Ruby master - Feature #4910

Classes as factories

06/20/2011 08:50 PM - rklemme (Robert Klemme)

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

Description

I suggest to add these two to class Class:

```
class Class
  alias call new

  def to_proc(*args)
    lambda {|*a| new(*args)}
  end
end
```

Then we can use class instances where blocks are needed and can easily use them as factory instances using the general contract of #call (see example attached).

Related issues:

Related to Ruby master - Feature #14498: Class#to_proc	**Rejected**

History

#1 - 06/20/2011 10:08 PM - Eregon (Benoit Daloze)

Hello,

Robert Klemme wrote:

> I suggest to add these two to class Class:
>
> ```
> class Class
>   alias call new
>
>   def to_proc(*args)
>     lambda {|*a| new(*args)}
>   end
> end
> ```

Did you want to mean:

```
def to_proc
  lambda { |*args| new(*args) } # or maybe lambda { |args| new(*args) }
end
```

?

#to_proc is called with no arguments (Symbol.instance_method(:to_proc).arity # => 0).

> Then we can use class instances where blocks are needed and can easily use them as factory instances using the general contract of #call (see example attached).

I don't really see the advantage of defining #call, you could use #newinstead at line 16.
If you want more flexibility, I believe it is fine to use a block.

But I like Class#to_proc, and it is indeed some kind of factory helper:

```
Pos = Struct.new :x,:y
[[1,2],[3,4]].map(&Pos) # => [#&lt;struct Pos x=1, y=2&gt;, #&lt;struct Pos x=3, y=4&gt;]
# instead of
[[1,2],[3,4]].map { |x,y| Pos.new(x,y) }
```

```
# note neither #to_proc defined as "lambda { |*args| new(*args) }" nor map(&Pos.method(:new)) would work:
# ([#&lt;struct Pos x=[1, 2], y=nil&gt;,...])
```

The obvious limitation being the lack of flexibility for common arguments (e.g.: y always the same). You would then have to use an explicit block.

I do not know if it is worth to add it for this specific case, but it can be nice.

I am also unsure if we need factories in Ruby (certainly not like in statically typed languages).

#2 - 06/20/2011 11:29 PM - rklemme (Robert Klemme)

Benoit Daloze wrote:

> Hello,
>
> Robert Klemme wrote:
>
> > I suggest to add these two to class Class:
> >
> > ```
> > class Class
> >   alias call new
> >
> >   def to_proc(*args)
> >   lambda {|*a| new(*args)}
> >   end
> > end
> > ```
>
> Did you want to mean:
>
> ```
> def to_proc
>   lambda { |*args| new(*args) } # or maybe lambda { |args| new(*args) }
> end
> ```
>
> ?
>
> #to_proc is called with no arguments (Symbol.instance_method(:to_proc).arity # => 0).

No, it was meant exactly as stated. Advantage is that you can provide parameters to #new if needed while mapping the parameterless call of to_proc easily to the parameterless call of Class#new.

> > Then we can use class instances where blocks are needed and can easily use them as factory instances using the general contract of #call (see example attached).
>
> I don't really see the advantage of defining #call, you could use #new instead at line 16.
> If you want more flexibility, I believe it is fine to use a block.

That's the exact point: by aliasing #new to #call we can pass in a lambda OR a class instance. The most general contract would then be '#call'able (i.e. an anonymous callback function) and as a shortcut we can pass in a class instance.

> But I like Class#to_proc, and it is indeed some kind of factory helper:
>
> ```
> Pos = Struct.new :x,:y
> [[1,2],[3,4]].map(&Pos) # => [#&lt;struct Pos x=1, y=2&gt;, #&lt;struct Pos x=3, y=4&gt;]
> # instead of
> [[1,2],[3,4]].map { |x,y| Pos.new(x,y) }
>
> # note neither #to_proc defined as "lambda { |*args| new(*args) }" nor map(&Pos.method(:new)) would work:
> # ([#&lt;struct Pos x=[1, 2], y=nil&gt;,...])
> ```
>
> The obvious limitation being the lack of flexibility for common arguments (e.g.: y always the same). You would then have to use an explicit block.
>
> I do not know if it is worth to add it for this specific case, but it can be nice.

I had considered that case as well and felt it might not be as common as the case where we try to provide arguments. I do not have any statistics though and I hope for others shedding some more light what they deem more useful.

A variant would be

```
class Class
  def to_proc(*args)
```

```
    if args.empty?
      lambda {|*a| new(*a)}
    else
      lambda {|*a| new(*args)}
    end
  end
end
```

In other words: if arguments are passed to to_proc use them as sole method arguments for #new; if not, use whatever is passed to the proc (which would support your mapping example).

We could probably make things even more complex by appending *a to *args and truncating the list with the arity of #new at the time of invocation of the block (or, more efficient, time of call of to_proc).

> I am also unsure if we need factories in Ruby (certainly not like in statically typed languages).

Any class in Ruby *is* a factory object already with method #new being the factory method.

#3 - 06/23/2011 02:12 AM - headius (Charles Nutter)

I'm not sure I agree with adding to_proc to Class instances, since it seems questionable that #new is what you'd always want to be called. Dodging that debate for now, there is another way to get the result you seek:

```
class Foo
  def initialize(i)
    @i = i
  end
end

(1..50).map(&Foo.method(:new))
```

This is both more explicit and less magic. If there were syntax added to get method objects (without calling #method) it would be even cleaner.

#4 - 06/24/2011 12:25 AM - rklemme (Robert Klemme)

Charles Nutter wrote:

> I'm not sure I agree with adding to_proc to Class instances, since it seems questionable that #new is what you'd always want to be called.

Hmm, but what else? I think it is a reasonable default.

> Dodging that debate for now, there is another way to get the result you seek:
>
> ```
> class Foo
>   def initialize(i)
>     @i = i
>   end
> end
>
> (1..50).map(&Foo.method(:new))
> ```
>
> This is both more explicit and less magic. If there were syntax added to get method objects (without calling #method) it would be even cleaner.

That's true. Though in absence of that syntax I prefer (1..50).map {|i| Foo.new i} over your solution as it is equally explicit and even less magic - could even be shorter to type. :-) Actually only (1..50).map(&Foo) would be an alternative I would consider.

Cheers

#5 - 03/25/2012 04:28 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

#6 - 11/20/2012 09:41 PM - mame (Yusuke Endoh)

- Target version set to 2.6

#7 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

#8 - 02/20/2018 08:50 AM - nobu (Nobuyoshi Nakada)

- Description updated

#9 - 02/21/2018 05:13 AM - matz (Yukihiro Matsumoto)

- Related to Feature #14498: Class#to_proc added

#10 - 02/21/2018 05:14 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

It can lead to unreadable code.

Matz.

Files

pro.rb	836 Bytes	06/20/2011	rklemme (Robert Klemme)