Ruby - Feature #16102

`Symbol#call`

08/14/2019 06:59 AM - sawa (Tsuyoshi Sawada)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Since symbols have a to_proc method, it is natural to expect that they would appear in a method chain like:

```
:some_symbol.to_proc.call(...)
```

In fact, I have use cases like this:

```
arrays = [["a", "b"], ["c"], ["d", "e"]]
hashes = [{"a" => 1}, {"b" => 2, "c" => 3}, {"d" => 4, "e" => 5}]

:product.to_proc.(*arrays)
# => [["a", "c", "d"], ["a", "c", "e"], ["b", "c", "d"], ["b", "c", "e"]]
:zip.to_proc.(*arrays) # => [["a", "c", "d"], ["b", nil, "e"]]
:union.to_proc.(*arrays) # => ["a", "b", "c", "d", "e"]
:merge.to_proc.(*hashes) # => {"a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5}
```

I request Symbol#call to be defined, which would implicitly call to_proc on the receiver and then the conventional Proc#call on the result. Then, I can do:

```
:product.(*arrays) # => [["a", "c", "d"], ["a", "c", "e"], ["b", "c", "d"], ["b", "c", "e"]]
:zip.(*arrays) # => [["a", "c", "d"], ["b", nil, "e"]]
:union.(*arrays) # => ["a", "b", "c", "d", "e"]
:merge.(*hashes) # => {"a"=>1, "b"=>2, "c"=>3, "d"=>4, "e"=>5}
```

This would solve what proposals #6499, #6727, #7444, #8970, #11262 aim to do.

Notice that proposals #12115 and #15301 ask for Symbol#call, but they ask for different things (a method that returns a proc), and are irrelevant to the current proposal.

History

#1 - 08/14/2019 07:00 AM - sawa (Tsuyoshi Sawada)

Related to #6499, #6727, #7444, #8970, #11262.

#2 - 08/14/2019 07:24 AM - sawa (Tsuyoshi Sawada)

- Description updated

#3 - 08/14/2019 07:29 AM - sawa (Tsuyoshi Sawada)

- Description updated

#4 - 08/14/2019 08:37 AM - shevegen (Robert A. Heiler)

I have no particular pro/con opinion on the suggested functionality here itself.

In my opinion, this is mostly a design consideration for how "useful" matz
wants to see symbols being used in ruby. (This may not be directly related
to the comment here, but more generally in how simple, complex or useful
matz may want to see symbols.)

On a side note, it is (to me) interesting that sawa is not the only one
with somewhat related ideas in this regard, as he has pointed out via
mentioning https://bugs.ruby-lang.org/issues/12115. :)

#5 - 08/14/2019 09:00 AM - mame (Yusuke Endoh)

I agree that ary1.zip(ary2, ary3) is asymmetric and uncool, but I don't like solving the issue by adding a method of Symbol. :zip.(*arrays) looks too cryptic and semantically hacky to me. Rather, I like your #8970: Array.zip(ary1, ary2, ary3). It is much simpler, clearer, and easier to understand.

#6 - 09/12/2019 09:25 AM - inopinatus (Joshua GOODALL)

I propose this general solution.

```
diff --git a/array.c b/array.c
index 3717c3ff34..3809af01cf 100644
--- a/array.c
+++ b/array.c
@@ -6988,6 +6988,7 @@ Init_Array(void)
     rb_define_method(rb_cArray, "dig", rb_ary_dig, -1);
     rb_define_method(rb_cArray, "sum", rb_ary_sum, -1);
 
+    rb_define_method(rb_cArray, "splat", rb_yield_splat, 0);
     rb_define_method(rb_cArray, "deconstruct", rb_ary_deconstruct, 0);
 
     id_random = rb_intern("random");
```

then:

```
arrays = [["a", "b"], ["c"], ["d", "e"]]

arrays.splat(&:product)
#=> [["a", "c", "d"], ["a", "c", "e"], ["b", "c", "d"], ["b", "c", "e"]]
```