Ruby trunk - Feature #15301

Symbol#call, returning method bound with arguments

11/13/2018 01:43 PM - zverok (Victor Shepelev)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

In one Reddit discussion I've got stuck with this simple, yet seemingly powerful idea, not sure if it was discussed anytime previously (can't find on the bug tracker, but maybe I am just bad at searching):

```
class Symbol
  def call(*args, &block)
    proc { |x| x.send(self, *args, &block) }
  end
end

[10, 20, 30].map &:modulo.(3) # => [1, 2, 0]
[[1, -2], [-3, -4]].map(&:map.(&:abs)) # => [[1, 2], [3, 4]]
[1, 2, 3, 4].map &:**.(2) # => [1, 4, 9, 16]
```

I understand and respect core team's reluctance for adding new methods to core classes, but from the top of my head I can't invent incredibly bad consequences (there, of course, could be some codebases that defined their own Symbol#call in a different way, but I don't estimate the probability as super-high; and the same could be said almost for any new method).

On the other hand, resulting code seems pretty nice, "Rubyish", explainable and mostly unambiguous.

Would like to hear other's opinions.

PS: One obvious objection could be that it is almost a de-facto standard to have any object's #to_proc to return proc doing exactly the same what the #call does (if the object happen to have both). It is unfortunate, but I believe the people will use to it, considering the possible gains. And, anyway, that's only "de-facto" rule, not the language standard :)

Related issues:

Is duplicate of Ruby trunk - Feature #12115: Add Symbol#call to allow to_proc... **Open**

History

#1 - 11/13/2018 01:48 PM - zverok (Victor Shepelev)

- Description updated

#2 - 11/13/2018 02:10 PM - Hanmac (Hans Mackowiak)

I once had a similar script (> 3 years old), but i extened it to be chainable

```
class Symbol
  class SymbolHelper
    def initialize(obj,methId,*args)
      @obj= obj
      @args=args
      @methId=methId
    end
    def method_missing(methId,*args)
      return SymbolHelper.new(self,methId,*args)
    end

    def to_proc
      proc {|obj| (@obj.nil? ? obj : @obj.to_proc.(obj)).public_send(@methId,*@args) }
    end
  end

  def call(*args)
    return SymbolHelper.new(nil,self,*args)
  end
end
```

you can chain it like that

```
[1,2,3,4].map(&:to_s.(2)) #=> ["1", "10", "11", "100"]
[1,2,3,4].map(&:to_s.(2).length) #=> [1, 2, 2, 3]
```

i think that was before the use of symbol kargs

#3 - 11/13/2018 02:42 PM - zverok (Victor Shepelev)

Hanmac (Hans Mackowiak) yeah, I myself invented and discarded several generations of similar things throughout my carreer.

But without too much of "going meta" (like hard-to-debug and guess "where it is from" method_missing tricks), I believe my suggestion is small and focused enough to eventually make its way into the lang.core.

#4 - 11/13/2018 03:28 PM - shevegen (Robert A. Heiler)

> I understand and respect core team's reluctance for adding new methods to core classes

Ultimately it is up to matz, but I believe it is not so much reluctance, but more the
general question "will this be of benefit to many people".

> [...] resulting code seems pretty nice, "Rubyish", explainable and mostly unambiguous.

I think this depends a lot on the particular style used by a ruby user. For example, to
me personally, a lot of the rails code is mysterious. To rails users rails is probably
simple to understand and "normal".

What I personally found with method_missing is that it can become very confusing. It's
great that we have method_missing but I am not sure that every use case of method_missing
is great.

Note that this is not a negative or positive opinion about your suggestion - I actually
am not even sure what your suggestion is about. :D

I think ".(2)" is very unusual though.

There is also a bit of a minor strangeness in the examples you gave, in that line:

```
[[1, -2], [-3, -4]].map(&:map.(&:abs)) # => [[1, 2], [3, 4]]
```

I think using .map twice like that is a bit odd. Matz likes DRI aka to not repeat
yourself (in ruby code) if possible. But again, that's mostly my opinion here; I
gladly leave that discussion to people who have a stronger opinion either way.

PS: Actually, I think this may require a comment by matz anyway, since I am not
sure it fits with how he sees Symbols e. g. having them respond to #call. I
understand that Symbols have changed a little bit over the years, e. g. got a
bit more String-like behaviour, but I am not sure if they are or should be
call-able? But again, I am not really invested into this idea, so I'll just
peace out. Please just view this here as suggestion for further comments rather
than anything pro or con. :)

#5 - 11/13/2018 03:45 PM - nobu (Nobuyoshi Nakada)

- Is duplicate of Feature #12115: Add Symbol#call to allow to_proc shorthand with arguments added

#6 - 11/14/2018 06:07 AM - marcandre (Marc-Andre Lafortune)

shevegen (Robert A. Heiler) wrote:

> ```
> [[1, -2], [-3, -4]].map(&:map.(&:abs)) # => [[1, 2], [3, 4]]
> ```
>
> I think using .map twice like that is a bit odd. Matz likes DRI

You probably misunderstood what the example is doing. map is not repeated per say here. You can replace the first one by flat_map for example:

```
[[1, -2], [-3, -4]].flat_map(&:map.(&:abs)) # => [1, 2, 3, 4]
```

#7 - 11/14/2018 08:30 AM - zverok (Victor Shepelev)

> What I personally found with method_missing is that it can become very confusing.

My proposal have absolutely nothing to do with method_missing.

> I think ".(2)" is very unusual though.

It is not, it is standard Ruby feature since Ruby 1.9, AFAIK (everything that responds to #call could be called with foo.(args)), for almost whole 11 years now. It is suggested as a preferred call-sequence by a lot of modern libraries.

> I think using .map twice like that is a bit odd.

Like marcandre (Marc-Andre Lafortune) explained already, it is just two nested cycles, e.g. the "non-abridged" version looks like this:

```
[[1, -2], [-3, -4]].map { |nested| nested.map(&:abs) }
```

#8 - 11/22/2018 10:41 PM - matz (Yukihiro Matsumoto)

Interesting idea of partial evaluation, but call is too generic, and could cause confusion. I am not positive about the expression.

Matz.

#9 - 11/23/2018 10:46 AM - zverok (Victor Shepelev)

matz (Yukihiro Matsumoto), but this is the whole point. Since introduction of .call() / .() synonyms, I looked for a good use for them, and this one looks almost perfect.

#10 - 12/16/2018 07:47 PM - shuber (Sean Huber)

matz (Yukihiro Matsumoto) (Yukihiro Matsumoto) wrote:

> Interesting idea of partial evaluation, but call is too generic, and could cause confusion. I am not positive about the expression.

Believe this concept is called partial application so maybe Symbol#apply? Would Proc#apply or Method#apply be useful as well?

#11 - 12/16/2018 09:24 PM - shuber (Sean Huber)

matz (Yukihiro Matsumoto)zverok (Victor Shepelev)

What do you think of this alternative syntax which wouldn't require a new method/operator?

```
[1, 2, 3, 4].map(&:**, 2) #=> [1, 4, 9, 16]
```

```
[1, 2, 3].tap(&:delete, 1) #=> [2, 3]
```

#12 - 12/16/2018 09:44 PM - zverok (Victor Shepelev)

> ...which wouldn't require a new method/operator?

...but instead will require the change of ALL the methods in standard library accepting a block? Or what do you mean?..

#13 - 12/16/2018 09:55 PM - shuber (Sean Huber)

zverok (Victor Shepelev) wrote:

> ...but instead will require the change of ALL the methods in standard library accepting a block? Or what do you mean?..

Exactly - no new method/operator, just changes at the parser/compiler level

#14 - 12/16/2018 09:58 PM - zverok (Victor Shepelev)

I don't believe it is possible at all (both from the technical and organizational points of view).
If you do, probably you should create another ticket, not turning this one, pretty short and focused on one particular proposal into the "who could invent 10 new syntaxes in one comment" challenge.

#15 - 12/17/2018 05:16 PM - shuber (Sean Huber)

zverok (Victor Shepelev) wrote:

> I don't believe it is possible at all (both from the technical and organizational points of view).

Ah ok bummer if true! Can you please clarify what you meant by organizational above?

Note to future self (or other interested parties) - poke around compile.c, parse.y, node.c, and node.h to see what something like this would take.

> If you do, probably you should create another ticket, not turning this one, pretty short and focused on one particular proposal into the "who could invent 10 new syntaxes in one comment" challenge.

Lol just trying to suggest some alternatives since Matz doesn't seem to be into the Symbol#call suggestion. I do very much like this shorthand concept of applying arguments to blocks/procs like you're proposing and I hope we get to see some implementation of it. Thank you for your feedback!

#16 - 01/12/2019 01:59 PM - zverok (Victor Shepelev)

In the light of discussions here: https://bugs.ruby-lang.org/issues/15428#change-76265 (TL;DR: #call is "implicit Proc conversion") I retract this proposal.

Please close.

#17 - 01/15/2019 03:22 AM - nagachika (Tomoyuki Chikanaga)

- Status changed from Open to Closed