Ruby master - Feature #16456

Ruby 2.7 argument delegation (...) should be its own kind of parameter in Method#parameters

12/27/2019 12:42 AM - aaronc81 (Aaron Christiansen)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

A method defined with ... as its parameter list is equivalent to one defined with *args, &blk, according to Method#parameters.

```
def foo(...); end
p method(:foo).parameters
# => [[:rest, :*], [:block, :&]]
```

Even in Ruby 2.7, ... and *args, &blk are not *quite* equivalent as the latter may produce a warning where the former does not. In Ruby 3.0 and beyond, ... and *args, &blk will have a substantial semantic difference. Due to this, I don't consider the current behaviour of Method#parameters particularly ideal when dealing with methods using this new syntax.

If the goal of ... is to be a "delegate everything" operator, even when parameter passing is changed like in Ruby 3.0, I would propose that Method#parameters considers it a unique type of parameter. For example:

```
def foo(...); end
p method(:foo).parameters
# => [[:delegate, :"..."]]
```

History

#1 - 12/27/2019 12:43 AM - aaronc81 (Aaron Christiansen)

- Description updated

#2 - 12/27/2019 12:43 AM - aaronc81 (Aaron Christiansen)

- Description updated

#3 - 12/27/2019 11:29 AM - Eregon (Benoit Daloze)

I think it should be:

```
[[:rest, :*], [:keyrest, :**], [:block, :&]]
```

because that's what it will act like in Ruby 3.0+.

Is there an advantage to have its own type of parameter?
That would make usages of #parameters more complex for I think very little gain.

#4 - 12/27/2019 11:37 AM - zverok (Victor Shepelev)

> I think it should be:

```
[[:rest, :*], [:keyrest, :**], [:block, :&]]
```

(I have a deja vu we already discussed it :)))

Names are redundant, it should be just

```
[[:rest], [:keyrest], [:block]]
```

Like this:

```
def foo(*, **, &b)
end
```

```
p method(:foo).parameters
# => [[:rest], [:keyrest], [:block, :b]]
```

(Not sure about "block" parameter -- unnamed block parameters aren't existing in current Ruby)

#5 - 12/27/2019 05:35 PM - aaronc81 (Aaron Christiansen)

> Is there an advantage to have its own type of parameter?

I believe the advantages of doing this are:

- If Ruby ever introduces a new type of parameter, the result of #parameters won't need to change for existing code which uses ..., making upgrades easier. This is especially important if ... is designed as a future-proof way of delegation, as then it seems important that its behaviour shouldn't change between versions.
- It could be useful for introspection to be able to differentiate between the two. For example, this could allow a complex DSL to assign a special meaning to ....

#6 - 12/29/2019 10:54 PM - Dan0042 (Daniel DeLorme)

In the future it will be possible to combine ... with other parameters. So if we think about what parameters would return in cases like these...

```
method(def foo(a, *args, ...); end).parameters
#possibility 1 => [[:req, :a], [:rest, :args], [:delegate]]
#possibility 2 => [[:req, :a], [:rest, :args], [:keyrest], [:block]]
#possibility 3 => [[:req, :a], [:rest, :args], [:rest], [:keyrest], [:block]]

method(def foo(a, **kw, ...); end).parameters
#possibility 1 => [[:req, :a], [:keyrest, :kw], [:delegate]]
#possibility 2 => [[:req, :a], [:keyrest, :kw], [:rest], [:block]]
#possibility 3 => [[:req, :a], [:keyrest, :kw], [:rest], [:keyrest], [:block]]
```

I see the point of wanting to know if the method signature includes ... or not, but I don't think I like the idea of having a :delegate that can mean different things.

What about this?
[[:rest, :"..."], [:keyrest, :"..."], [:block, :"..."]]

#7 - 12/31/2019 05:30 PM - aaronc81 (Aaron Christiansen)

I think that the [[:rest, :"..."], [:keyrest, :"..."], [:block, :"..."]] solution looks like a good option, as it keeps roughly the same behaviour while adding the differentiation between *args, &blk and ....

#8 - 02/23/2020 11:53 PM - connorshea (Connor Shea)

I'd definitely like to see this as well. It'd be useful for Sorbet since it uses the parameters method to reconstruct methods when generating scaffolds for gem methods and other Ruby code.

https://github.com/sorbet/sorbet/blob/29a967a22cc8cfcccb3d6a502b9ccb99cace0f5c/gems/sorbet/lib/gem-generator-tracepoint/tracepoint_serializer.rb#L186-L206

#9 - 06/18/2020 09:37 AM - mame (Yusuke Endoh)

Adding a new type of parameters will break existing code. In fact, if we add a new type :delegate, the Sorbet code written in #note-8 will raise "Unknown parameter type: #{type}". So, we must be careful.

Currently, (...) parameter generates a unique indicator [:rest, :*]. Note that it uses an invalid variable name :*, so it is not produced by other parameters than (...). Thus, without ambiguity, it is already able to determine if (...) parameter is used or not by checking if the result of Method#parameters includes [:rest, :*]. So, currently, I don't see a strong reason to add a new type.

That being said, feeling is important for Ruby. Changing :* to :"..." may be acceptable, though it brings incompatibility. I have no idea if it is worth or not.