Description

Here is a followup for a ruby2.7 issue discussed here [https://gitlab.com/groups/gitlab-org/-/epics/2380](https://gitlab.com/groups/gitlab-org/-/epics/2380)

I run gitlab with ruby2.7. gitlab/lib/api/api_guard.rb calls Rack's use method:

```ruby
use Rack::OAuth2::Server::Resource::Bearer, 'The API' do |request|
  request.access_token
end
```

The use method looks like:

```ruby
def use(middleware, *args, &block)
  if @map
    mapping, @map = @map, nil
    @use << proc { |app| generate_map app, mapping }
  end
  @use << proc { |app| middleware.new(app, *args, &block) }
end
```

For some reason, a Proc object was set to &block and added to args. It sounds wrong. A Proc should only be set to &block, and args should contain only one argument.

Related issues:

| Related to Ruby master - Bug #16504: `foo(*args, &args.pop)` should pass all ... | Open |

History

#1 - 01/11/2020 12:31 AM - mame (Yusuke Endoh)

Thank you for the report! I cannot reproduce the issue by a simple config.ru:

```ruby
require "rack/oauth2"

use Rack::OAuth2::Server::Resource::Bearer, 'The API' do |request|
  request.access_token
end
```

Adding p args into the definition of Rack's use shows only "The API". So I'd like to try it with the source code of gitlab. I have no idea at all about gitlab, so could you tell me how to reproduce it?

#2 - 01/11/2020 12:52 AM - ioquatix (Samuel Williams)

We cannot reproduce this.

Can you make some script to reproduce this in isolation?

Including Gemfile and Gemfile.lock details.

#3 - 01/11/2020 01:21 AM - ioquatix (Samuel Williams)

https://github.com/ruby-grape/grape/blob/d58dc0ab7a0b51625217deedd8110d1030be7cf7/lib/grape/middleware/stack.rb#L80-L84 is probably responsible for the failure.
On Ruby 2.7.0:

```
irb(main):020:-> x = [1, 2, ->{}]; puts(*x, &x.pop)
1
2 => nil
```
```
irb(main):021:-> x = [1, 2, ->{}]; puts(*x, &x.last)
1
2 #<Proc:0x0000562763a56398 (irb):21 (lambda)> => nil
```

This seems like buggy behaviour related to the original issue.

I'm not sure if that behavior is buggy. puts ignores a passed block, so the behavior seems expected.

```
def a(*args, &b)
  puts(*args, &b)
end
```
```
x = [1, 2, ->{}]; a(*x, &x.pop)
# => [[1, 2], #<Proc:0x00001100c1362518 (irb):4 (lambda)>]
x = [1, 2, ->{}]; a(*x, &x.last)
# => [[1, 2, #<Proc:0x00000562763a56398 (irb):21 (lambda)>], #<Proc:0x000011004f2b0ba0 (irb):5 (lambda)>]
```

Same results with Ruby 1.9.

Ruby 2.7 partially changed the behavior. Copied from my comment: https://github.com/ruby-grape/grape/issues/1967#issuecomment-573366122

```
# in 2.6 or before
args = [1, 2, ->{}]; foo( *args, &args.pop) #=> passes [1, 2] (bug; [1, 2, ->{}] is expected)
args = [1, 2, ->{}]; foo(0, *args, &args.pop) #=> passes [0, 1, 2] (bug; [0, 1, 2, ->{}] is expected)
```
```
# in 2.7
args = [1, 2, ->{}]; foo( *args, &args.pop) #=> passes [1, 2] (bug; [1, 2, ->{}] is expected)
args = [1, 2, ->{}]; foo(0, *args, &args.pop) #=> passes [0, 1, 2, ->{}] (good)
```

So, there are two issues.

- The issue that this ticket says is caused by a behavior change of 2.7, and ruby-grape's change will fix the issue https://github.com/ruby-grape/grape/commit/dec3e1ff5dbf3215a714565e62b12bd2e6b0dcb
- The behavior of args = [1, 2, ->{}]; foo( *args, &args.pop) should pass [1, 2, ->{}], and we should fix the bug on the ruby interpreter.
#8 - 01/12/2020 12:52 AM - mame (Yusuke Endoh)
- Related to Bug #16504: `foo(*args, &args.pop)` should pass all elements of args added

#9 - 01/12/2020 01:40 PM - sawa (Tsuyoshi Sawada)
- Description updated
- Subject changed from Argument added both to splat and last &block argument to Argument is added to both splat and last &block argument

#10 - 01/12/2020 04:38 PM - Eregon (Benoit Daloze)
mame (Yusuke Endoh) wrote:

- The behavior of `args = [1, 2, ->{}]; foo( *args, &args.pop)` should pass `[1, 2, ->{}], and we should fix the bug on the ruby interpreter.

Why is that better than the previous behavior?
Changing that will be more incompatible.

#11 - 01/12/2020 04:45 PM - Eregon (Benoit Daloze)
FWIW the 2.6 behavior is detailed in ruby/spec:
From commit
https://github.com/ruby/spec/commit/01992ab93dd893d9e8bf79db9f5f7d250982097
which shows RubyGems used to rely on the 2.6 behavior.

What I want to say is existing code relies on 2.6 behavior.
It's probably good to always evaluate positional arguments before the block argument for consistency, but it's an incompatible change.