Ruby master - Bug #6832

Module#instance_method and Module#method_defined? act inconsistently w.r.t #respond_to_missing?

08/04/2012 03:39 PM - myronmarston (Myron Marston)

Status: Closed
Priority: Normal
Assignee: matz (Yukihiro Matsumoto)
Target version: 2.0.0
ruby -v: 1.9.3p194

Description
It's awesome that #respond_to_missing? allows Object#method to work for messages handled by #method_missing. However, I was surprised to discover that Module#instance_method and Module#method_defined? don't similarly take #respond_to_missing? into account. It seems very inconsistent.

Here's the behavior I'm seeing:

https://gist.github.com/3255162

In this example, I would expect Foo#method_defined?(:foo_bar) to return true, and I would expect Foo#instance_method(:foo_bar) to return an UnboundMethod that, when bound to a Foo instance, would use #method_missing to perform the method.

History

#1 - 11/03/2012 12:25 PM - mame (Yusuke Endoh)
- Status changed from Open to Assigned
- Assignee set to matz (Yukihiro Matsumoto)
- Target version set to 2.0.0

Let me summarize:

class Foo
  def method_missing(name, *args)
    return super unless name =~ /^foo_/
    puts name
  end

  def respond_to_missing?(name, include_private)
    super || name =~ /^foo_/
  end
end

Foo.new.method(:foo_bar).call #=> "foo_bar" (as expected)
p Foo.method_defined?(:foo_bar) #=> true expected, but actual false
p Foo.instance_method(:foo_bar) #=> UnboundMethod expected, but actual NameError

Matz, is this an intended behavior?

--
Yusuke Endoh mame@tsg.ne.jp

#2 - 11/03/2012 01:53 PM - claytrump (Clay Trump)
+1, at least for instance_method returning an UnboundMethod.

On Fri, Nov 2, 2012 at 11:25 PM, mame (Yusuke Endoh) mame@tsg.ne.jp wrote:

Issue #6832 has been updated by mame (Yusuke Endoh).

Status changed from Open to Assigned
Assignee set to matz (Yukihiro Matsumoto)
Target version set to 2.0.0

Let me summarize:
class Foo
  def method_missing(name, *args)
    return super unless name =~ /^foo_/
    puts name
  end

  def respond_to_missing?(name, include_private)
    super || name =~ /^foo_/
  end
end

Foo.new.method(:foo_bar).call #=> "foo_bar" (as expected)
p Foo.method_defined?(:foo_bar) #=> true expected, but actual false
p Foo.instance_method(:foo_bar) #=> UnboundMethod expected, but actual
NameError

Matz, is this an intended behavior?
--

Yusuke Endoh mame@tsg.ne.jp

Bug #6832: Module#instance_method and Module#method_defined? act inconsistently w.r.t #respond_to_missing?
https://bugs.ruby-lang.org/issues/6832#change-32279

Author: myronmarston (Myron Marston)
Status: Assigned
Priority: Normal
Assignee: matz (Yukihiro Matsumoto)
Category:
Target version: 2.0.0
ruby -v: 1.9.3p194

It's awesome that #respond_to_missing? allows Object#method to work for messages handled by #method_missing. However, I was surprised to discover that Module#instance_method and Module#method_defined? don't similarly take #respond_to_missing? into account. It seems very inconsistent.

Here’s the behavior I’m seeing:
https://gist.github.com/3255162

In this example, I would expect Foo#method_defined?(:foo_bar) to return true, and I would expect Foo#instance_method(:foo_bar) to return an UnboundMethod that, when bound to a Foo instance, would use #method_missing to perform the method.
--
http://bugs.ruby-lang.org/

--

#3 - 11/26/2012 09:21 AM - myronmarston (Myron Marston)

While I still think this is initially surprising behavior, I’ve thought about it some more and realized that there’s a big problem here. Consider this class:

class Foo
  def initialize(prefix)
    @prefix = prefix
  end

  def respond_to_missing?(name, include_private)
    name.to_s.start_with?(@prefix)
  end

  def method_missing(name, *args)
    return super unless name.to_s.start_with?(@prefix)
    puts name
  end
end

09/09/2021
There is no way for `Foo.instance_method` or `Foo.method_defined?` to take into account method-missing-handled messages because it depends on the state of the instances of this class. The example I posted above was a trivial example that didn't use any instance state, and so apparently should be able to work. Given the ambiguities that arise in these situations, I don't think it makes sense for `instance_method` and `method_defined?` to take `respond_to_missing?` into account.

#4 - 11/26/2012 09:22 AM - myronmarston (Myron Marston)
Is there a way to close the issue? I haven't used rubymine enough to figure out how to do that, but I would if I could.

#5 - 11/26/2012 09:54 AM - duerst (Martin Dürst)
- Status changed from Assigned to Closed

Closed on request of myronmarston.
(I'm not totally sure, but I think only people in the Assignee list can close issues.)