Ruby - Bug #13271

Clarifications on refinement spec

03/02/2017 09:38 PM - Gondolin (Damien Robert)

Status:	Closed		
Priority:	Normal		
Assignee:	shugo (Shugo Maeda)		
Target version:			
ruby -v:	ruby 2.4.0p0 (2016-12-24 revision 57164) [x86_64-linux]	**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

Consider the following code:

```
class Foo
  def foo
    "Foo#foo"
  end
  def bar
    "Foo#bar"
  end
end
class Bar < Foo
  def foo
    "Bar#foo -> "+super
  end
  def bar
    "Bar#bar -> "+super
  end
end

module R1
  def foo
    "R1#foo -> "+super
  end
  def bar
    "R1#bar -> "+super
  end
end
module R2
  def foo
    "R2#foo -> "+super
  end
  def bar
    "R2#bar -> "+super
  end
end

module M2
  refine Foo do
    include R2
    def foo
      "refinement:Foo@M2#foo -> "+super
    end
    def bar
      "refinement:Foo@M2#bar -> "+super
    end
  end
end

module M1
  include M2
  refine Foo do
```

```
    include R1
    def foo
      "refinement:Foo@M1#foo -> "+super
    end
  end
end

using M1
puts Foo.new.foo #refinement:Foo@M1#foo -> R1#foo -> Foo#foo
puts Foo.new.bar #R1#bar -> refinement:Foo@M2#bar -> R2#bar -> Foo#bar
puts Bar.new.foo #Bar#foo -> Foo#foo
puts Bar.new.bar #Bar#bar -> Foo#bar
```

I have several questions about the results.

1. As I was expecting, 'using M1' not only activate the refinements of M1 but also the refinements of the included module M2.
   In other word it behaves as if I had specified 'using M2; using M1'. This is what I was expecting but from reading the spec "(3) Activate refinements in the defined refinement table of mod" it looks like the spec only speak about the refined modules in M1.

2. As noted in the spec, refinement:Foo@M1 should behave as if its superclass was Foo, so the fact that Foo.new.foo does not go through the refinements of M2 was expected.
   However I find the result of 'Foo.new.bar' strange: refinement:Foo@M1 does not contain the bar method, but its included module R1 does. So should not the result be
   R1#bar -> Foo#bar, whithout going through the refinements methods of M2?
   The spec states "(3) If C has included modules, for these modules M If a method with name N found in the method table of M, return the method."
   But it does not stipulate what happen if we call super in one of these included method.

3. I am also surprised by the behaviour of Bar.new.foo and Bar.new.bar. According to the spec:
   "(7) If C has a direct superclass, search the method N as specified in "Normal method lookup" from Step 4, where C is the superclass."
   and Step 4 start by looking at the activated refinements.

   So I was expecting the results to go through Foo's refinement, so that for instance 'Bar.new.foo' would be
   Bar#foo -> refinement:Foo@M1#foo -> R1#foo -> Foo#foo

Associated revisions

Revision 476f9b638895f1550e75552c49ef169ef1adc008 - 12/03/2017 08:35 AM - shugo (Shugo Maeda)

Specify refinement inheritance by Module#include.

[ruby-core:79880] [Bug #13271]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@60992 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 476f9b63 - 12/03/2017 08:35 AM - shugo (Shugo Maeda)

Specify refinement inheritance by Module#include.

[ruby-core:79880] [Bug #13271]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@60992 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 03/13/2017 08:55 AM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

- Assignee set to shugo (Shugo Maeda)

#2 - 03/14/2017 07:38 AM - nobu (Nobuyoshi Nakada)

- Description updated

#3 - 03/15/2017 07:39 AM - shugo (Shugo Maeda)

Gondolin (Damien Robert) wrote:

> 1. As I was expecting, 'using M1' not only activate the refinements of M1 but also the refinements of the included module M2.
>   In other word it behaves as if I had specified 'using M2; using M1'. This is what I was expecting but from reading the spec
>   "(3) Activate refinements in the defined refinement table of mod" it looks like the spec only speak about the refined modules in M1.

The behavior of my first proposal was what you expected, but it was changed to reduce implicit effects of refinements.

> 1. As noted in the spec, refinement:Foo@M1 should behave as if its superclass was Foo, so the fact that Foo.new.foo does not go through the refinements of M2 was expected.
>   However I find the result of 'Foo.new.bar' strange: refinement:Foo@M1 does not contain the bar method, but its included module R1 does.
>   So should not the result be
>   R1#bar -> Foo#bar, whithout going through the refinements methods of M2?
>   The spec states "(3) If C has included modules, for these modules M If a method with name N found in the method table of M, return the method."
>   But it does not stipulate what happen if we call super in one of these included method.

This behavior seems strange, and may be a bug of the implementation.

> 1. I am also surprised by the behaviour of Bar.new.foo and Bar.new.bar. According to the spec:
>   "(7) If C has a direct superclass, search the method N as specified in "Normal method lookup" from Step 4, where C is the superclass."
>   and Step 4 start by looking at the activated refinements.
>
>   So I was expecting the results to go through Foo's refinement, so that for instance 'Bar.new.foo' would be
>   Bar#foo -> refinement:Foo@M1#foo -> R1#foo -> Foo#foo

refinement:Foo@M1 is not active in the definitions of Bar#foo and Bar#bar, so super in them
never invoke refinement:Foo@M1#foo.

#4 - 03/15/2017 08:30 AM - Gondolin (Damien Robert)

Thanks for the answers!

shugo (Shugo Maeda) wrote:

> The behavior of my first proposal was what you expected, but it was changed to reduce implicit effects of refinements.

So should I expect that in future ruby versions using M will not also activate the refinements of included and prepended modules of M?

> This behavior seems strange, and may be a bug of the implementation.

Ok!

> refinement:Foo@M1 is not active in the definitions of Bar#foo and Bar#bar, so super in them
> never invoke refinement:Foo@M1#foo.

Oh of course, the definition of Bar#foo is in a lexical scope where the refinements were not active, so since refinements work with the lexical scope they won't be activated there, thanks!

#5 - 03/16/2017 01:33 AM - shugo (Shugo Maeda)

Gondolin (Damien Robert) wrote:

> > The behavior of my first proposal was what you expected, but it was changed to reduce implicit effects of refinements.
>
> So should I expect that in future ruby versions using M will not also activate the refinements of included and prepended modules of M?

Yes.

> > This behavior seems strange, and may be a bug of the implementation.
>
> Ok!

I'll investigate further.

#6 - 03/16/2017 03:12 AM - shugo (Shugo Maeda)

shugo (Shugo Maeda) wrote:

> Gondolin (Damien Robert) wrote:
>
> > > The behavior of my first proposal was what you expected, but it was changed to reduce implicit effects of refinements.
> >
> > So should I expect that in future ruby versions using M will not also activate the refinements of included and prepended modules of M?
>
> Yes.

Sorry, it's wrong:(
The behavior was changed by Feature #8571.

#7 - 03/16/2017 09:20 PM - Gondolin (Damien Robert)

shugo (Shugo Maeda) wrote:

> The behavior was changed by Feature #8571.

Well I prefer that this feature remains so that's nice! But this probably means that the spec should be updated then?

#8 - 03/17/2017 12:35 AM - shugo (Shugo Maeda)

Gondolin (Damien Robert) wrote:

> shugo (Shugo Maeda) wrote:
>
> > The behavior was changed by Feature #8571.
>
> Well I prefer that this feature remains so that's nice! But this probably means that the spec should be updated then?

Yes. I'll update doc/syntax/refinements.rdoc after investigating 2) further.

#9 - 12/03/2017 08:35 AM - shugo (Shugo Maeda)

- Status changed from Assigned to Closed

Applied in changeset trunk|r60992.

Specify refinement inheritance by Module#include.

[ruby-core:79880] [Bug #13271]