

Ruby - Feature #12125

Proposal: Shorthand operator for Object#method

02/28/2016 09:39 PM - Papierkorb (Stefan Merettig)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		
Description Hello, The & operator lets one pass a #call-able object as block. Really useful feature, but at the moment, if you want to pass a Method this way the syntax is not really concise: <pre>Dir["*/*.c"].map(&File.method(:basename))</pre> More often than not, at least I end up writing this instead <code>.map{ a File.basename a}</code> which isn't that great either. Thus, I want to propose adding a short-hand operator to the ruby language, which simply calls <code>#method</code> on an Object. It could look like this: <code>an_object->the_method</code> which is 100% equivalent to doing <code>an_object.method(:the_method)</code> I'm reusing the <code>-></code> operator which is already used for the stabby lambda. But I think it makes sense: You have an object, and from that object you point at a method to get it as Method. With this, the example from above becomes: <code>Dir["*/*.c"].map(&File->basename)</code> I attached a proof of concept patch. When you apply this to trunk, you can try the example above yourself. Do note however that this PoC also breaks stabby lambda for the moment. I'll work on fixing that the following days. Thank you for reading, Stefan.		
Related issues:		
Related to Ruby - Feature #16275: Revert `` syntax		Closed
Has duplicate Ruby - Feature #13581: Syntax sugar for method reference		Closed

Associated revisions

Revision 67c574736912003c377218153f9d3b9c0c96a17b - 12/31/2018 03:00 PM - nobu (Nobuyoshi Nakada)

Method reference operator

Introduce the new operator for method reference, `..`
[Feature #12125] [Feature #13581]
[EXPERIMENTAL]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@66667 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 67c574736912003c377218153f9d3b9c0c96a17b - 12/31/2018 03:00 PM - nobu (Nobuyoshi Nakada)

Method reference operator

Introduce the new operator for method reference, `..`
[Feature #12125] [Feature #13581]
[EXPERIMENTAL]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@66667 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 67c57473 - 12/31/2018 03:00 PM - nobu (Nobuyoshi Nakada)

Method reference operator

Introduce the new operator for method reference, `..`

History

#1 - 02/28/2016 09:43 PM - Papierkorb (Stefan Merettig)

The & operator lets one pass a #call-able object as block.

I meant #to_proc-able objects, sorry for the confusion.

#2 - 02/29/2016 12:37 AM - matz (Yukihiro Matsumoto)

I like the idea of short hand notation for Object#method(), but I don't think -> is a good idea.

Matz.

#3 - 02/29/2016 12:58 PM - zverok (Victor Shepelev)

For this kind of "conceptual" methods I sometimes define just one-letter shortcuts in core_ext.rb, like .map(&File.m(:basename)). I'm not sure, though, if any of existing/popular libraries will struggle from such kind of solution (or may be it would not play really well with local variables, which are frequently have one-letter names).

#4 - 02/29/2016 02:14 PM - Papierkorb (Stefan Merettig)

Yukihiro Matsumoto wrote:

but I don't think -> is a good idea.

Other options I can think of are:

- Using .>: the _object.>a_method While its look is nearer to a normal method call (Which I think is a plus), I fear that the period would be hard to see in some fonts. Another plus is that this would be a entirely new operator, so no (unintentional?) breaking changes in the parser.
- Using <.>: the _object<a_method> Inspired by the look of generics/templates in other programming languages. Should not clash with existing code and parsers and is well-readable in fonts I guess. Maybe it doesn't read as well anymore though, and the intention of <.> may not be that clear at first to someone who doesn't know the syntax (yet). No idea if that is a concern.
- Using &.: the _object&.>a_method Also readable in any font I can think of. It's a spin on &., reading like "and this"
- Using |>: the _object|>a_method I think the *Elixir* language has this operator too (albeit with other semantics?). Its read like "and pipe it through this", so maybe it reads a bit like a shell script too?

Regards,
Stefan

#5 - 02/29/2016 02:29 PM - ksss (Yuki Kurihara)

How about this one?

```
class UnfoundMethod
  def initialize(receiver)
    @receiver = receiver
  end

  def method_missing(name, *args, &block)
    @receiver.method(name)
  end
end

module UnfoundMethodAttacher
  def method(name=nil)
    if name
      super
    else
      UnfoundMethod.new(self)
    end
  end
end

Object.prepend UnfoundMethodAttacher

File.method #=> #<UnfoundMethod receiver=File>
```

```
File.method.basename #=> #<Method: File.basename>
Dir["*/*.c"].map(&File.method.basename) #=> ["foo.c", "bar.c"]
```

#6 - 02/29/2016 04:40 PM - funny_falcon (Yura Sokolov)

Please don't do this!!! No need to make the language more complex just to solve such small issue!!!

If you want to do something pretty, then whole closure syntax should be simplified, not just call to '#method'.

```
Dir["*/*.c"].map{File.basename(_0)} # where `_0` is magic var
```

Then bytecode compiler may optimize it to `.map(&File.method(:basename))`

But it then allows to do more pretty things, for example:

```
# dumps key=>value pairs
myhash.each{|k,v| puts "#{k}>#{v}"}
# do it in shorter way
myhash.each{puts "#{_0}>#{_1}"}
```

#7 - 02/29/2016 08:40 PM - shevegen (Robert A. Heiler)

I think the `&File->basename` looks confusing since we also have
-> standalone now.

`object->method` reminds me a lot of php/perl and ruby uses the
prettier `.` instead, `object.method`.

I also think that :

```
.map{File.basename(_0)} # where `_0` is magic var
```

Is not good either. I like `_` as a variable name a lot but on
its own, not with extra. :)

Yuki Kurihara's proposal is somewhat better as he does not
have to use special constructs/tokens.

```
Dir["*/*.c"].map(&File.method.basename)
```

I believe that crystal allows some parameter for `&;` and I think
there have been earlier proposals in ruby too.

But I think it makes sense: You have an object, and from that
object you point at a method to get it as `Method`.

I do not think that this argument is a good one because the `->`
is used in a dissimilar way, akin to `Proc.new / lambda`,
whereas your syntax proposal would be more similar to php
and perl syntax style, which I think will be confusing in
addition to `->` already having another method. So this is
not good in my opinion.

#8 - 03/01/2016 02:23 AM - nobu (Nobuyoshi Nakada)

Stefan Merettig wrote:

- Using `.>`: the `_object.>a_method`
- Using `<.>`: the `_object<a_method>`

These two conflict with existing syntax and break compatibility.
Note that `object.>(other)` is a valid method call.

#9 - 03/01/2016 02:24 AM - nobu (Nobuyoshi Nakada)

- Description updated

#10 - 03/14/2016 01:42 AM - jwmittag (Jörg W Mittag)

A proposal that has existed for years, if not decades, is to deprecate usage of the `::` double colon binary infix namespace operator for message sends,
and instead re-use it for method references:

```
Dir["*/*.c"].map(&File::basename)
```

This is also the syntax chosen by Java for method references.

There is one big problem, though: ambiguity with constant references for methods which start with an uppercase letter. Maybe, it would be possible to require parentheses in that case?

```
%w[1 2 3].map(&::Integer())
```

#11 - 03/14/2016 01:44 AM - jwmittag (Jörg W Mittag)

It would be nice if we could find symmetric syntax for getting an UnboundMethod from a module.

#12 - 03/14/2016 04:46 PM - mame (Yusuke Endoh)

- *File dot-symbol.patch added*

+1 for this proposal. How about `recv.fname`?

```
$ ./miniruby -e 'p Dir["*/*.c"].map(&File::basename) '
["hypot.c", "memcmp.c", "erf.c", ...]
```

Fortunately, it brings no conflict.

A patch is attached. (*dot-symbol.patch*)

--

Yusuke Endoh mame@ruby-lang.org

#13 - 03/15/2016 01:35 AM - nobu (Nobuyoshi Nakada)

Yusuke Endoh wrote:

+1 for this proposal. How about `recv.fname`?

`dot_or_colon` means that `File::basename` also works?

#14 - 03/15/2016 03:41 AM - mame (Yusuke Endoh)

Nobuyoshi Nakada wrote:

Yusuke Endoh wrote:

+1 for this proposal. How about `recv.fname`?

`dot_or_colon` means that `File::basename` also works?

Yes it works. But I don't think it is important. Only `..` is also okay to me.

--

Yusuke Endoh mame@ruby-lang.org

#15 - 03/15/2016 07:56 PM - Papierkorb (Stefan Merettig)

Yusuke Endoh wrote:

Nobuyoshi Nakada wrote:

Yusuke Endoh wrote:

+1 for this proposal. How about `recv.fname`?

`dot_or_colon` means that `File::basename` also works?

Yes it works. But I don't think it is important. Only `..` is also okay to me.

The tetris-operator `..` makes sense to me, but in respect to `::` I don't think we should allow "alternative" operators to do the same thing.

#16 - 03/15/2016 08:06 PM - funny_falcon (Yura Sokolov)

-1000

Please, don't!!!

I don't wanna Ruby to become Perl!!!

No more unnecessary syntax!!!

You all are not so weak! you are strong humans!!

You just can type a bit more characters!!!

#17 - 03/16/2016 08:57 AM - hanachin (Seiei Miyagi)

How about `File[:basename]` ?

#18 - 03/17/2016 06:07 AM - shyouhei (Shyouhei Urabe)

Just a status update: I heard from Matz in this month's developer meeting that however he wants this, all proposed syntax so far didn't charm him.

#19 - 03/17/2016 07:48 AM - nobu (Nobuyoshi Nakada)

Seiei Miyagi wrote:

How about `File[:basename]` ?

It causes confusion if the receiver has `#[]` method.

#20 - 03/17/2016 07:52 PM - VeryBewitching (RP C)

Stefan Merettig wrote:

Hello,

The `&` operator lets one pass a `#call-able` object as block.

Really useful feature, but at the moment, if you want to pass a `Method` this way the syntax is not really concise:

```
Dir["*/*.c"].map(&File.method(:basename))
```

More often than not, at least I end up writing this instead `.map{|a| File.basename a}` which isn't that great either.

Thus, I want to propose adding a short-hand operator to the ruby language, which simply calls `#method` on an `Object`.

It could look like this: `an_object->the_method` which is 100% equivalent to doing `an_object.method(:the_method)`

I'm reusing the `->` operator which is already used for the stabby lambda. But I think it makes sense: You have an object, and from that object you point at a method to get it as `Method`.

With this, the example from above becomes: `Dir["*/*.c"].map(&File->basename)`

I attached a proof of concept patch. When you apply this to trunk, you can try the example above yourself. Do note however that this PoC also breaks stabby lambda for the moment. I'll work on fixing that the following days.

Thank you for reading,
Stefan.

```
Dir["/c"].map(File[:basename])
```

From `File`, employ `basename` method. The referencing `&` should be applied to the method, not the class, as it is really the method you're concerned with. I'm not a language designer, but this is how I would *expect* this to work by looking at it.

#21 - 03/17/2016 08:53 PM - VeryBewitching (RP C)

Another thought: `Dir["/c"].map(File.&basename)`

#22 - 09/28/2017 12:07 PM - vassilevsky (Ilya Vassilevsky)

Is it possible to use a single colon for this?

```
object:name
```

```
File:basename
```

URI:parse

As far as I can see (not far, really, I don't even know C), it is currently not used for anything.

#23 - 09/28/2017 01:23 PM - nobu (Nobuyoshi Nakada)

A colon does too many things already, a ternary operator, a symbol literal, and a keyword argument.

#24 - 09/28/2017 01:32 PM - nobu (Nobuyoshi Nakada)

File[&.basename] and File.&.basename are valid syntax already.

#25 - 09/29/2017 12:19 AM - k0kubun (Takashi Kokubun)

- Has duplicate Feature #13581: Syntax sugar for method reference added

#26 - 12/31/2018 03:00 PM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset trunk|r66667.

Method reference operator

Introduce the new operator for method reference, ..

[Feature [#12125](#)] [Feature [#13581](#)]

[EXPERIMENTAL]

#27 - 11/20/2019 10:21 AM - znz (Kazuhiro NISHIYAMA)

- Related to Feature #16275: Revert `.:` syntax added

#28 - 03/24/2021 11:58 PM - cvss (Kirill Vechera)

Since the proposed feature was reverted in [#16275](#), maybe this issue should be re-opened to continue discussion and "re-introduce with a big picture".

And I'd like add another perspective:

```
upcased = somehash.each_pair.map{|key, value| [key.upcase, value.upcase]}.then(&Hash.[:])
```

instead of mixed chaining and parentheses

```
upcased = Hash[somehash.each_pair.map{|key, value| [key.upcase, value.upcase]}]
```

or a bit ugly chaining with #method

```
upcased = somehash.each_pair.map{|key, value| [key.upcase, value.upcase]}.then(&Hash.method(:[:]))
```

#29 - 03/25/2021 02:26 AM - nobu (Nobuyoshi Nakada)

- Status changed from Closed to Open

Re-opened as this feature has been reverted.

#30 - 03/25/2021 06:32 AM - marcandre (Marc-Andre Lafortune)

Since then, we have _1:

```
Dir["*/*.c"].map{File.basename(_1)}
```

My understanding is that a syntax for method would also be less performant.

@cvss' example with Hash#[] is not convincing as using to_h is simply better.

#31 - 03/25/2021 02:55 PM - cvss (Kirill Vechera)

[@marcandre \(Marc-Andre Lafortune\)](#), comparing performance, the #method way is better:

The map{File.basename(_1)} code has two performance leaks: 1) it creates a new block to pass it to 'map', 2) this block looks up the basename method on each iteration

The map(&File.method(:basename)) passes directly the method without creating an intermediate block and the basename is looked up only once

while expanding the argument of map.

```
a = ['aa/bb']*10000
Benchmark.measure { 100.times{ a.map{|a| File.basename(a)} } } # 0.400000 0.000000 0.400000 ( 0.411006)
Benchmark.measure { 100.times{ a.map(&File.method(:basename)) } }
# 0.310000 0.010000 0.320000 ( 0.330659)
```

#32 - 03/25/2021 03:25 PM - cvss (Kirill Vechera)

I agree, to_h is better, but for hashes only. When we have no such a shorthand with other classes or other constructors, we can:

```
some_even_set = some_array&.select(&:even?)&.then(&Set::new)
```

After writing some code in 2.7.0 with .: I feel that

```
[1, 2, 3].any?(&some_even_set.:include?)
```

is more clean than

```
[1, 2, 3].any?{|x| some_even_set.include? x}
[1, 2, 3].any?{some_even_set.include? _1}
```

#33 - 03/25/2021 03:32 PM - marcandre (Marc-Andre Lafortune)

cvss (Kirill Vechera) wrote in [#note-31](#):

[@marcandre \(Marc-Andre Lafortune\)](#), comparing performance, the #method way is better:

The map{File.basename(_1)} code has two performance leaks: 1) it creates a new block to pass it to 'map', 2) this block looks up the basename method on each iteration

My understanding is that 1) has no cost as the block is never captured 2) indeed plays a (small) role

The map(&File.method(:basename)) passes directly the method without creating an intermediate block and the basename is looked up only once while expanding the argument of map.

Right, but there needs to be a Method object created, which is non negligible.

Using benchmark-ips improves the reliability. Here are the results I get:

```
# 10000 elements (as above):
  method:      364.4 i/s
  block:       322.4 i/s - 1.13x (± 0.00) slower
# 1000 elements:
  method:      3624.2 i/s
  block:       3212.6 i/s - same-ish: difference falls within error
# 100 elements:
  method:      35250.7 i/s
  block:       33684.3 i/s - same-ish: difference falls within error
# 10 elements:
  block:       320628.1 i/s
  method:      249288.9 i/s - 1.29x (± 0.00) slower
# 3 elements:
  block:       956542.2 i/s
  method:      477741.7 i/s - 2.00x (± 0.00) slower
```

I would venture to say that most loops are done with arrays much smaller than 1000 elements.

Here is the code I used:

```
gem 'benchmark-ips'
require 'benchmark/ips'

[10000, 1000, 100, 10, 3].each do |n|
  a = ['aa/bb']*n
  puts "#{n} elements"
  Benchmark.ips do |x|
    x.report('block') {
      a.map{|a| File.basename(a)}
    }
    x.report('method') {
      a.map(&File.method(:basename))
    }
  end
end
```

```

    x.compare!
  end
end
end

```

#34 - 03/25/2021 03:37 PM - marcandre (Marc-Andre Lafortune)

@cvss, your two other examples are also not convincing. Here is how one can write them:

```

some_even_set = some_array.select(&:even?).to_set

[1, 2, 3].any?(some_even_set)

```

#35 - 11/08/2021 06:13 PM - Azuma-01 (Azuma Eleth)

Hello,

I cannot help myself to see than all proposal are of the form:

(expression) (shorthand_operator) (method_name)

while I had seen some interesting option, to me, that format seem to add an obscure behaviour rater than a shorthand.

Personaly, when i mix the words 'Ruby' and 'shorthand', i see %string

%i, %q, %r, %s, %w, %x

Why not adding %m to the party? something like

%m{(expression)(space)(method_name)}

That should not be to hard to idantify and translate to

(expression).method(:method_name)

If there is no expression, we assume the value self

```

%m<Math sqrt>          #=> #<Method: Math.sqrt>
%m(foo.bar.baz something) #=> #<Method: Baz#something>
%m:to_s:               #=> #<Method: main.to_s()>
[1, 4, 9, 16, 25].map(& %m"Math sqrt") #=> [1.0, 2.0, 3.0, 4.0, 5.0]

```

To enable meta-programming, it can be done the same way than all others %string does it...

with an upper case version.

```

x = ::Kernel.gets.chomp
%M[MyModule #{x}]

```

In that form, the question I ask myself is if the method_name should be an explicit symbol or not. The above exemple assume it dose not.

#36 - 12/02/2021 02:00 PM - cvss (Kirill Vechera)

One more use case - method composition, for example from [#18369](#):

```

collection.detect(&:first_name>>"Dorian".:==)

```

#37 - 01/07/2022 06:23 PM - konslebox (K B)

Has && been considered?

```

Dir["*/*.c"].map(&&File.basename)

```

It can also adapt to outside usage so &File.basename equals File.method(:basename).

Files

method_shorthand.diff	740 Bytes	02/28/2016	Papierkorb (Stefan Merettig)
dot-symbol.patch	554 Bytes	03/14/2016	mame (Yusuke Endoh)