically a third not needed step, already in 2.7 while we can't actually use Ruby 3-style delegation yet without breaking Ruby 2.x support.

> In summary, I prefer if we make all changes necessary to the keyword arguments split sooner than later but that is coming from someone with a different opinion about stability when compared with the Ruby Core.

I argued for that, which would mean removing ruby2_keywords in Ruby 3.0.
The rest of ruby core seemed to disagree because it would mean a lot of version checks around delegation methods (to keep supporting Ruby 2.x).

#11 - 12/29/2019 12:05 AM - mame (Yusuke Endoh)

I'm against this change unless many real-world, difficult-to-avoid problems are reported.

If I understand Rafael correctly, Rails issue seems to be avoidable by automatical ruby2_keywords in method_added. It is not clean, but may be a good idea. User-defined methods should be finally updated (from def foo(*args) to def foo(*args, **kwargs)) after ruby2_keywords is deprecated, but only one change is required in the user side.

And, ruby2_keywords by default is not a silver bullet. If a method is intended to accept only positional arguments, and if a keyword is passed to the method, it must be warned. ruby2_keywords by default hides the appropriate warning. A user will miss the code that should be fixed. In addition, if the flagged hash is unintentionally leaked, the would cause very time-consuming issue: "What's happen!?" -> "Why is this hash passed as keywords!?" -> "Where is this flagged hash created!?". (Of course ruby2_keywords is possible to cause the same issue, but the possibility is much lower than "by default".)

#12 - 12/29/2019 03:01 AM - jeremyevans0 (Jeremy Evans)

mame (Yusuke Endoh) wrote:

> I'm against this change unless many real-world, difficult-to-avoid problems are reported.

Agreed. Further, so far, not that many problems have been reported, and the problems reported are not difficult to avoid.

> If I understand Rafael correctly, Rails issue seems to be avoidable by automatical ruby2_keywords in method_added. It is not clean, but may be a good idea. User-defined methods should be finally updated (from def foo(*args) to def foo(*args, **kwargs)) after ruby2_keywords is deprecated, but only one change is required in the user side.

Using method_added doesn't even appear to be necessary. In Rafael's Rails issue, all you need is ruby2_keywords in a few places in framework code, and no changes to user code, see https://github.com/rails/rails/pull/38105#discussion_r361803111

> And, ruby2_keywords by default is not a silver bullet. If a method is intended to accept only positional arguments, and if a keyword is passed to the method, it must be warned.

To clarify, this is only the case if the arguments are later passed to another method that uses keyword arguments. Calling a method that doesn't explicitly accept keywords with keywords is fine, the keywords are just treated as a positional hash in this case.

> ruby2_keywords by default hides the appropriate warning. A user will miss the code that should be fixed. In addition, if the flagged hash is unintentionally leaked, the would cause very time-consuming issue: "What's happen!?" -> "Why is this hash passed as keywords!?" -> "Where is this flagged hash created!?". (Of course ruby2_keywords is possible to cause the same issue, but the possibility is much lower than "by default".)

I agree. I worked on a branch with ruby2_keywords by default before 2.7.0 was released. I consider that approach too risky. ruby2_keywords can cause significant problems if used incorrectly. I think it is only safe to enable it in cases where you know the usage is correct, which is basically the ruby 2.7.0 behavior.

#13 - 12/29/2019 11:34 AM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote:

> I'm against this change unless many real-world, difficult-to-avoid problems are reported.

Here is an example: https://github.com/rails/rails/pull/38105#discussion_r361842686
I think nobody is able to explain it fully right now, and it shows that it's basically impossible to correctly know where ruby2_keywords is needed without guessing.

In fact, both you and Jeremy, which I think know the best about ruby2_keywords can't fully explain where it's needed.
What about people which weren't involved in so many discussions about keyword arguments and ruby2_keywords?
They simply have no chance and have to guess.
I think it shows that my proposal is needed if we want people to use Ruby 2.7 at all (I think many will just give up otherwise, or just disable all warnings).

> If I understand Rafael correctly, Rails issue seems to be avoidable by automatical ruby2_keywords in method_added. It is not clean, but may be a good idea. User-defined methods should be finally updated (from def foo(*args) to def foo(*args, **kwargs)) after ruby2_keywords is deprecated, but only one change is required in the user side.

For the particular case above, method_added is not needed.
Using method_added is basically acknowledging we have no idea where to add ruby2_keywords, i.e. it's a sign we want to enable it for all methods.

> If a method is intended to accept only positional arguments, and if a keyword is passed to the method, it must be warned.

Incorrect. Passing keyword arguments to a method taking only positional arguments just passes the keywords arguments as a final positional Hash.
So it's completely harmless in such a case.

> ruby2_keywords by default hides the appropriate warning.

No, I believe it never does.

> A user will miss the code that should be fixed.

No, it only avoids false-positive warnings in delegation.
It does not remove other legit warnings.
And delegation code shouldn't need to change now, it can only be changed to *args, **kwargs-delegation when dropping Ruby 2.x compatibility.

> In addition, if the flagged hash is unintentionally leaked, the would cause very time-consuming issue: "What's happen!?" -> "Why is this hash passed as keywords!?" -> "Where is this flagged hash created!?". (Of course ruby2_keywords is possible to cause the same issue, but the possibility is much lower than "by default".)

Actually it simplifies the reasoning, because all keyword arguments Hash are flagged.
And as mentioned we can add a debug mode to know things like when the Hash was flagged and when it's passed as keyword arguments.
We can already add that in 2.7, and it will be needed for Ruby 3.warn to migrate to *args, **kwargs-delegation.

As I already mentioned in the original description, the only case where ruby2_keywords semantics might be not wanted is
if you want foo(*args), with args[-1] a Hash with a "keyword arguments" flag (so it was keyword arguments originally), to pass as positional to def foo(*args, **kwargs).
For that, you need foo(*args, **{}) in Ruby 2.6 anyway. So it just makes sense to use it in Ruby 2.7 as well.

Which I think means, there is exactly 0 case where ruby2_keywords semantics are not wanted.

Can anyone show a counter-example? And if there is, I'd expect it would be <1% of the cases.

#14 - 12/29/2019 03:46 PM - mame (Yusuke Endoh)

Eregon (Benoit Daloze) wrote:

> > If a method is intended to accept only positional arguments, and if a keyword is passed to the method, it must be warned.
>
> Incorrect. Passing keyword arguments to a method taking only positional arguments just passes the keywords arguments as a final positional Hash.
> So it's completely harmless in such a case.

Oops, sorry. I was confused again.

Currently, we have only one unsettled issue yet. And I think that there is a false negative bug in the warning mechanism ([#16466](#16466)), which are making the issue complicated. Before we go to extremes, how about focusing on the Rails issue?

#15 - 12/29/2019 05:00 PM - Eregon (Benoit Daloze)

FWIW, I tried to rebase Jeremy's branch on top of 2.7.0:
https://github.com/ruby/ruby/compare/v2_7_0...eregon:ruby2_keywords-by-default
There were a few conflicts though, I'm not sure it's correct.

All specs but one pass. test-all passes except test_keyword.rb.
The difference there seems to be that def m(*a) a end; m(**{}) gives [{}] with ruby2_keywords, and [] without.
That seems a separate workaround that was included in Module#ruby2_keywords but I think it doesn't need to be together.

Unfortunately, that still warns for the Rails case. I suspect a bug somewhere else, maybe an incorrect warning from MRI?

#16 - 12/29/2019 06:39 PM - jeremyevans0 (Jeremy Evans)

Eregon (Benoit Daloze) wrote:

> Which I think means, there is exactly 0 case where ruby2_keywords semantics are not wanted.
> Can anyone show a counter-example? And if there is, I'd expect it would be <1% of the cases.

```
def foo(x, y: $stderr)
  y.puts x.inspect
end

def bar(*args)
  args.each do |arg|
    foo_args = [arg]
    foo(*foo_args)
  end
end

bar(2, y: 1)
```

Basically, this is any place where you have a method that takes an arbitrary number of arguments and later use each argument in a separate splat call and want it to be treated positionally. This code works correctly in 2.6, 2.7, and 3.0. It will warn in 2.7 and fail in 3.0 if ruby2_keywords is made the default behavior.

If we want to do this, we should at least add a method to remove the ruby2_keywords flag in the cases where it causes problems.

#17 - 12/29/2019 07:32 PM - Eregon (Benoit Daloze)

jeremyevans0 (Jeremy Evans) wrote:

> If we want to do this, we should at least add a method to remove the ruby2_keywords flag in the cases where it causes problems.

Thank you for the example.

I'm not sure we need that, it's very easy to make this example work with either:

- bar(2, {y: 1}) which clearly shows we want to pass positional arguments to that method, and I think makes perfect sense. It could also be written as bar(2, h), bar(2, **h) would be clearly wrong here. Passing as positional to a method taking positional arguments is nicely consistent here.
- foo(*foo_args, **{}) which is an easy workaround if needed, but it's probably almost never needed.
- foo(arg) which shows the example is IMHO quite contrived. If you're going to iterate *args and pass them one by one, then you're almost always going to pass them as foo(arg) or foo(arg, <some extra args>), which both work fine.

#18 - 12/29/2019 07:59 PM - jeremyevans0 (Jeremy Evans)

One thing to consider before deciding whether this behavior should be the default is how to deprecate it if we decide to remove it later. With ruby2_keywords, we would just warn when ruby2_keywords is called. Without this, we would probably also have to warn on all cases where a keyword flagged hash is treated as keywords. If we agree to keep this behavior forever, I suppose we don't need to worry about it, though.

I think Eregon (Benoit Daloze) is correct that ruby2_keywords by default will be correct in most cases, and will reduce the amount of changes Ruby developers will need to make to their libraries. Still, I think such magical behavior is likely to introduce difficult to diagnose breakage in some cases, with no obvious reason why the code is not working correctly. For that reason I remain against this.

#19 - 12/30/2019 03:57 PM - Eregon (Benoit Daloze)

jeremyevans0 (Jeremy Evans) wrote:

> Without this, we would probably also have to warn on all cases where a keyword flagged hash is treated as keywords.

Yes, I think that warning should be just as precise as an explicit ruby2_keywords, and guide people to migrate to *args, **kwargs-delegation incrementally (from deepest callee back to caller).

As I mentioned in the summary:

> In Ruby 3.warn we can easily warn for every case that passes keyword arguments using foo(*args) and even have a debug mode telling where the Hash was flagged as a "keyword Hash".

Which I think would help significantly with debugging and migration. We can already have that in earlier versions too.

> If we agree to keep this behavior forever, I suppose we don't need to worry about it, though.

I'm definitely against that, we would be left with only a partial keyword arguments separation, which would question the whole effort IMHO.
I think we want to go to clear and simple semantics, and for that that I think we need to separate positional and keyword arguments cleanly in the longer term (Ruby 3.clean).

#20 - 12/30/2019 04:20 PM - Dan0042 (Daniel DeLorme)

Wow, we've finally reached the point where even Jeremy thinks ruby2_keywords by default *might* be the better choice. I didn't think I'd see that. Maybe just one more push and we can reach the point where the hidden keyword hash flag can be made explicit via a subclass of Hash, and then we'd have a nice clean object-oriented design like I attempted to describe before.

#21 - 12/30/2019 04:34 PM - jeremyevans0 (Jeremy Evans)

Dan0042 (Daniel DeLorme) wrote:

> Wow, we've finally reached the point where even Jeremy thinks ruby2_keywords by default *might* be the better choice.

That is not what I wrote. You either did not read or understand what I wrote or you are purposely misrepresenting it. Let me repeat the end of my comment: "I remain against this". I am definitely not in favor this. Seeing possible benefits of an approach does not mean you believe the benefits outweigh the costs.

#22 - 12/30/2019 06:23 PM - Dan0042 (Daniel DeLorme)

My apologies, it seems I overestimated the meaning of "I think Eregon is correct". And perhaps you missed the part where I emphasized "might", as I never meant to imply you were 100% behind the idea. So hard to convey nuances in writing online :-/

#23 - 01/01/2020 04:40 PM - Eregon (Benoit Daloze)

I think one of the main advantages of doing this is a much better migration path.

Imagine I have this code, which works fine in Ruby 2.6:

```
def foo(*args)
  bar(*args)
end

def bar(*args)
  cdr(*args)
end

def cdr(*args)
  target(*args)
end

def target(**kwargs)
```

```
  kwargs
end

p foo(a: 1)
```

If I run it on Ruby 2.7.0, it still works but warns:

```
migration.rb:10: warning: Using the last argument as keyword parameters is deprecated; maybe ** should be adde
d to the call
migration.rb:13: warning: The called method `target' is defined here
{:a=>1}
```

Now what should I do about this? If I know about ruby2_keywords,
I might guess it's needed on line 9, because the warning is at line 10.
So I go ahead and add ruby2_keywords on line 9, rerun the program and see:

```
migration.rb:10: warning: Using the last argument as keyword parameters is deprecated; maybe ** should be adde
d to the call
migration.rb:13: warning: The called method `target' is defined here
{:a=>1}
```

That is, the exact same warning.
No indication I solved anything, no indication of what to fix next.
Of course, in such a trivial example it might be easy to see where to add ruby2_keywords, but it's clearly far more difficult on any non-trivial gem where not all calls are in the same files, polymorphism and super calls are often used, etc (e.g., Rails and any other non-trivial gem).
puts caller might helper in some situation, but fails to follow data dependencies.

Warnings should give a clear way to fix them. Ruby 2.7.0 is failing at this for the delegation case.

If we apply my proposition, ruby2_keywords by default:

- This script works fine without warning in Ruby 2.7.1
- It will warn in Ruby 3.warn, where we can actually do something about it.

At that point, we can actually rewrite delegation to use *args, **kwargs which is both natural, consistent with the separation and intuitive.
(we cannot use *args, **kwargs in Ruby 2.7 as that's incorrect/broken in both 2.6 and 2.7)

When we rewrite cdr to:

```
def cdr(*args, **kwargs)
  target(*args, **kwargs)
end
```

and rerun the program, it will give us what to do next:

```
migration.rb:6: warning: Using the last argument as keyword parameters is deprecated; delegate explicitly usin
g *args, **kwargs
migration.rb:9: warning: The called method `cdr' is defined here
{:a=>1}
```

Which means next is adding , **kwargs to bar. And then it will tell us about foo and the program is fixed.

I think just for the migration it's worth to apply my change.
Right now, the user experience to migrate delegation in Ruby 2.7 is frustrating, produces unhelpful warnings, and confuses many people.
It creates a third way to delegate (*args in 2.6, explicit ruby2_keywords in 2.7, *args, **kwargs in Ruby 3.warn+) which implies changing a lot of code, for little to no benefit and actually significant drawbacks.

#24 - 01/01/2020 04:56 PM - Eregon (Benoit Daloze)

Dan0042 (Daniel DeLorme) wrote:

> Maybe just one more push and we can reach the point where the hidden keyword hash flag can be made explicit via a subclass of Hash, and then we'd have a nice clean object-oriented design like I attempted to describe before.

Representing the Hash flag as a subclass of Hash sounds nice for debugging.
I think in that comment it got conflated with other problems, but seen as a replacement for the existing flag on Hash it sounds a good idea to me. Both literal: keyword and **h would create a KwHash.
A subclass would prevent changing the flag for an existing Hash, but I don't think we need that currently.

I don't think we'd want foo(someNonLiteralKwhash) to call with keyword arguments though, that would be very expensive to check on every method call.

In the Rails issue, it became very clear we need:

- An easy way to know if a Hash is flagged as "keyword arguments" (part of Hash#inspect?, a new Hash method?, the subclass?)
- A way to find out where that Hash was created, which should give us the file and line number in the user code.
- A way to flag a Hash, but this is easy enough to workaround by defining a helper method (marked with ruby2_keywords if not the default) taking *args, returning args.last and calling it with kwargs = helper(**hash).

#25 - 01/02/2020 11:28 PM - Eregon (Benoit Daloze)

- Related to Bug #16473: New deprecated warning disallows keyword arguments bypassing added

#26 - 01/03/2020 03:01 AM - Dan0042 (Daniel DeLorme)

> Representing the Hash flag as a subclass of Hash sounds nice for debugging.
> I think in that comment it got conflated with other problems, but seen as a replacement for the existing flag on Hash it sounds a good idea to me.
> Both literal: keyword and **h would create a KwHash.

I must admit I've been struggling with the conflation issue. I think even with the current design it would be nice if the double-splat created a KwHash. I can think of a way to optimize keyword arguments but it would require a different implementation for the keyword hashes, and it's much easier/cleaner to have a different implementation if it's tied to a different class.

At the same time (but orthogonally) a subclass would allow an approach with measurably better backward compatibility, but I realize this comes down to "compatibility vs clean design" tradeoffs and not everyone has the same beliefs and priorities. ruby2_keywords by default would achieve roughly the same effect by postponing this part of the migration (ideally until ruby 2.6 is EOL).

> I don't think we'd want foo(someNonLiteralKwhash) to call with keyword arguments though, that would be very expensive to check on every method call.

I think not any more expensive than in ruby 2.6 checking if the last positional argument is a Hash.

#27 - 01/14/2020 06:45 AM - mame (Yusuke Endoh)

Hi, I talked about this ticket with ko1, nobu, and znz before the dev-meeting. After the discussion, I am still against this.

We premise that the ruby2_keywords flag is never a wonderful thing. We want to delete it in the future. Module#ruby2_keywords will serve as a mark showing "we need to change this method definition so to accept not only positional rest arguments but also keyword ones explicitly if ruby2_keywords flag is deleted."

So, assuming the flag is removed at Ruby 4.0, we'd like users to change their code in the following style:

```
# 2.6
def foo(*args)
  bar(*args)
end

# 2.7 .. 4.0 (until ruby2_keywords flag is deprecated)
ruby2_keywords def foo(*args)
  bar(*args)
end

# 4.0 ..
def foo(*args, **kwargs)
  bar(*args, **kwargs)
end

# or, if possible
def foo(...)
  bar(...)
end
```

If we set ruby2_keywords by default, users cannot identify where to fix, and their code will break suddenly after the flag is removed. It would be unacceptable and we will be unable to deprecate Module#ruby2_keywords forever. This is not what we want.

#28 - 01/14/2020 11:14 AM - Eregon (Benoit Daloze)

mame (Yusuke Endoh) wrote:

> If we set ruby2_keywords by default, users cannot identify where to fix, and their code will break suddenly after the flag is removed. It would be unacceptable and we will be unable to deprecate Module#ruby2_keywords forever. This is not what we want.

Did you see https://bugs.ruby-lang.org/issues/16463#note-23?
It concludes the opposite of what you wrote above.

That explains:

- The current way (2.7.0) to add ruby2_keywords is extremely difficult and there is no help from the warnings. The warnings just give a single location and nothing about all the other places needing ruby2_keywords.
- It would be very easy to produce accurate warnings with ruby2_keywords by default, which would make the transition a lot easier.

Therefore I believe migration is very significantly easier with this proposal.

#29 - 01/15/2020 05:10 PM - Eregon (Benoit Daloze)

Here are slides I made to explain this proposal more visually:
https://docs.google.com/presentation/d/1J6voqHFQ46-MsEm_vUJsBJiktNvoA6niz3fea9awmco/edit?usp=sharing

I think it's clear there that migration of delegation is so much better with this proposal.

#30 - 01/15/2020 07:15 PM - decuplet (Nikita Shilnikov)

Side note. One particular technique I used to deal with warnings is the gem 'warning' made by Jeremy some time ago and the following code in spec_helper.rb:

```
Warning.ingore(/filter-warnings-from-other-gems-with-a-regexp/)
Warning.process { |warning| raise warning }
```

When a warning is shown, it throws an exception with the full backtrace. This way it's easy to catch warnings, it just takes some time, depending on the size of the project.
I also want to state I understand the change was necessary. It was quite a bit of work, but it was mostly about fixing the dry-rb and rom-rb gems (there are more than twenty of them in total). Updating application code was a lot less painful, partially because applications normally work with a single ruby version.

Eregon (Benoit Daloze), to clarify, will your proposal affect already migrated code in any way?

#31 - 01/15/2020 07:58 PM - Eregon (Benoit Daloze)

decuplet (Nikita Shilnikov) wrote:

> I also want to state I understand the change was necessary.

Making delegation warn is actually not necessary in 2.7.
My proposal would warn in later Ruby versions when we can actually use *args, **kwargs-delegation (and drop Ruby 2.x support).

> It was quite a bit of work, but it was mostly about fixing the dry-rb and rom-rb gems (there are more than twenty of them in total).

How did you fix delegation cases? Using ruby2_keywords?
Was it easy to find where to add ruby2_keywords?
Due to [Bug #16466] you might need more ruby2_keywords calls.

> Eregon (Benoit Daloze), to clarify, will your proposal affect already migrated code in any way?

It would make ruby2_keywords a no-op.
I think it should warn since it would be useless, not pretty and confusing as to whether it serves some purpose.

#32 - 01/15/2020 08:34 PM - decuplet (Nikita Shilnikov)

> How did you fix delegation cases? Using ruby2_keywords?

Yes, mostly.

> Was it easy to find where to add ruby2_keywords?

I showed the trick. I didn't have major problems with fixing warnings except for one gem. It led me to think that having warnings is ultimately a good thing. Eventually, keywords will be separated, it's better to decide now which methods should accept keywords and which should work with positional ones. Design-wise.

> I think it should warn since it would be useless, not pretty and confusing as to whether it serves some purpose.

Changing behavior in patch versions is a tricky business. I can only speak for myself but I started updating gems in November, the work was almost done by the release of 2.7. Now I'll have to go over the codebase once again because not all of gem maintainers did so. I understand the code will have to be updated anyway but at a different time and this is important. I had enough time to prepare for 2.7. I'm not so sure about 2.7.1. Don't get me wrong, it's not a complaint, I'm just thinking it through.

I want to add that I deeply appreciate everyone's effort on the subject whichever side they take :) The whole keyword story I mean.

#33 - 01/16/2020 04:36 AM - Dan0042 (Daniel DeLorme)

My alternative proposal to accomplish the same objective: #16511

#34 - 01/16/2020 07:41 AM - Eregon (Benoit Daloze)

- Related to Feature #16511: Staged warnings and better compatibility for keyword arguments in 2.7.1 added

#35 - 01/21/2020 10:53 PM - Eregon (Benoit Daloze)

I opened a PR implementing this change: https://github.com/ruby/ruby/pull/2853
I think it's quite clean and I think it's a huge help to migrate to Ruby 2.7.

It will also be very useful to migrate delegation once Ruby 2 is EOL, as it will make it possible to have accurate warnings such as
warning: Passing keyword arguments with *args is deprecated, use *args, **kwargs for delegation.
Such accurate warnings for delegation are simply not possible in 2.7, since *args, **args-delegation doesn't work in 2.7, so delegation should be migrated after Ruby 2 EOL.

#36 - 01/23/2020 09:57 PM - Eregon (Benoit Daloze)

- Description updated

#37 - 02/18/2020 09:18 PM - Eregon (Benoit Daloze)

- Assignee set to matz (Yukihiro Matsumoto)

#38 - 03/14/2020 11:32 AM - Eregon (Benoit Daloze)

From the meeting log:

Discussion:
akr: This proposal makes incompatible between 2.6 and 2.7(.1). When ruby2_keywords is not used, it should be compatible with 2.6. But if ruby2_keyword is enabled by default, user cannot control the compatibility.
matz: It has too big impact.

Conclusion:
matz: Reject. Will reply.

I don't understand akr (Akira Tanaka)'s point.
In fact, 2.7.1 with this change would be far more compatible with Ruby 2.6 than the current 2.7.0 regarding delegation (which warns and needs explicit ruby2_keyword to make delegation work again without warnings).
It would also mean delegation wouldn't simply break in Ruby 3.0 if people didn't find out how to address the warnings from 2.7.0 (which are hard to address for delegation because they are imprecise, this approach would allow much better warnings for delegation as explained above).
It would also mean people could migrate from *args-delegation to *args, **kwargs-delegation (when they stop supporting Ruby 2.x), without needing an intermediate step to add many ruby2_keywords and remove it a few releases later.

If the concern is sometimes not wanting delegation semantics for *args, that's possible with call(*args, **{}), which seems extremely rarely needed (only known realistic case is pp).

#39 - 03/16/2020 02:58 AM - matz (Yukihiro Matsumoto)

I have investigated this proposal for a long time and concluded we cannot accept the proposal.

The first reason is the impact of the change. Although it works for most of the delegation cases, it also changes the behavior of all rest arguments. It could cause compatibility problems that are hard to predict.
The second reason is the time frame. It is a soft approach but delays migration for years (until 3.3?). The community would be left in the unstable status for a very long time.
The last reason is the migration path. Currently (some) Rails core contributors use the recent master (which raises errors instead of warnings) to detect/fix keyword argument warnings. Warnings do not give them enough information to fix. The errors from the head do. Once we accept the proposal, even 3.0 would warn, not raise errors.

I really liked the idea at the first moment. But we cannot take this way. Sorry.

Matz.

#40 - 03/16/2020 02:58 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Closed

#41 - 03/16/2020 05:45 PM - Dan0042 (Daniel DeLorme)

Benoit, thank you for your efforts on this issue. Too bad about the result but it was worth the shot. Considering matz chose to cancel the deprecation of the flip-flop operator for the sake of compatibility, I'm surprised that for keywords he prefers a hard break. And like you I'm confused with what akr and matz are saying; this approach may cause a few problems but the notion that this could cause compatibility problems is clearly the opposite of reality; to the point that I wonder if there's some weird Japanese/English misunderstanding/mistranslation going on with the meaning of "compatibility problems".

#42 - 03/16/2020 08:33 PM - Eregon (Benoit Daloze)

Thanks for explaining your thoughts in details.

> The last reason is the migration path. Currently (some) Rails core contributors use the recent master (which raises errors instead of warnings) to detect/fix keyword argument warnings. Warnings do not give them enough information to fix. The errors from the head do. Once we accept the proposal, even 3.0 would warn, not raise errors.

That's a nice acknowledgment that the delegation warnings in 2.7.0 are imprecise, which I argued for a lot in this thread.
This proposal would provide precise warnings, which I believe would be even more helpful to migrate than the hard errors of 3.0.

This approach would delay the first and only migration of delegation code to around 3.4.
With the current warnings/errors though, we enforce changing delegation code twice:

- once in 2.7 using the temporary workaround of ruby2_keywords,
- change again in 3.4 where most likely *args, **kwargs-delegation should be used instead.

So in both cases delegation will be fully migrated in 3.4 (i.e., when people drop support for Ruby 2.x).

With this approach we'd only need to change delegation code once in 3.4, and we wouldn't need temporary changes around all methods delegating keyword arguments.

#43 - 04/02/2020 07:15 PM - Eregon (Benoit Daloze)

- Related to Bug #16466: `*args -> *args` delegation should be warned when the last hash has a `ruby2_keywords` flag added

#44 - 05/28/2020 12:03 PM - Eregon (Benoit Daloze)

- Related to Feature #16378: Support leading arguments together with ... added

#45 - 06/05/2020 03:52 PM - Eregon (Benoit Daloze)

- Related to Misc #16188: What are the performance implications of the new keyword arguments in 2.7 and 3.0? added

#46 - 06/05/2020 03:52 PM - Eregon (Benoit Daloze)

- Related to Feature #16897: General purpose memoizer in Ruby 3 with Ruby 2 performance added