Ruby master - Bug #16154

lib/delegate.rb issues keyword argument warnings

09/09/2019 02:35 AM - jeremyevans0 (Jeremy Evans)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.5: DONTNEED, 2.6: DONTNEED

Description

In the current master branch, lib/delegate.rb (e.g. Delegator/SimpleDelegator/DelegateClass) does not delegate keyword arguments, leading to keyword argument warnings when using it. It is simple to fix this by changing it to delegate keyword arguments, and I have a commit that does this (https://github.com/ruby/ruby/pull/2439/commits/30a70491b11d5235877a13d1a1d3fe0157960a64).

However, one issue with this approach, that you can notice at the bottom of the commit, is that it warns in a case where the behavior will not change between 2.6 and 3.0:

```
foo = Object.new
def foo.bar(*args)
  args
end
d = SimpleDelegator.new(foo)
d.bar({a: 1})
# warns in 2.7, even though 2.6, 2.7, and 3.0 will return [{a: 1}]
```

The reason it warns even though the behavior will not change is that there is a behavior change, but the behavior change is inside the delegator. When you call d.bar, the positional hash is converted to a keyword argument, which generates a warning. When d.bar calls foo.bar, the keyword splat is converted back to a positional hash. Because of this hash->keyword->hash conversion, the net result is behavior does not change outside the delegator, and therefore it would be great if we can somehow flag this particular case as not warning, while still warning in the case where foo.bar would accept keyword arguments (as in that case, behavior will change).

I came up with an implementation that makes delegate not emit the warning in the case where behavior does not change, but still emit the warning in the case where behavior will change (https://github.com/ruby/ruby/pull/2439/commits/6b7bd4f89143d692932dc89fadc341980816ee14). This works by flagging a method that accepts keyword arguments. If the flag is set for the method, and the method is passed a positional hash that is converted to a keyword argument, no warning is emitted, and instead a VM frame flag is set in the next frame so that keyword splats inside that method are converted to positional arguments. The method flag is enabled by calling a Module#pass_positional_hash method, and this method would only exist in 2.7, since it is only for skipping this 2.7-specific warning.

I'm not sure this implementation approach is best, and would be open to any approach that allows delegate to not emit a warning for the code above, as long as it doesn't change the default keyword behavior in 2.7.

Attached is a patch that implements this, and there is also a pull request that has passed CI for it (https://github.com/ruby/ruby/pull/2439).

History

#1 - 09/09/2019 02:22 PM - Dan0042 (Daniel DeLorme)

This is an issue for *all* delegation code. This pass_positional_hash workaround is ok for the stdlib, but what about gems? What if a gem wants to stay compatible with 2.6? This is a question I really don't know the answer to, so I'm asking for an authoritative answer in #16157.

#2 - 09/10/2019 05:34 PM - jeremyevans0 (Jeremy Evans)

As some background for the dev meeting, I tried a couple of different approaches before this that didn't work.

One approach was to flag methods and have all positional hashes be treated as positional hashes (Ruby 3 behavior). That didn't work as it changed the behavior for the case where bar accepts keyword arguments.

Another approach was to flag methods and ignore positional hash to keyword warnings for the method. But in that case, this is no warning for the case where bar accepts keyword arguments, even though that behavior will change in Ruby 3.

I chose to use Module#pass_positional_hash as the way to turn on the flag, as that seemed easiest. I thought about using a per-method magic

comment, but the issue with that approach is it probably doesn't work well for methods defined with define_method.

Instead of using a VM frame flag in the next frame when doing positional hash to keyword argument conversion, we could set an object flag on the keyword argument hash, and check the flag later. That will probably be more challenging to implement, as I think it will require compiler modifications, because when you use **kw, the object passed to the callee is not kw, but a copy of kw. It also raises issues if pass kw to another method as a positional argument, and that method splats it.

I've updated the pull request to document that pass_positional_hash should only be used with methods that do not modify keyword arguments or append positional arguments before delegation. Doing so can result in behavior changes in Ruby 3 without any warning in 2.7. See https://github.com/jeremyevans/ruby/commit/97eeab3d90fe2f560996618c391b68aed4783610#diff-5dbe613f725860801fa4c2c812b1d181.

#3 - 09/11/2019 09:30 PM - jeremyevans0 (Jeremy Evans)

I added an alternative approach for handling delegate keyword warnings in https://github.com/ruby/ruby/pull/2449. This approach also uses a VM frame flag and a Module method (Module#pass_keywords), and allows for passing keywords through a regular argument splat. The advantage of this approach is it should allow for using the same delegation method code on older versions of Ruby without modification, by just marking existing methods that do delegation to have them pass through the keyword arguments.

#4 - 09/19/2019 03:32 PM - jeremyevans0 (Jeremy Evans)

At the dev meeting yesterday, matz recommended changing the method name from pass_keywords to ruby2_keywords. I have updated the pull request to do that: https://github.com/ruby/ruby/pull/2449

akr disliked the approach of using a VM frame flag. I believe it would not be too difficult to change the implementation from using a VM frame flag to using a flag on the final hash object. It's possible you could support that via an option to ruby2_keywords (e.g. ruby2_keywords :method_name, flag: :hash). There are some cases that are handled by a VM frame flag and not handled by a hash object flag (e.g. def foo(*args) args[-1] = args[-1].merge(...) if args[-1].is_a?(Hash); bar(*args) end), and others that are handled by a hash object flag and not a VM frame flag (e.g. def bar(*args) @args = args end; def baz; quux(*@args) end).

#5 - 09/21/2019 08:54 PM - jeremyevans0 (Jeremy Evans)

I worked on an alternative approach of using ruby2_keywords with a hash flag approach instead of a VM frame flag approach: https://github.com/ruby/ruby/pull/2477 . This approach is significantly less invasive to implement, and I think handles more common argument delegation cases than the VM frame flag approach (both approaches handle the simple case). I think we should use the hash flag approach.

There have been discussions about whether to make the ruby2_keywords behavior the default behavior. I do not think it is a good idea. Consider this example:

```
def foo(*args)
  args.each do |arg|
    bar(arg, **{})
  end
end
def bar(*args, **kw)
  baz(*args) unless kw[:skip]
end
def baz(a=1, **kw)
  [h, kw]
end
```

Here foo is designed to take only positional arguments, not to pass through keywords. Let's say you call foo(a: 1). Because the last argument to foo is flagged as keywords, when baz is called, the {a: 1} hash intended to be a positional argument is treated as a keyword argument. Note that baz could be at a completely different place in the program, far from the definition of foo, which will make cases like this difficult to debug.

If you would like to experiment with the approach of passing keywords through argument splats by default, I have a branch that implements it: https://github.com/jeremyevans/ruby/tree/ruby2_keywords-by-default . The only make check failures are in the keyword arguments tests, due to empty hash splats no longer being dropped in the case where a method accepts an argument splat but no keywords.

#6 - 09/23/2019 08:01 PM - Dan0042 (Daniel DeLorme)

When reading the code it seems like the intent of foo(test:42) is for baz to return [{:test=>42}, {}] but that's not what I get when running it on 2.7. I get a warning and [1, {:test=>42}]. To fix the code on 2.7 I need to change baz(*args) to baz(*args, **{}). And at that point, I believe enabling ruby2_keywords by default would not change the result.

Also let's say that such code exists today out there in the wild. It would not use **{} because no one uses that no-op. The result of baz would be [1, {}]. Enabling ruby2_keywords by default would simply preserve the existing behavior without warnings in 2.7, whether that's a good or a bad thing.

What I'm getting at here is that by enabling ruby2_keywords by default we get a behavior that is a bit closer to 2.6. Both the good and the bad. So the separation of keyword arguments is a bit more muddy, but at least it can be said to be backward compatible. It's debatable which is better.

#7 - 09/25/2019 07:35 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed

The hash-flag approach for ruby2_keywords has been merged at 3b302ea8c95d34d5ef072d7e3b326f28a611e479. That commit uses

ruby2_keywords in delegate, fixing the keyword argument separation issues.

Files

delegate-keyword-argument-separation.patch	19.1 KB	09/09/2019	jeremyevans0 (Jeremy Evans)