Ruby trunk - Bug #13781

Should the safe navigation operator invoke `nil?`

08/04/2017 03:18 AM - ioquatix (Samuel Williams)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	2.4.1	**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

In the following example:

```
class Later < BasicObject
    def initialize(&block)
        raise ::ArgumentError, "Block required" unless block

        if block.arity > 0
            raise ::ArgumentError, "Cannot store a promise that requires an argument"
        end
        @block  = block
    end

    def __resolve__
        @result ||= @block.call
    end

    def nil?
        __resolve__.nil?
    end

    def respond_to?(name)
      __resolve__.respond_to?(name)
    end

    def method_missing(name, *args, &block)
        __resolve__.__send__(name, *args, &block)
    end
end

Person = Struct.new(:name, :age)

person = Later.new do
  nil # Person.new("Froob", 200)
end

puts person.nil?
puts person&.name
```

The code fails because person is a proxy object.

If safe navigation operator invoked nil? it should work. But, it's not clear exactly how the implementation should behave, or whether it's possible to implement this style of proxy.

History

#1 - 08/04/2017 04:59 AM - marcandre (Marc-Andre Lafortune)

Short answer is "no".

Longer answer is:

- is there an actual use case? I very much doubt there is one

- BasicObject does not respond to nil?, so the safe operator would not work in that case?
- other Ruby conditional (like if foo or foo ? bar : baz) do not call nil? or ==(nil), they simply do a straight comparison with nil and false. That's the way it should be for the safe navigation operator too.

#2 - 08/04/2017 05:21 PM - shevegen (Robert A. Heiler)

> is there an actual use case? I very much doubt there is one

This is what some filed issues appear to be - a mostly theoretical
view that does not appear to be likely to emerge.

When Hiroshi filed the request, I do not think that he had any
"proxy" object in mind - it was simply to avoid compound methods
check e. g:

> if u && u.profile && u.profile.thumbnails && u.profiles.thumbnails.large

versus Activerecord

> if u.try!(:profile).try!(:thumbnails).try!(:large)

To be honest, I actually find the .try! variant more readable than the
&. variant but this is not the topic of course (not sure why activerecord
uses the '!' there, it also makes the chain ugly, in my opinion; perhaps
I am way too picky).

https://bugs.ruby-lang.org/issues/11537

#3 - 11/22/2017 09:44 AM - marcandre (Marc-Andre Lafortune)

- Status changed from Open to Closed