

Ruby master - Bug #11182

Refinement with alias causes strange behavior

05/26/2015 08:20 AM - ko1 (Koichi Sasada)

Status: Feedback	
Priority: Normal	
Assignee: ko1 (Koichi Sasada)	
Target version:	
ruby -v: 2.3dev	Backport: 2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN
Description The following script causes strange behavior. <pre>class C def foo p "C" end end module M refine C do def foo p "Refiend C" end end end class D < C alias bar foo end using M D.new.bar #=> t.rb:21:in `<main>': undefined method `bar' for #<D:0x29fbf58> (NoMethodError)</pre> <p>It seems strange.</p> <p>Maybe (1) C#foo or (2) M#C#foo should be called. But I'm not sure which is suitable.</p> <p>Previous versions:</p> <pre>ruby 2.0.0p606 (2014-11-28 revision 48636) [i386-mswin32_110] t.rb:9: warning: Refinements are experimental, and the behavior may change in future versions of Ruby! "C" ruby 2.1.5p312 (2015-03-10 revision 49912) [i386-mswin32_110] "C"</pre>	
Related issues:	
Related to Ruby master - Bug #13817: test/unit breaks Hash	Closed

Associated revisions

Revision 834d2bbe - 05/26/2015 01:16 PM - shugo (Shugo Maeda)

- vm_method.c (rb_alias): should resolve refined methods. [ruby-core:69360] [Bug #11182]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@50642 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 50642 - 05/26/2015 01:16 PM - shugo (Shugo Maeda)

- `vm_method.c (rb_alias)`: should resolve refined methods. [ruby-core:69360] [Bug #11182]

Revision 50642 - 05/26/2015 01:16 PM - shugo (Shugo Maeda)

- `vm_method.c (rb_alias)`: should resolve refined methods. [ruby-core:69360] [Bug #11182]

Revision 50642 - 05/26/2015 01:16 PM - shugo (Shugo Maeda)

- `vm_method.c (rb_alias)`: should resolve refined methods. [ruby-core:69360] [Bug #11182]

Revision 50642 - 05/26/2015 01:16 PM - shugo (Shugo Maeda)

- `vm_method.c (rb_alias)`: should resolve refined methods. [ruby-core:69360] [Bug #11182]

Revision 50642 - 05/26/2015 01:16 PM - shugo (Shugo Maeda)

- `vm_method.c (rb_alias)`: should resolve refined methods. [ruby-core:69360] [Bug #11182]

History

#1 - 05/26/2015 12:25 PM - shugo (Shugo Maeda)

- Status changed from Open to Assigned
- Assignee set to shugo (Shugo Maeda)

#2 - 05/26/2015 01:16 PM - shugo (Shugo Maeda)

- Status changed from Assigned to Closed

Applied in changeset r50642.

- `vm_method.c (rb_alias)`: should resolve refined methods. [ruby-core:69360] [Bug [#11182](#)]

#3 - 05/26/2015 01:22 PM - shugo (Shugo Maeda)

- Backport changed from 2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN to 2.0.0: DONTNEED, 2.1: DONTNEED, 2.2: REQUIRED

Koichi Sasada wrote:

Maybe (1) `C#foo` or (2) `M#C#foo` should be called. But I'm not sure which is suitable.

By `D.new.bar`, not `foo` but `bar` is searched in refinements, so `M#C#foo` cannot be found.
I've fixed alias to use `C#foo`.

#4 - 05/26/2015 06:59 PM - ko1 (Koichi Sasada)

By `D.new.bar`, not `foo` but `bar` is searched in refinements, so `M#C#foo` cannot be found.
I've fixed alias to use `C#foo`.

Why that? I assumed two scenarios.

(1) Invoke `C#foo`

- `D#bar` -> find a method entry (alias to `C#foo`)
- Invoke `C#foo`

(2) Invoke `M#C#foo`

- `D#bar` -> find a method entry (alias to `C#foo`)
- `C#foo` is refined by `M` so invke `M#C#foo`

Of course, if `D#bar` is refined, then the refining method should be invoked.

#5 - 05/26/2015 07:08 PM - ko1 (Koichi Sasada)

Simplify example:

```
class C
  def foo
    p "C"
  end
end

module M
  refine C do
    def foo
      p "Refiend C"
    end
  end
end

class C
  alias bar foo
end

C.new.foo
C.new.bar
using M
C.new.foo
C.new.bar

C.new.foo # C
C.new.bar # C
using M
C.new.foo # refined C
C.new.bar # C
```

I agree that invoking only "C" is reasonable because it can be direct pointer to C#foo.
However, I think it is also reasonable to call refined C because people can assume C#bar should be same as C#foo.

#6 - 05/27/2015 02:49 AM - shugo (Shugo Maeda)

Koichi Sasada wrote:

However, I think it is also reasonable to call refined C because people can assume C#bar should be same as C#foo.

If so, what should be printed by the last C.new.bar in the following example?

```
class C
  def foo
    p "C#foo"
  end
end

module M
  refine C do
    def foo
      p "Refiend C#foo"
    end

    def bar
      p "Refined C#bar"
    end
  end
end

class C
  alias bar foo
end

C.new.foo
C.new.bar
using M
C.new.foo
C.new.bar
```

#7 - 05/27/2015 05:05 AM - shugo (Shugo Maeda)

- Status changed from Closed to Open

- Backport changed from 2.0.0: DONTNEED, 2.1: DONTNEED, 2.2: REQUIRED to 2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN

#8 - 05/27/2015 05:15 AM - ko1 (Koichi Sasada)

- File 1.PNG added

- File 2.PNG added

I can consider two models (1) and (2) the following pictures show.

1.png

2.png

I believe the implementation uses (2) model.

For (1), calling `M::C#bar` is reasonable.

For (2), calling `M::C#bar` is reasonable.

Wow, both should be `M::C#bar`.

Example on #5, both models should call `M::C#bar`. But my comment at #5 "`C#bar` should be same as `C#foo`" is wrong.

#9 - 05/27/2015 05:33 AM - shugo (Shugo Maeda)

Koichi Sasada wrote:

I believe the implementation uses (2) model.

For (1), calling `M::C#bar` is reasonable.

For (2), calling `M::C#bar` is reasonable.

Wow, both should be `M::C#bar`.

Example on #5, both models should call `M::C#bar`. But my comment at #5 "`C#bar` should be same as `C#foo`" is wrong.

`M::C#bar` is not defined in Example #5, so the original `C#bar`, which is an alias of `C#foo`, should be called, shouldn't it?

#10 - 05/27/2015 05:36 AM - ko1 (Koichi Sasada)

- File 3.PNG added

- File 4.PNG added

My last explanation was wrong.

I wrote pictures again.

3.PNG

With model (1), it seems `C#foo` will be called.

With model (2), it seems `M::C#foo` will be called.

How about it?

#11 - 05/27/2015 05:12 PM - headius (Charles Nutter)

I will offer a quick opinion: refinements are an overlay on the method table, and therefore modifications to the method table should not reflect refinements. Put differently: the alias `bar foo` should call `C#foo` always, because there it is a modification to `D`'s method table.

We already established in past discussions about refinements that the class's actual method table (and methods that query or manipulate it) should *not* reflect refinements, since that would require them to be aware of the caller's scope. This is just another case of the same thing: creating an alias looks up in the target class/module's method table and adds a new entry. It's as simple as that.

#12 - 05/28/2015 08:03 AM - shugo (Shugo Maeda)

- File `alias_affected_by_original_refinement.diff` added

- Status changed from *Open* to *Feedback*

- Assignee changed from *shugo (Shugo Maeda)* to *ko1 (Koichi Sasada)*

Koichi Sasada wrote:

With model (1), it seems `C#foo` will be called.

With model (2), it seems `M::C#foo` will be called.

How about it?

I agree that both models are possible and attached a patch implementing model (2). However, even in the selector model, alias need not to be affected by refinements of the original method as implemented in trunk, where alias creates a direct link to the original method instead of a link to its selector.

I agree with Charles from past discussions.
At least the behavior should not be changed in 2.2.

#13 - 05/28/2015 09:22 AM - ko1 (Koichi Sasada)

- File 6.PNG added

- File 7.PNG added

I wrote pictures again for original examples D, C and M.

8.PNG

In this case, I think both are acceptable (calling C#foo directly or M::C#foo). Why you say "It must be C#foo"? Or model (2) is wrong model?

We already established in past discussions about refinements that the class's actual method table (and methods that query or manipulate it) should not reflect refinements, since that would require them to be aware of the caller's scope.

It is just MRI do (selector model, I wrote).
We choose this because we don't have any penalty if nobody use refinements.
(this is based on this assumption: only a few people use refinements)

#14 - 05/28/2015 09:57 AM - ko1 (Koichi Sasada)

Let's discuss with use cases. I don't have good example, but please assume we want to make new Hash class to support something like HashWithIndifferentAccess.

Okay, we need an extra Hash class doing something special.

```
class MyHash < Hash
  def initialize *args
    args.each{|(k, v)|self[k] = v}
  end

  alias iterate each
end

h = MyHash.new([:b, 1], [:a, 2])

h.each{|k, v| p [k, v]} # [:b, 1] [:a, 2]
```

Good.

Next time, I invented a nice refinement to iterate Hash contents by ordered.

```
module OrderedHashEach
  refine Hash do
    def each
      sort.each{|k, v|
        yield k, v
      }
    end
  end
end

using OrderedHashEach
{:b: 1, a: 2}.each{|k, v| p [k, v]} # [:a, 2], [:b, 1]
```

Excellent.

And of course, we can combine MyHash and OrderedHashEach.

```
module OrderedHashEach
  refine Hash do
    def each
      sort.each{|k, v|
        yield k, v
      }
    end
  end
end
```

```

    }
  end
end
end

class MyHash < Hash
  def initialize *args
    args.each{|(k, v)|self[k] = v}
  end
end

h = MyHash.new([:b, 1], [:a, 2])

using OrderedHashEach
h.each{|k, v| p [k, v]} # [:a, 2] [:b, 1]

```

Great.

Wait. each is not good terminology for our project. Use iterate instead.

```

module OrderedHashEach
  refine Hash do
    def each
      sort.each{|k, v|
        yield k, v
      }
    end
  end
end

class MyHash < Hash
  def initialize *args
    args.each{|(k, v)|self[k] = v}
  end

  alias iterate each
end

h = MyHash.new([:b, 1], [:a, 2])

using OrderedHashEach
h.iterate{|k, v| p [k, v]} # [:b, 1] [:a, 2]

```

It doesn't affect :(It is unexpected result for me.

Note that I don't want to change this specification. I want to know what is ideal specification.

This time, I made a scenario that we may want to use M::C#foo.

#15 - 05/28/2015 10:31 PM - headius (Charles Nutter)

- Assignee changed from ko1 (Koichi Sasada) to shugo (Shugo Maeda)

Alias does not redispach, so this is the result I'd expect. These are some of the edge cases that I don't think we can solve for everyone.

Bottom line is that method table changes are method table changes. The only way we could implement this the way you want would be to have aliased methods redispach to the original name, which has all sorts of really nasty problems (redispach from bottom class or current class? super reflects new name or old name?).

I don't think we can do it ko1's way without introducing many more significant problems (by changing what alias means).

#16 - 05/29/2015 02:06 AM - shugo (Shugo Maeda)

Koichi Sasada wrote:

Wait. each is not good terminology for our project. Use iterate instead.

```

module OrderedHashEach
  refine Hash do
    def each
      sort.each{|k, v|
        yield k, v
      }
    end
  end
end

```

```

end

class MyHash < Hash
  def initialize *args
    args.each{|(k, v)|self[k] = v}
  end

  alias iterate each
end

h = MyHash.new([:b, 1], [:a, 2])

using OrderedHashEach
h.iterate{|k, v| p [k, v]} # [:b, 1] [:a, 2]

```

It doesn't affect :(It is unexpected result for me.

If you need to refine iterate you should define OrderedHashIterate and use it instead.

Refinements are designed not to change behavior implicitly. If refinements are changed to affect aliases, someone might complain that refinements don't support local rebinding.

#17 - 11/05/2016 03:13 AM - shugo (Shugo Maeda)

- Assignee changed from shugo (Shugo Maeda) to gotoken (Kentaro Goto)

#18 - 11/05/2016 03:13 AM - shugo (Shugo Maeda)

- Assignee changed from gotoken (Kentaro Goto) to ko1 (Koichi Sasada)

#19 - 08/16/2017 03:25 AM - wanabe (_ wanabe)

- Related to Bug #13817: test/unit breaks Hash added

Files

1.PNG	38.7 KB	05/27/2015	ko1 (Koichi Sasada)
2.PNG	43.7 KB	05/27/2015	ko1 (Koichi Sasada)
4.PNG	38.5 KB	05/27/2015	ko1 (Koichi Sasada)
3.PNG	37.9 KB	05/27/2015	ko1 (Koichi Sasada)
alias_affected_by_original_refinement.diff	1.34 KB	05/28/2015	shugo (Shugo Maeda)
6.PNG	36.6 KB	05/28/2015	ko1 (Koichi Sasada)
7.PNG	40.7 KB	05/28/2015	ko1 (Koichi Sasada)