Ruby - Feature #5364

How about new syntax: "object.\method" returns a Method instance?

09/25/2011 11:36 PM - yimutang (Joey Zhou)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

I'm afraid the normal way of creating a Method instance is circuitous and a bit hard to write:

```
a_method = object.method(:method_name)
```

I find the way of obtaining a function/method object in Python is so easy, for example, "str.split()" returns a list, while "str.split" (without parentheses) returns a method object. However, parenthesis in Ruby is always optional, so this road is blocked.

Well, how about "object.\method" style? You can type "str.\split" to get "str.method(:split)".

The reasons are:

1. It makes people happy, writing less code, and no harm to readability.
2. "" is not a frequently used token. It can only be used in strings, regexps (or any else?). I think using this token leads to no ambiguity.
3. "" is like Perl's referrence symbol. In Perl, you can use "$f = &func;" to make a referrence to the function. In Ruby, "&" seems unnecessary.
4. It enhances the consistency of the language syntax. There are two correlative methods "Object#send" and "Object#method", but...

   str.send(:split) == str.split
   str.method(:split) == ???????

If adding this new syntax, it looks more pretty:

```
str.method(:split)    == str.\split
```

History

#1 - 09/26/2011 12:04 AM - waj (Juan Wajnerman)

How about "object.:method" ? Looks more Ruby style for me.

#2 - 09/26/2011 01:23 AM - aprescott (Adam Prescott)

What about object:method ? The \ version doesn't seem very appealing.

#3 - 09/26/2011 05:13 AM - drbrain (Eric Hodel)

- Category set to core

=begin
I find myself rarely needing to call #method and I don't often see code where other people call #method, so I don't see the need to add special syntax for it. Is there a reason why you use #method very often and that having a more compact syntax would help?

Note that \ can be used at the end of a line to continue it to the next line:

return
really_long_expression
=end

#4 - 09/26/2011 05:53 AM - Anonymous

Of note, there is already a working patch (for a very similar feature) for the 1.9 branch at the ruby-patches github repo [0], ported from the old Suby experiment project [1].

It doesn't use ".", just "" and has an implicit self form, too. So:

\class #

#5 - 09/26/2011 07:23 AM - yeban (Anurag Priyam)

On Sun, Sep 25, 2011 at 8:34 PM, Juan Wajnerman jwajnerman@manas.com.ar wrote:

> Issue #5364 has been updated by Juan Wajnerman.
>
> How about "object.:method" ? Looks more Ruby style for me.

Haskell uses '' to create lambda which seems very natural to me. I like the idea of using '' to get get method objects in Ruby.

--
Anurag Priyam

#6 - 09/26/2011 09:29 AM - jballanc (Joshua Ballanco)

On Sun, Sep 25, 2011 at 10:36 AM, Joey Zhou yimutang@gmail.com wrote:

> 1. It makes people happy, writing less code, and no harm to readability.

Personally, I would like to hear more on how having another operator for new Rubyists to learn does not harm readability? One of the oft-cited advantages of Ruby vs Perl is that Perl so often devolves into characters indistinguishable from line-noise. Admittedly, it does seem that certain languages are more than happy to chase down this rabbit hole (Haskell, Scala, C++, etc.), but I think Ruby is not competing for mind-share with these sorts of languages. Rather, I think Ruby could stand to learn a lot from the simplicity of a language like Lua.

Furthermore, I do not think that the worst part of the obj.method(:meth_name) form is its verbosity or really any part of its syntax. Instead, what troubles me is all of the various hidden differences between a method object, a proc object, and a proc(lambda) object, and the relative uselessness of detached methods (e.g. they can only be rebound to objects of the same class; not even subclasses are valid). I would rather see these addressed before worrying about adding new syntax.

#7 - 09/26/2011 10:53 AM - normalperson (Eric Wong)

Joshua Ballanco jballanc@gmail.com wrote:

> On Sun, Sep 25, 2011 at 10:36 AM, Joey Zhou yimutang@gmail.com wrote:
>
> > 1. It makes people happy, writing less code, and no harm to readability.
>
> Personally, I would like to hear more on how having another operator for new Rubyists to learn does not harm readability? One of the oft-cited advantages of Ruby vs Perl is that Perl so often devolves into characters indistinguishable from line-noise.

Completely agreed. A smaller syntax and fewer operators improves readability and learnability in my experience[1]. I think pronounce-ability is a good factor to readability, too.

Code is written once and read many times (often by people who are not the author), so optimize for readability.

[1] - I've done a fair bit of Perl, but also languages with small syntaxes like AWK and C.

#8 - 09/26/2011 11:19 AM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Feedback

We need a needs fot this syntax. I don't reject it right now, because there's a
room for someone to find a real use case for it. Through personally I'm skeptical
about existance of such thing.

#9 - 09/26/2011 12:16 PM - trans (Thomas Sawyer)

It's just a convenience thing, so I think it's not possible to prove a "real use case" as such. And as only a static syntax it's probably not a convenience worth the bother. However it might be more understandable if it could also be used dynamically, e.g.

```
m = :split
str:m
```

Syntax not withstanding. I suppose if this were to be done, the nicest syntax would be:

```
str#split
```

and

```
str#:m
```

In which case comments would require a space before the hash mark.

But, I think this hints at a larger matter that a future version of Ruby could really use -- real meta-programming functions. As things stand Ruby's meta-programming methods are fragile. Hence the need for things like **id** and **send**. Likewise, if #method is overwrite by some class, it could spell a bad day for some meta-code. As to the syntax for real meta-programming functions, I'm not sure. Perhaps a "global meta-izing syntax" like:

```
$(object).id
$(object).send(...)
```

And then perhaps #[] could get a method:

```
$(object)[:foo]
```

On the other hand, perhaps $ could be used as a "meta-message" device:

```
object$id
object$send(...)
object$:foo
```

In any case, one way or the other a more robust address of meta-programming would be a good thing.

#10 - 09/26/2011 12:38 PM - mame (Yusuke Endoh)

- Status changed from Feedback to Open

Hello,

To prevent `bikeshed' discussion, we should focus use case first
about this ticket, I think. Why don't you talk about the concrete
notation after that?

I come up with some use cases. First case:

```
ary.map {|x| x+3 }
ary.map(&3.+)
```

I like the latter not only because it is shorter, but also because
I don't have to bother about the name of the temporal variable.
We can already write ary.map(&:succ) in 1.9, so I think this style
will become more in demand.
Indeed, there is a trade-off between readability and writability.
I think it is a matter of getting familiar with it, though.

Another one for DSL. Consider rspec example:

```
lambda { foo.bar(baz) }.should raise(SomeError)
foo.\bar.with(baz).should raise(SomeError)
```

I think the latter is more readable than the formar.
I guess there are many possible use case in this field.

Finally, use case in future.
Ruby's Method class is poor currently.
I believe this proposal will encourage us to enrith Method class.
For example, I often use Object#method to glance over an unfamiliar

library:

irb(main):001:0> unfamiliar_obj.methods - 0.methods
[..., interesting_method, ...]

After I find an interesting method, I also want to read the rdoc
in similar way:

irb(main):002:0> unfamiliar_obj.\interesting_method.doc

Of course there is no Method#doc currently. I think this shorthand
notation will enrich such a feature.

--
Yusuke Endoh mame@tsg.ne.jp

#11 - 09/26/2011 03:37 PM - yimutang (Joey Zhou)

Well, Proc instances and Method instances are both closures. To obtain a closure, the Ruby idiom is almost from a block, not very often from a method. But in some situation, creating a closure from a method is more natural.

For example, if I want this:

```
['file1', 'file2', 'file3'].map {|f| open(f) }
```

how about:

```
['file1', 'file2', 'file3'].map &\open
```

and:
str = "abcdabcdabcd"
%w(a b c d).map {|sep| str.split(sep)}

how about:

```
%w(a b c d).map &str.\split
```

Of course, every Method instance can be converted to a Proc object with Method#to_proc.

```
str.method(:split)
```

is somewhat like:

```
lambda {|*args| str.send(:split, *args) }
```

but I'm afraid it's verbose to write and read.

I've just begun to learn Python recently, and find some interesting differences between the two languages.

Python's lambda is trivial, only one line code is allowed to create a lambda. However, its functions are first class objects, so that can be assigned to another variable easily.

Python seems mainly using objects from function to do some functional programming thing, while Ruby seems mainly using objects from blocks.

```
>>> a = lambda n: n + 1
>>> a
<function <lambda> at 0x011F6170>
>>> def foo(n):
        return n + 1
>>> b = foo
>>> b
<function foo at 0x011F2670>
```

whether from a lambda, or from a function, the returnning objects are both functions.

But in Ruby, if we do the similar thing, a will be a Proc instance, and b will be a Method instance.

Ruby distinguishes Proc and Method, but seems not paying much attention to Method? You can obtain a closure from a method, but it seems not Ruby idiom?

Joshua Ballanco and Yusuke Endoh san remind me the deeper subject: what's the future plan about the relationship between Proc and Method?

#12 - 09/27/2011 03:59 AM - jballanc (Joshua Ballanco)

On Monday, September 26, 2011 at 2:37 AM, Joey Zhou wrote:

> Joshua Ballanco and Yusuke Endoh san remind me the deeper subject: what's the future plan about the relationship between Proc and Method?

I think that, in order to address Method and Proc and how to go forward with them, we first need to meditate a bit about Ruby's treatment of first-class environments (or, in Ruby parlance, 'binding'). I will be giving a presentation at RubyConf this coming Thursday in which I hope to present what I feel are the biggest concerns surrounding environments. If anyone will be attending and would like to discuss bindings, methods, procs, and lambdas, I would love to get together and chat...

#13 - 09/27/2011 06:36 AM - kstephens (Kurt Stephens)

Do we *really* need new syntax to support a basic metaprogramming feature?
If Ruby metaprogramming is first-class, no additional syntax should be needed. This is a slippery slope.
This is what makes Ruby beautiful. :)

#14 - 10/02/2011 11:00 PM - yimutang (Joey Zhou)

How about this?

I want to create an enumerator from a method, the normal way is using the Object#to_enum method, if we add a method Method#to_enum, we can write in a clearer way. For example:

str.each_char # want an enumerator from this

normal way ==> str.to_enum(:each_char)

clearer way ==> str.\each_char.to_enum

of course, str.each_char returns an enumerator itself, but some other methods are not.

Dir.glob("*.rb") {|f| } # if I want an enumerator, not an array:

normal way ==> Dir.to_enum(:glob, "*.rb")

clearer way ==> Dir.\glob.to_enum("*.rb")

str.scan(/[abc]+/) {|s| }

normal way ==> str.to_enum(:scan, /[abc]+/)

clearer way ==> str.\scan.to_enum(/[abc]+/)

array.sort {|a,b| }

normal way ==> array.to_enum(:sort)

clearer way ==> array.\sort.to_enum

In my opinion, compare with the methods which take a method name as its argument, "object.\method" style is more readable.

#15 - 10/25/2011 09:34 AM - trans (Thomas Sawyer)

I wonder if

str->:each_char

Would be viable.

#16 - 10/26/2011 02:53 PM - nobu (Nobuyoshi Nakada)

Hi,

(11/10/25 9:34), Thomas Sawyer wrote:

> I wonder if
>
> str->:each_char
>
> Would be viable.

No.

1. it doesn't appear what it means well.
2. it caused hundreds of conflicts.

--
Nobu Nakada

#17 - 03/27/2012 03:22 AM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

Anyone is interested in facilitating the discussion?

Personally, I believe this proposal is the right way.
I tentatively assign this to matz, but I don't think
this is going to end well.

--
Yusuke Endoh mame@tsg.ne.jp

#18 - 03/27/2012 04:17 AM - trans (Thomas Sawyer)

Personally I think it's a ugly notation. Also methods aren't 1st class --you don't get the same method object each time. If we did, then I say it agree it would be important to have a concise syntax.

#19 - 03/31/2012 12:32 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

I am not against adding syntax notation to get method object, but I am not satisfied with all proposed notations. I feel they are not intuitive enough. Maybe we can accustom to one of them in the future. Until that I can stand with #method().

Matz.