Ruby master - Feature #12113

Global method inside Delegator causes NameError

02/26/2016 09:17 AM - oleg_antonyan (Oleg Antonyan)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

```
def some_func
  puts '12'
end

class Klass < Delegator
  def initialize(obj)
    @obj = obj
  end

  def __getobj__
    @obj
  end

  def func
    some_func
#=> /home/oleg/.rbenv/versions/2.3.0/lib64/ruby/2.3.0/delegate.rb:85:in `instance_method': undefin
ed method `some_func' for module `Kernel' (NameError)
  end
end

Klass.new(0).func
```

Delegator uses Kernel.instance_method (https://github.com/ruby/ruby/blob/trunk/lib/delegate.rb#L85) but:

```
::Kernel.respond_to?(:some_func, true) #=> true
::Kernel.instance_method(:some_func)
#=> `instance_method': undefined method `some_func' for module `Kernel' (NameError)
::Kernel.method(:some_func)            #=> #<Method: Module(Object)#some_func>
```

I think there should be Kernel.method instead of instance_method (in Delegator). Otherwise you get respond_to? == true, but cannot use this method and get an error.

History

#1 - 02/28/2016 05:17 AM - nobu (Nobuyoshi Nakada)

- Tracker changed from Bug to Feature

- Description updated

It's the current spec that delegator cannot call private method of the target, because it was impossible to tell it was called as function call form, (without the receiver).

https://github.com/ruby/ruby/compare/trunk...nobu:feature/12113-Delegator-private-method
https://github.com/ruby/spec/compare/master...nobu:delegator-send

#2 - 02/28/2016 09:50 AM - Eregon (Benoit Daloze)

Would that solve the OP example code since some_func is defined in Object, not Kernel?

Using Kernel.method would be wrong, because the receiver would be the Kernel class, and not the delegated object (Klass.new(0).inspect would return "Kernel" for instance).

It would be quite tempting to make Delegator inherit from Object (or include the true Kernel) and redefine all the existing methods to try the delegate first,
but that would not work for methods added to Object/Kernel after require 'delegate', and those would just hide the methods of the delegate.

Maybe this could be worked around by hooking into #method_added, but that would not be very resilient if the user code redefines Object/Kernel#method_added.
If we had a way to hook into new method definition without the user accidentally break it, maybe this would be a good implementation. Or am I missing something?

#3 - 02/29/2016 08:01 AM - nobu (Nobuyoshi Nakada)

Benoit Daloze wrote:

> Would that solve the OP example code since some_func is defined in Object, not Kernel?

It is because of the private method of the target, in Object or in Kernel or in target itself is not the point.

#4 - 02/29/2016 10:17 AM - Eregon (Benoit Daloze)

Nobuyoshi Nakada wrote:

> Benoit Daloze wrote:
>
> > Would that solve the OP example code since some_func is defined in Object, not Kernel?
>
> It is because of the private method of the target, in Object or in Kernel or in target itself is not the point.

Ah, right, because in this case the method would be found on target since target is an Object (a Fixnum actually).

Would this patch allow to remove the special case for Kernel dispatch then? Or should it be kept for compatibility if !target.is_a?(Kernel) and most likely a BasicObject?

#5 - 02/29/2016 12:35 PM - nobu (Nobuyoshi Nakada)

Benoit Daloze wrote:

> Would this patch allow to remove the special case for Kernel dispatch then? Or should it be kept for compatibility if !target.is_a?(Kernel) and most likely a BasicObject?

It's necessary to avoid infinite recursion in the case of [Bug #9155].

#6 - 03/01/2016 01:58 AM - nobu (Nobuyoshi Nakada)

Correction: the infinite recursion was a bug in the patch, and fixed the branch.

The trick of Kernel for [Bug #9155] is for the case a global method is called *before* the target object is set.
It is for the backward compatibility and convenience, we may be better to warn to use :: and remove it in the future.