Ruby master - Feature #10982

Clarify location of NoMethod error

03/18/2015 05:59 PM - schneems (Richard Schneeman)

Status:	Open	
Priority:	Normal	
Assignee:		
Target version:		

Description

In Ruby, the error NoMethodError on happens frequently, especially when it occurs on nil. This error can be confusing to beginners since, many of them think there is a problem with the method instead of the receiver. This error can be confusing to advanced developers when there are multiple method calls in the on the same line. In this example it is unclear if foo or bar returned nil:

```
foo.call && bar.call
NoMethodError: undefined method `call' for nil:NilClass
```

I would like to make this a better error message and to indicate where the exception occurs.

```
@foo.call && @bar.call
                 ^----
NoMethodError: The method `call` is undefined on receiver nil:NilClass
```

Now it is more clear that the @bar is currently nil and that is the source of the error. I believe exposing this information will help developers of all abilities.

History

#1 - 03/19/2015 10:47 PM - parkr (Parker M)

This new syntax is a pretty sizable divergence from traditional Ruby error reporting. At the moment, errors generally (always?) have an accompanying stack trace:

```
~$ ruby errors.rb
errors.rb:2:in `<main>': undefined method `call' for nil:NilClass (NoMethodError)
```

It indicates the line, but excludes the column. It's uniform for all errors, so users seeing these stack traces know in what file, what line, and what method (the binding?) the error occurred. If the proposed format were implemented, would it only be for NoMethodError's, or would it apply to ArgumentError's or TypeError's, and so on?

Would the column number suffice? So your example would yield:

```
irb:1:16:in `<main>': undefined method `call' for nil:NilClass (NoMethodError)
```

You'd know that line 1 at column 16 caused the error.

#2 - 03/21/2015 11:47 PM - shevegen (Robert A. Heiler)

I think ruby warnings do not start with capitalized character such as in "The" :)

By the way your two examples are not equivalent right? Because in the first example
you use either a local variable, or a method called foo, and in the second you
use an instance variable called @foo

#3 - 04/18/2015 07:41 PM - yuki24 (Yuki Nishijima)

It wasn't so hard to implement this in pure Ruby, the last example doesn't work, though.

```
# -*- coding: utf-8 -*-
class NoMethodError
  REPL_LABELS = {
    "irb_binding" => -> { Readline::HISTORY.to_a.last },
    "__pry__"     => -> { ::Pry.history.to_a.last }
  }

  def to_s
    msg = super
    msg << "\n\n".freeze
    msg << "  #{code}\n"
    msg << "  #{marker}\n"
  rescue
    super
  end

  def code
    # Memoize since the #backtrace_locations method creates a lot of objects.
    @code ||= begin
      loc = backtrace_locations.first

      if REPL_LABELS.include?(loc.label)
        REPL_LABELS[loc.label].call
      elsif File.exist?(loc.absolute_path)
        File.open(loc.absolute_path) do |file|
          file.detect { file.lineno == loc.lineno }
        end
      end
    end.to_s.strip
  end

  def marker
    code.strip.gsub(/(.*)(#{name}).*/) do
      (" ".freeze * $1.size.pred) + '^'.freeze + ("‾".freeze * $2.size)
    end if !code.empty?
  end
end

# works
e = 1.foo rescue $!
puts e
# => undefined method `foo' for 1:Fixnum
#
#  e = 1.foo rescue $!
#       ^‾‾‾

# works
e = 1.send(:bar) rescue $!
puts e
# => undefined method `bar' for 1:Fixnum
#
#  e = 1.send(:bar) rescue $!
#             ^‾‾‾

# works
@foo = ->{ true }
e = (@foo.call && @bar.call) rescue $!
puts e
# => undefined method `call' for nil:NilClass
#
#  e = (@foo.call && @bar.call) rescue $!
#                        ^‾‾‾‾

# doesn't work
e = (@something.call && @else.call) rescue $!
puts e
# => undefined method `call' for nil:NilClass
#
#  e = (@something.call && @else.call) rescue $!
#                               ^‾‾‾‾
```

#4 - 12/01/2017 04:02 PM - olivierlacan (Olivier Lacan)

parkr (Parker M) wrote:

> This new syntax is a pretty sizable divergence from traditional Ruby error reporting.
> Yes, and I would consider that a good thing.
>
> At the moment, errors generally (always?) have an accompanying stack trace:

I don't think this suggestion implies removing the stack trace, which is still valuable context.

> It indicates the line, but excludes the column. It's uniform for all errors, so users seeing these stack traces know in what file, what line, and what method (the binding?) the error occurred.

There's no reason we can't work incrementally and slowly improve each major error class to include clearer indications as to what particular portion of a line caused an error.

> If the proposed format were implemented, would it only be for NoMethodError's, or would it apply to ArgumentError's or TypeError's, and so on?
>
> Would the column number suffice? So your example would yield:

Do you often identify pieces of a line of code by column number or based on the actual code keywords? I'm not trying to be snarky, but columns are for machine and text editors, not humans. This wouldn't be very consistent with Ruby's focus on programmer-friendliness to do half the work (find the column) and let programmers do the rest.

shevegen (Robert A. Heiler) wrote:

> I think ruby warnings do not start with capitalized character such as in "The" :)

Agreed. method 'call' is undefined on receiver nil:NilClass makes more sense and to me is more legible than the current undefined method 'call' for nil:NilClass.

yuki24 (Yuki Nishijima) wrote:

> It wasn't so hard to implement this in pure Ruby, the last example doesn't work, though.

Thank you so much for writing this proof of concept.

The last example works fine for me though:

```
irb(main):043:0> @foo = nil
=> nil
irb(main):044:0> @bar = nil
=> nil
irb(main):045:0> @foo.call && @bar.call
NoMethodError: undefined method `call' for nil:NilClass

  @foo.call && @bar.call
                   ^‾‾‾‾

    from (irb):45
    from /Users/olivierlacan/.rbenv/versions/2.4.0/bin/irb:11:in `<main>'
```

#5 - 07/24/2018 02:02 AM - olivierlacan (Olivier Lacan)

I just ran into a SyntaxError on heapy today (in Ruby 2.1.6) and it did exactly what I wish would occur for NoMethodError:

```
$ heapy read ruby-heap-2018-07-24\ 01\:29\:05.dump
/Users/olivierlacan/.rbenv/versions/2.1.6/lib/ruby/site_ruby/2.1.0/rubygems/core_ext/kernel_require.rb:55:in `
require': /Users/olivierlacan/.rbenv/versions/2.1.6/lib/ruby/gems/2.1.0/gems/heapy-0.1.3/lib/heapy/alive.rb:13
3: syntax error, unexpected '.' (SyntaxError)
...lf.address_to_object(address)&.inspect || "object not traced...
...                               ^
    from /Users/olivierlacan/.rbenv/versions/2.1.6/lib/ruby/site_ruby/2.1.0/rubygems/core_ext/kernel_require.r
b:55:in `require'
    from /Users/olivierlacan/.rbenv/versions/2.1.6/lib/ruby/gems/2.1.0/gems/heapy-0.1.3/lib/heapy.rb:69:in `<t
op (required)>'
    from /Users/olivierlacan/.rbenv/versions/2.1.6/lib/ruby/site_ruby/2.1.0/rubygems/core_ext/kernel_require.r
b:127:in `require'
    from /Users/olivierlacan/.rbenv/versions/2.1.6/lib/ruby/site_ruby/2.1.0/rubygems/core_ext/kernel_require.r
b:127:in `rescue in require'
    from /Users/olivierlacan/.rbenv/versions/2.1.6/lib/ruby/site_ruby/2.1.0/rubygems/core_ext/kernel_require.r
b:40:in `require'
    from /Users/olivierlacan/.rbenv/versions/2.1.6/lib/ruby/gems/2.1.0/gems/heapy-0.1.3/bin/heapy:4:in `<top (
required)>'
    from /Users/olivierlacan/.rbenv/versions/2.1.6/bin/heapy:22:in `load'
    from /Users/olivierlacan/.rbenv/versions/2.1.6/bin/heapy:22:in `<main>'
```

If we can do this for SyntaxError, I don't see why we shouldn't do it for NotMethodError as well. Is anyone familiar with how the SyntaxError feedback is generated?