Ruby trunk - Bug #9967

`define_method(:name, &block)` breaks the use of the block on its own

06/20/2014 05:34 PM - myronmarston (Myron Marston)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.1.1p76 (2014-02-24 revision 45161) [x86_64-darwin12.0]	**Backport:**	2.0.0: UNKNOWN, 2.1: UNKNOWN

Description

In RSpec, we've run into a very odd issue. For some of the features in RSpec 3, we want to distinguish between block args that have defaults vs block args that don't. Normal blocks do not support this:

```
irb(main):001:0> Proc.new { |a| }.parameters
=> [[:opt, :a]]
irb(main):002:0> Proc.new { |a=1| }.parameters
=> [[:opt, :a]]
```

However, lambdas do:

```
irb(main):003:0> lambda { |a| }.parameters
=> [[:req, :a]]
irb(main):004:0> lambda { |a=1| }.parameters
=> [[:opt, :a]]
```

To accomplish this, we've implemented a proc_to_lambda method that returns a version of the proc that provides the parameters of it as if it was a lambda:

```
def proc_to_lambda(block)
  return block if block.lambda?

  obj = Object.new
  obj.define_singleton_method(:to_lambda, &block)
  obj.method(:to_lambda).to_proc
end
```

This implementation was inspired by Richard Schneeman's proc_to_lambda gem.

However, I've discovered a very odd behavior with this solution: the use of the block to define a method mutates the block so that it does not behave as normal, even if the defined method is never used. I've put together a gist that demonstrates the issue in isolation. Here's the same code:

```
puts RUBY_DESCRIPTION

class SuperClass
  def foo
    "foo"
  end
end

class SubClass < SuperClass
  def foo(arg)
    arg.bar { super() }
  end
end

class Arg
  def bar(&block)
    Class.new { define_method(:some_method, &block) } if $define_method
    yield
  end
```

```
end

print "Without define_method: "
puts SubClass.new.foo(Arg.new)

$define_method = true
print "With define_method: "
puts SubClass.new.foo(Arg.new)
```

This works fine on 1.8.7, but the second SubClass.new.foo(Arg.new) line breaks on 1.9.2, 1.9.3, 2.0 and 2.1 with a "super called outside of method" error:

```
   code  chruby 2.1.1
   code  ruby block_weirdness.rb
ruby 2.1.1p76 (2014-02-24 revision 45161) [x86_64-darwin12.0]
Without define_method: foo
With define_method: block_weirdness.rb:11:in `block in foo': super called outside of method (NoMet
hodError)
    from block_weirdness.rb:18:in `bar'
    from block_weirdness.rb:11:in `foo'
    from block_weirdness.rb:27:in `<main>'
   code  chruby 2.0
   code  ruby block_weirdness.rb
ruby 2.0.0p353 (2013-11-22 revision 43784) [x86_64-darwin12.5.0]
Without define_method: foo
With define_method: block_weirdness.rb:11:in `block in foo': super called outside of method (NoMet
hodError)
    from block_weirdness.rb:18:in `bar'
    from block_weirdness.rb:11:in `foo'
    from block_weirdness.rb:27:in `<main>'
   code  chruby 1.9.3
   code  ruby block_weirdness.rb
ruby 1.9.3p448 (2013-06-27 revision 41675) [x86_64-darwin12.4.0]
Without define_method: foo
With define_method: block_weirdness.rb:11:in `block in foo': super called outside of method (NoMet
hodError)
    from block_weirdness.rb:18:in `bar'
    from block_weirdness.rb:11:in `foo'
    from block_weirdness.rb:27:in `<main>'
   code  chruby 1.9.2
   code  ruby block_weirdness.rb
ruby 1.9.2p320 (2012-04-20 revision 35421) [x86_64-darwin12.4.0]
Without define_method: foo
With define_method: block_weirdness.rb:11:in `block in foo': super called outside of method (NoMet
hodError)
    from block_weirdness.rb:18:in `bar'
    from block_weirdness.rb:11:in `foo'
    from block_weirdness.rb:27:in `<main>'
   code  chruby 1.8.7
   code  ruby block_weirdness.rb
ruby 1.8.7 (2013-06-27 patchlevel 374) [i686-darwin12.4.0]
Without define_method: foo
With define_method: foo
```

This is very unexpected: there's no reason to think that using the block to define a method (but not using that method) should change how the block behaves on its own. It's also a regression since this worked in 1.8.7.

On a side note, if #9777 was implemented, we would use that, but I still think this is a bug on its own, regardless of our particular use case.

History

#1 - 06/21/2014 07:19 AM - JonRowe (Jon Rowe)

I tried to work around this bug by duplicating the proc with .dup but it seems that dup doesn't actually duplicate the proc. So I went digging through the C code and discovered that define_method actually does try to dup the proc with proc_dup(body) (https://github.com/ruby/ruby/blob/trunk/proc.c#L1748) but the proc dup code seems not do actually duplicate the block but just copy the pointers across (https://github.com/ruby/ruby/blob/trunk/proc.c#L106), in Ruby 1.8.7 (where this bug does not occur) the implementation of proc dup is

completely different and seems to actually dup the proc. (https://github.com/ruby/ruby/blob/v1_8_7_374/eval.c#L8596)