Ruby master - Bug #8982

NoMethodError#message produces surprising output when #inspect is defined on an anonymous class

10/03/2013 03:23 PM - myronmarston (Myron Marston)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.0.0p247 (2013-06-27 revision 41674) [x86_64-darwin12.4.0]	**Backport:**	

Description

=begin
Given the following script:

```
def raise_no_method_error_for_anonymous_class_with_inspect(&block)
klass = Class.new do
define_method(:inspect, &block)
end

begin
instance = klass.new
puts "#inspect output: #{instance.inspect} (#{instance.inspect.length} chars)"
instance.undefined_method
rescue NoMethodError => e
puts e.message
end

puts
end

raise_no_method_error_for_anonymous_class_with_inspect do
"#"
end

raise_no_method_error_for_anonymous_class_with_inspect do
""
end

raise_no_method_error_for_anonymous_class_with_inspect do
"#"
end

raise_no_method_error_for_anonymous_class_with_inspect do
"#"
end
```

It produces the following output:

```
#inspect output: # (19 chars)
undefined method `undefined_method' for #

#inspect output:  (18 chars)
undefined method `undefined_method' for :#Class:0x1017270e8

#inspect output: # (65 chars)
undefined method `undefined_method' for #

#inspect output: # (66 chars)
undefined method `undefined_method' for #<#Class:0x1017266e8:0x101726698>
```

There are two surprising things here:

- It matters whether or not the first character in my inspect is a #. If it's not, ruby appends the class's #inspect output to it.
- It matters how long my inspect string is. If it's less than 66 characters, it's used; if it's more than 65, it's discarded, and the default anonymous #inspect is used instead.

Both of these things are extremely surprising and seem very arbitrary and inconsistent.

I brought this up on ruby parley and charliesome (Charlie Somerville) was kind enough to point me to the code that's the source of this issue:

(())

So it looks intentional, but I think this is a bug.
=end

History

#1 - 10/18/2013 12:13 AM - nobu (Nobuyoshi Nakada)

- Description updated

myronmarston (Myron Marston) wrote:

> - It matters whether or not the first character in my inspect is a #. If it's not, ruby appends the class's #inspect output to it.

'#' at the beginning is assumed the string is same as Object#inspect, otherwise its class name is appended since the string may not represent the class.

> - It matters how long my inspect string is. If it's less than 66 characters, it's used; if it's more than 65, it's discarded, and the default anonymous #inspect is used instead.

If the inspect string is too long, just ignore it and use default conersion method

#2 - 10/18/2013 01:57 PM - alexeymuranov (Alexey Muranov)

nobu (Nobuyoshi Nakada) wrote:

> '#' at the beginning is assumed the string is same as Object#inspect, otherwise its class name is appended since the string may not represent the class.
>
> If the inspect string is too long, just ignore it and use default conersion method

Looks like defensive programming to me.

#3 - 10/27/2013 11:59 AM - myronmarston (Myron Marston)

nobu (Nobuyoshi Nakada) wrote:

> myronmarston (Myron Marston) wrote:
>
> > - It matters whether or not the first character in my inspect is a #. If it's not, ruby appends the class's #inspect output to it.
>
> '#' at the beginning is assumed the string is same as Object#inspect, otherwise its class name is appended since the string may not represent the class.

Ruby itself provides many classes whose definition of inspect does not include #. Why cannot it not allow me to do the same? Also, # at the beginning of the string being taken as a sign that it's the same string as Object#inspect would produce seems incredibly broken. And I'm not sure why you care? Why can't I define how my class is represented as a string? Isn't that the point of inspect?

> - It matters how long my inspect string is. If it's less than 66 characters, it's used; if it's more than 65, it's discarded, and the default anonymous #inspect is used instead.

> If the inspect string is too long, just ignore it and use default conersion method

Many of ruby's built-in classes can produce longer inspect strings than this. (As an example:([1] * 1000).inspect). Ignoring what a user has defined for inspect seems incredibly surprising and would be considered a bug by every ruby programmer I know (well, every ruby programmer I've shown this issue to, at least).

IMO, Ruby should either retain complete control over how objects represent themselves strings, or allow users to define how objects are represented as strings...but giving us the illusion that we can define how objects represent themselves as strings, and then not actually allowing that, is the worst possible outcome.

And this isn't just theoretical: I spent a couple hours a few weeks ago scratching my head, trying to figure out why in the world my object's inspect wasn't working.

#4 - 09/12/2014 06:47 AM - myronmarston (Myron Marston)

So I've run into this issue again, and as it turns out, this is a much more widespread problem than I first thought. I originally thought that this was just a problem with anonymous classes, but that's not the case; now I'm running up against it in RSpec. We're trying to define RSpec::Core::ExampleGroup#inspect so that when users have a typo (or otherwise trigger a NoMethodError for a message sent to an example group) the example group is represented in the error message in a pretty, useful way, rather than having the obscure hex value:

https://github.com/rspec/rspec-core/issues/1590
https://github.com/rspec/rspec-core/pull/1687

As far as I can tell, it's impossible to achieve this behavior on MRI, and that really bums me out :(.

Any chance this can get fixed in Ruby 2.2?

#5 - 08/15/2019 11:56 PM - jeremyevans0 (Jeremy Evans)

- Backport deleted (1.9.3: UNKNOWN, 2.0.0: UNKNOWN)

- Status changed from Open to Feedback

Starting in Ruby 2.3, you can do this:

```
begin
instance = klass.new
puts "#inspect output: #{instance.inspect} (#{instance.inspect.length} chars)"
instance.undefined_method
rescue NoMethodError => e
puts "undefined method `#{e.name}' for #{e.receiver.inspect}"
end
```

Hopefully that provides a way to accomplish what you want. Do you think this is sufficient for your needs?

#6 - 10/21/2019 10:39 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Feedback to Closed