Ruby master - Feature #16430

Resultion of constants in enclosing class/module affected by how nested classes/modules are declared

12/17/2019 08:13 PM - MikeVastola (Mike Vastola)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

I'm not sure if this is intentional (in which case it really isn't documented anywhere, and probably should be) or a bug, but imagine the following code:

```
# lib/a.rb
module A
  FOO = :BAR
end

# lib/a/b.rb
require_relative '../a'
module A::B
  def self.foo
    FOO
  end
end

# lib/a/c.rb
require_relative '../a'
module A
  module C
    def self.foo
      FOO
    end
  end
end
```

If I were to evaluate A::B.foo, I would trigger a NoMethodError (undefined method 'foo' for A::B:Module).
However, if I were to evaluate A::C.foo, I would get :BAR.

This was really confusing to debug because I've been writing the more compact syntax forever where possible without realizing it impacted variable resolution, and it seems kind of bizarre and counter-intuitive that it would work this way.

Also, playing with this a bit more, there are some really weird artifacts going on: apparently different methods within the same class/module can have different nestings depending on the context in which they were added to the class?

For example:

```
module A
  X = 1
end

module A::B
  X = 6
end

module A
  module B::C
    Y = 9
    Z = X + Y # 10
  end
end
```

```
module A::B
  module C
    N = X + Y # 15
  end
end
```

Related issues:

Is duplicate of Ruby master - Bug #11705: Namespace resolution in nested modu...	**Rejected**
Is duplicate of Ruby master - Feature #6810: `module A::B; end` is not equiva...	**Assigned**

History

#1 - 12/18/2019 12:08 AM - sawa (Tsuyoshi Sawada)

What is the purpose of writing the code in three different files? It only looks to me that it is making things more complicated.

A::B.foo and A::C.foo are undefined, and return a NoMethodError, as expected. If you meant A::B#foo and A::C#foo, then they do not make sense because A::B and A::C are modules, not classes.

#2 - 12/18/2019 01:18 AM - mame (Yusuke Endoh)

- Is duplicate of Bug #11705: Namespace resolution in nested modules with short syntax added

#3 - 12/18/2019 01:20 AM - mame (Yusuke Endoh)

- Status changed from Open to Rejected

It is by design. See "Scope" section of https://github.com/ruby/ruby/blob/master/doc/syntax/modules_and_classes.rdoc

#4 - 12/21/2019 02:23 AM - MikeVastola (Mike Vastola)

- Backport deleted (2.5: UNKNOWN, 2.6: UNKNOWN)

- ruby -v deleted (ruby 2.6.3p62 (2019-04-16 revision 67580) [x86_64-linux])

- Tracker changed from Bug to Feature

mame (Yusuke Endoh) wrote:

> It is by design. See "Scope" section of https://github.com/ruby/ruby/blob/master/doc/syntax/modules_and_classes.rdoc

Ok, I see that, but is this a good design? I would argue not and am therefore changing this to a feature request (I would ask it be re-opened.)

For me, this is literally the first instance -- having worked heavily with Ruby for the past 7 or so years -- wherein I have encountered something in the language that I found to be completely unintuitive.

Further, the documentation linked states "This style has the benefit of allowing the author to reduce the amount of indentation. Instead of 3 levels of indentation only one is necessary." this benefit is heavily attenuated if it is not possible to maintain the same scope as the more heavily indented syntax.

For the feature request, I would suggest either adding any parent namespaces to the nesting, or else (if there is concern about this being a breaking change) add a variant for the scope resolution operator in declaring a class/module to alter how nesting is interpreted (maybe A:::B -- a triple colon)?

Lastly, I would argue that there are far fewer (if any) use cases for the current design than for what I am suggesting.

#5 - 12/21/2019 02:25 AM - MikeVastola (Mike Vastola)

sawa (Tsuyoshi Sawada) wrote:

> What is the purpose of writing the code in three different files? It only looks to me that it is making things more complicated.

I was trying to offer a minimal (non)-working example. The benefit arises when there is more code and it makes sense to separate out features into their own files/nested modules.

> A::B.foo and A::C.foo are undefined, and return a NoMethodError, as expected. If you meant A::B#foo and A::C#foo, then they do not make sense because A::B and A::C are modules, not classes.

My mistake. I meant to write self.foo. I will fix.

#6 - 12/21/2019 02:27 AM - MikeVastola (Mike Vastola)

- Description updated

Fixed small bug in initial description.

#7 - 12/21/2019 03:50 AM - mame (Yusuke Endoh)

- Is duplicate of Feature #6810: `module A::B; end` is not equivalent to `module A; module B; end; end` with respect to constant lookup (scope) added

#8 - 12/21/2019 03:52 AM - mame (Yusuke Endoh)

If you want to request a feature, please research the old tickets. I've found #6810.

#9 - 12/21/2019 01:45 PM - MikeVastola (Mike Vastola)

mame (Yusuke Endoh) wrote:

> If you want to request a feature, please research the old tickets. I've found #6810.

Ok, thanks. I tried searching before submitting this, but I apparently didn't use the right terms.