Ruby trunk - Feature #12115

Add Symbol#call to allow to_proc shorthand with arguments

02/26/2016 05:51 PM - felixbuenemann (Felix Bünemann)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

I am a great fan of the Symbol#to_proc shorthand when mapping or reducing collections:

```
[1,2,16].map(&:to_s)
=> ["1", "2", "16"]
[1,2,16].reduce(&:*)
=> 32
```

I often wish it would be possible to pass an argument to the method when doing this, which currently requires a block and is more verbose:

```
[1,2,16].map { |n| n.to_s(16) }
=> ["1", "2", "10"]
# active_support example
{id: 1, parent_id: nil}.as_json.transform_keys { |k| k.camelize :lower }.to_json
=> '{"id":1,"parentId":null}'
```

It would be much shorter, if ruby allowed this:

```
[1,2,16].map(&:to_s.(16))
=> ["1", "2", "10"]
# active_support example
{id: 1, parent_id: nil}.as_json.transform_keys(&:camelize.(:lower)).to_json
=> '{"id":1,"parentId":null}'
```

This can be implemented easily, by adding the Symbol#call method:

```
class Symbol
  def call(*args, &block)
    ->(caller, *rest) { caller.send(self, *rest, *args, &block) }
  end
end
```

Source: [stackoverflow: Can you supply arguments to the map(&:method) syntax in Ruby?]

I think this is a rather common use case, so I propose to add Symbol#call to the Ruby standard library.

Related issues:

Related to Ruby trunk - Feature #4146: Improvement of Symbol and Proc	**Rejected**
Has duplicate Ruby trunk - Feature #15301: Symbol#call, returning method boun...	**Closed**

History

#1 - 02/26/2016 05:59 PM - felixbuenemann (Felix Bünemann)

- Description updated

#2 - 02/26/2016 06:03 PM - felixbuenemann (Felix Bünemann)

Edited to remove **kwargs argument I added, which would require checking if the called method supports them.

#3 - 02/27/2016 08:09 AM - nobu (Nobuyoshi Nakada)

- Related to Feature #4146: Improvement of Symbol and Proc added

#4 - 02/29/2016 06:35 PM - shelvacu (Shel vacu)

I agree that there should be some syntax for doing this, but I don't think this is the proper way to do it.

Personally, the syntax is confusing to me. I would prefer something like:

```
[1,2,16].map(&:to_s(16))
```

This way, the way my brain parses it is that &: is the operator for turning a symbol into a block that calls the method on the argument. However, this would require changes in the parser instead of stdlib.

My other concern is that Symbol#call returning a proc feels wrong. It leads to code like this:

```
a = :to_s
a.call(16).call(15)
```

While such code may never be written even if this is implemented, I hope it conveys how odd it feels to have a method named "call" always return a proc which is then meant to be called, instead of calling anything.

Would the change from &(:meth_name_as_symbol) to special operator &: followed by method name and optionally arguments (ie. the syntax I used above) break any existing code?

#5 - 02/29/2016 08:45 PM - shevegen (Robert A. Heiler)

I think there have been many other similar proposals. Nobu linked to other discussions.

From what I have seen, I think the major problem is coming up with a nice syntax proposal.

```
.map(&:foo)
```

is ok because it is short.

Adding implicit arguments to it is harder.

```
[1,2,16].map(&:to_s(16))
```

Is probably ok. But I am not sure if the parser is happy with it.

```
[1,2,16].map(&:to_s.(16))
```

Is not good IMHO, the . there is very confusing for me.

There is a slight alternative to .call() which is the []

I like [] a lot, but I think it looks a bit weird too inside of ().

Perhaps we do not have a syntax that will be better if we require arguments for &: ?

#6 - 02/29/2016 09:34 PM - felixbuenemann (Felix Bünemann)

Although I don't understand the Japanese, the linked issue, with a similar syntax to what Shel vacu proposed above, was rejected by Matz. So probably not too much hope on getting this into core…

#7 - 03/01/2016 02:41 AM - nobu (Nobuyoshi Nakada)

Yes, &:to_s(16) is exactly my (rejected) proposal.

#8 - 03/01/2016 06:15 AM - sawa (Tsuyoshi Sawada)

For a similar proposal, please cf. #10394.

#9 - 10/05/2017 03:35 AM - knu (Akinori MUSHA)

Wouldn't Array#to_proc make sense?

```
class Array
  def to_proc
    proc { |x| x.__send__(*self) }
  end
end

[100, 200, 300].map(&[:to_s, 16])
# => ["64", "c8", "12c"]
```

#10 - 10/05/2017 10:44 AM - zverok (Victor Shepelev)

> Wouldn't Array#to_proc make sense?

For me, it looks confusing. Array is (usually) a set of homogenous objects, so my first guess for Array#to_proc would be this:

```
[100,200,300].map(&%i[to_s reverse to_i])
```

(chain of calls on argument)

#11 - 10/06/2017 12:53 AM - jwmittag (Jörg W Mittag)

knu (Akinori MUSHA) wrote:

> Wouldn't Array#to_proc make sense?

```
class Array
  def to_proc
    proc { |x| x.__send__(*self) }
  end
end

[100, 200, 300].map(&[:to_s, 16])
# => ["64", "c8", "12c"]
```

I disagree: responding to to_proc in Ruby more or less means "I am a function-like thing". And arrays *are* function-like things, they are basically functions from their indices to their values. Having to_proc mean something *different* than that would be a big mistake, IMO. It would also be inconsistent with Hash#to_proc.

See #11653 (Hash#to_proc) for what I mean, and #11262 for a more comprehensive argument.

#12 - 10/06/2017 01:03 AM - nobu (Nobuyoshi Nakada)

jwmittag (Jörg W Mittag) wrote:

> I disagree: responding to to_proc in Ruby more or less means "I am a function-like thing". And arrays *are* function-like things, they are basically functions from their indices to their values. Having to_proc mean something *different* than that would be a big mistake, IMO. It would also be inconsistent with Hash#to_proc.

Even if arrays were function-like things, the elements are not arguments.

#13 - 10/06/2017 05:27 AM - duerst (Martin Dürst)

nobu (Nobuyoshi Nakada) wrote:

> jwmittag (Jörg W Mittag) wrote:
>
> > I disagree: responding to to_proc in Ruby more or less means "I am a function-like thing". And arrays *are* function-like things, they are basically functions from their indices to their values. Having to_proc mean something *different* than that would be a big mistake, IMO. It would also be inconsistent with Hash#to_proc.
>
> Even if arrays were function-like things, the elements are not arguments.

In Jörg's proposals, array elements are indeed not arguments, they are return values. Index values are arguments.

#14 - 10/06/2017 08:09 AM - knu (Akinori MUSHA)

I think &[symbol, *args] can be a natural extension to &symbol, as they are both a shorthand for { |_| _.__send__(*object) }.

For Array to provide #to_proc would be just a little bit weird convention for greater convenience, just as Symbol#to_proc is. Symbol had been in no way a function-like entity, but once Symbol#to_proc was added we almost instantly grew used to it and now we all take it for granted because being able to map(&:to_s) is so handy and useful.

The original proposal that is to introduce a new syntax is a bit too costly for one of the most frequently wanted features like this because it takes years of time before everyone can start using it. On the other hand, adding Array#to_proc is easily backportable and you can start using it today. FWIW, Matz once said he was not in favor of extending the syntax just for this:
http://blade.nagaokaut.ac.jp/cgi-bin/vframe.rb/ruby/ruby-dev/45404?45384-45592 (written in Japanese, try Google translate)

I've done some more googling and it turned out that Array#to_proc was not a new idea at all.

- https://www.sanityinc.com/articles/adding-array-to-proc-to-ruby/
- http://t.y13i.com/post/83319234720/ruby-%E3%83%96%E3%83%AD%E3%83%83%E3%82%AF%E6%9B%B8%E3%81%8D%E3%81%9F%E3%81%8F%E3%81%AA%E3%81%84%E7%97%85%E3%81%AB%E7%BD%B9%E6%82%A3%E3%81%97%E3%81%A6arraytoproc%E3%82%92%E5%AE%9A%E7%BE%A9%E3%81%97%E3%81%A6%E3%81%BF%E3%81%9F (written in Japanese)
- http://blade.nagaokaut.ac.jp/cgi-bin/vframe.rb/ruby/ruby-dev/45404?45384-45592 (written in Japanese)
- https://stackoverflow.com/a/5702174 (Jörg, I found you there!)

So, these people independently have reached the same idea! Isn't that a good sign?

#15 - 11/13/2018 03:45 PM - nobu (Nobuyoshi Nakada)

- Has duplicate Feature #15301: Symbol#call, returning method bound with arguments added

#16 - 11/16/2018 09:49 AM - RichOrElse (Ritchie Buitre)

I have a related proposal #15302 with a different implementation and interface.