

Ruby - Feature #1586

Including a module already present in ancestors should not be ignored

06/07/2009 10:18 AM - bitsweat (Jeremy Daer)

Status:	Rejected	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	3.0	

Description

=begin

The scenario:

- I include Foo in Numeric to provide #bar
- Some other library includes a module in Float to provide #bar
- So I include Foo in Float to use my #bar
- But including Foo in Float is ignored since it's already in the ancestor chain

I think it should be added to the ancestor chain, even if it's already present, since I may want to override some other method earlier in the ancestor chain.

Including a module already included in a superclass is ignored

```
module Foo; end
=> nil
class Numeric; include Foo; end
=> Numeric
Float.ancestors
=> [Float, Precision, Numeric, Foo, Comparable, Object, Kernel]
class Float; include Foo; end
=> Float
Float.ancestors
=> [Float, Precision, Numeric, Foo, Comparable, Object, Kernel]
```

Reversing the order of inclusion works as expected

```
module Foo; end
=> nil
class Float; include Foo; end
=> Float
Float.ancestors
=> [Float, Foo, Precision, Numeric, Comparable, Object, Kernel]
class Numeric; include Foo; end
=> Numeric
Float.ancestors
=> [Float, Foo, Precision, Numeric, Foo, Comparable, Object, Kernel]
```

And so does including a dupe of the existing module in the subclass

```
module Foo; end
=> nil
class Numeric; include Foo; end
=> Numeric
Float.ancestors
=> [Float, Precision, Numeric, Foo, Comparable, Object, Kernel]
class Float; include Foo.dup; end
=> Float
Float.ancestors
```

```
=> [Float, #Module:0x19bcd40, Precision, Numeric, Foo, Comparable, Object, Kernel]
=end
```

Related issues:

Related to Ruby - Bug #5236: Including a module in a superclass after it has ...	Closed	08/27/2011
Related to Ruby - Bug #3351: stack overflow on super	Closed	
Related to Ruby - Feature #9112: Make module lookup more dynamic (Including m...	Closed	
Has duplicate Ruby - Bug #8066: Inconsistency in ancestors chain	Rejected	03/10/2013

History

#1 - 06/07/2009 10:22 AM - nobu (Nobuyoshi Nakada)

- Category set to core
- Assignee set to matz (Yukihiro Matsumoto)
- Target version set to 3.0

#2 - 06/07/2009 10:48 AM - RickDeNatale (Rick DeNatale)

=begin

Actually, for a while, back in 2006, Ruby 1.9 (in its experimental form) used to do just what this ticket asks for:

<http://talklikeaduck.denhaven2.com/2006/10/09/a-subtle-change-to-mixin-semantics-in-ruby-1-9>

However, that change got reverted.

I asked Matz why at RubyConf 2007, and documented our conversation

<http://talklikeaduck.denhaven2.com/2007/11/03/a-chat-with-matz-classs-variable-reversion-and-a-mystery-explained>

The problem is that MRI, and I guess YARV doesn't keep track of where in the chain of classes, and module proxies it found the currently executing method, so super is implemented by doing a method search starting with the klass of self, and proceeding until the method is found a SECOND time. With this implementation it's easier to turn module re-inclusion into a nop than to deal with the consequences.

=end

#3 - 06/07/2009 12:30 PM - bitsweat (Jeremy Daer)

=begin

Fascinating. Thanks for the history behind this, Rick.

Despite the implementation difficulties, I'd like to see this choice revisited for a future Ruby. I consider it a bug.

=end

#4 - 09/14/2010 04:27 PM - shyouhei (Shyouhei Urabe)

- Status changed from Open to Assigned

#5 - 10/18/2011 09:16 AM - naruse (Yui NARUSE)

- Project changed from Ruby to 14
- Category deleted (core)
- Target version deleted (3.0)

#6 - 10/23/2011 05:21 PM - naruse (Yui NARUSE)

- Project changed from 14 to Ruby

#7 - 02/14/2012 09:30 PM - mame (Yusuke Endoh)

- Status changed from Assigned to Rejected

I'm rejecting this feature ticket because no progress has been made for a long time. See [\[ruby-core:42391\]](#).

This is indeed a hard issue.

There are two approaches to make it consistent:

- A) prohibit multiple appearances of one module in ancestors
- B) permit multiple appearances

I guess matz prefers A to B, so B will not be admissible. (just my guess, though) The current behavior aims A, but is indeed incomplete.

In casual use case, however, module inclusion is used statically, i.e., used when a class is first defined.
Thus, no actual problem is caused in practical case, I think.

In addition, A will be very hard to implement completely. Even if it is possible, I'm afraid that the behavior will be bug-prone (for both user code and ruby impl), because it means that an already-included module may be later removed from ancestors, unexpectedly.

Thus, I think that we can not fix this elegantly and will not change the current behavior. But we might have to do something if there is any *concrete* scenario in that the current behavior causes a trouble. Let us know such a scenario if you have.

--

Yusuke Endoh mame@tsg.ne.jp

#8 - 02/26/2012 05:19 AM - ko1 (Koichi Sasada)

- *Category set to core*
- *Status changed from Rejected to Open*
- *Target version set to 2.0.0*

I want to change this behavior as permitting multiple modules in an ancestors list (option B).

Can I do it? (If I can, I'll implement it after this Apr).

#9 - 02/26/2012 05:35 AM - marcandre (Marc-Andre Lafortune)

Koichi Sasada wrote:

I want to change this behavior as permitting multiple modules in an ancestors list (option B).

Can I do it? (If I can, I'll implement it after this Apr).

That would be great.

I'm assuming it be allowed to include a module multiple times in a row too?

```
module M
  def to_s
    "*" + super + "*"
  end
end

3.times{ Fixnum.prepend(M) }
42.to_s # => "****42***"
```

#10 - 03/18/2012 06:46 PM - shyouhei (Shyouhei Urabe)

- *Status changed from Open to Assigned*

#11 - 06/24/2012 03:20 AM - marcandre (Marc-Andre Lafortune)

Has this feature been accepted by Matz? Or else, is anyone producing a one minute slide-show?

#12 - 07/01/2012 08:53 AM - claytrump (Clay Trump)

- *File inclusion.pdf added*

I guess I'll do it. Hopefully this slide can be useful.

--

#13 - 07/01/2012 10:44 AM - mame (Yusuke Endoh)

Hi Clay and MarcAndre,

I received your slide. Thanks!

--

Yusuke Endoh mame@tsg.ne.jp

#14 - 07/23/2012 10:37 PM - mame (Yusuke Endoh)

- Assignee changed from matz (Yukihiko Matsumoto) to ko1 (Koichi Sasada)

Jeremy Kemper and Clay Trump,

This proposal was discussed as two separate ones:

1. allow multiple inclusion
2. propagate when a module includes a module

Both have been (basically) accepted. Congrats!

For part (1), matz said that he had tried to implemented this feature for 1.9.0, but had given up due to a bug of YARV ([#3351](#)). Ko1 will challenge to fix the bug.

For part (2), the behavior was designed because matz had no idea to implement it effectively. However, after the discussion, we concluded it was possible to implement the feature effectively. Then, matz said he was happy to change the current behavior as a "bug". Ko1 will challenge to implement it.

Note that these decisions may be cancelled if ko1 finds any (practical or technical) significant problem.

--

Yusuke Endoh mame@tsg.ne.jp

#15 - 10/27/2012 04:47 AM - ko1 (Koichi Sasada)

- Priority changed from Normal to 5

Nobu, could you help to implement (1)?

I'm not sure (2) can get 2.0 release. Sorry.

#16 - 10/27/2012 05:23 AM - claytrump (Clay Trump)

I'm looking forward to consistent module inclusion :-)

I should have talked about Marc-andre's example with multiple inclusion at the same level (same class/module/object), coz I don't think it should be done this way. The example was:

```
module M
  def to_s
    "" + super + ""
  end
end

3.times{ Fixnum.prepend(M) }
42.to_s # => ""42"" would be best, not ""*42*""
```

My reasoning is that otehrwise it makes it hard to reload a Ruby class definition. When debugging, it is frequent that one modifies a method definition, reloads the whole file and tests away. It would not be practical if any +include+ or +prepend+ in the class definition induced a change in the ancestry.

If someone really wants multiple inclusion *at the same level*, it can be achieved by dupping the module.

So I believe that a module should only be included once per class/module/object. Same thing for prepending. Prepending and including

should be independent though.

```
3.times{ Fixnum.prepend(M) }
2.times{ Fixnum.include(M) }
Numeric.prepend(M)
Fixnum.ancestors # => [M, Fixnum, M, M, Numeric, ...], not [M, M, M,
```

Fixnum, M, M, M, Numeric, ...]

Is that the plan? Was this brought up when discussing with Matz about this?

#17 - 11/24/2012 09:30 AM - mame (Yusuke Endoh)

ko1, what's the status?

--

Yusuke Endoh mame@tsq.ne.jp

#18 - 11/26/2012 09:01 AM - ko1 (Koichi Sasada)

- Assignee changed from ko1 (Koichi Sasada) to nobu (Nobuyoshi Nakada)

nobu, could you check it?

#19 - 11/30/2012 07:04 PM - nobu (Nobuyoshi Nakada)

- Priority changed from 5 to Normal

- Target version changed from 2.0.0 to 2.6

This would cause compatibility issue, in some cases, when a module is included twice but it expects it never get called twice or more by super. Or, if a method of the module does not call super, the super calls stops unexpectedly.

One idea is to introduce a method of Module which tells a module is multi-time includable.

#20 - 11/30/2012 07:05 PM - nobu (Nobuyoshi Nakada)

- Assignee changed from nobu (Nobuyoshi Nakada) to matz (Yukihiro Matsumoto)

#21 - 11/30/2012 11:43 PM - marcandre (Marc-Andre Lafortune)

As Clay said, I think the example I gave previously is not the way to go.

The important aspect is the ability to include a module at different levels in the hierarchy.

If $C < B < A$, we should be able to include a module M for each of A, B and C, but only once for each of them.

There should be no incompatibility this way.

#22 - 12/02/2012 01:14 AM - matz (Yukihiro Matsumoto)

- Target version changed from 2.6 to 3.0

When I made this change in early 1.9, it caused huge incompatibility. I had to fix many bundled programs. So it shouldn't be a minor change.

Matz.

#23 - 12/06/2012 03:17 AM - marcandre (Marc-Andre Lafortune)

matz (Yukihiro Matsumoto) wrote:

When I made this change in early 1.9, it caused huge incompatibility. I had to fix many bundled programs. So it shouldn't be a minor change.

When you did this, was it possible to include a module more than once for the same class, or simply possible once for any class/module, even when it present somewhere else in the ancestor chain? (See <http://bugs.ruby-lang.org/issues/1586#note-16> for an example)

I believe that allowing inclusion of a module once per module is the most useful and would not cause incompatibility.

#24 - 12/23/2012 08:43 PM - Anonymous

I join my voice with Marc-Andre. (More precisely, a module in the "is kind of?" sense should be included or not included only once, globally. But tweaking the method lookup sequence is imaginable.)

#25 - 12/24/2012 07:01 PM - matz (Yukihiro Matsumoto)

My change at the beginning of 1.9 was include module once for a class/module, as you described, and still it caused incompatibility. I had to change hundreds of lines in the standard libraries. So I think this change is good but we need to make it in proper timing.

Matz.

#26 - 10/08/2014 03:58 AM - matz (Yukihiro Matsumoto)

- *Related to Feature #9112: Make module lookup more dynamic (Including modules into a module after it has already been included) added*

#27 - 10/20/2017 01:29 AM - mame (Yusuke Endoh)

When is proper time? Ruby 3.0? :-)

#28 - 11/28/2017 03:54 AM - marcandre (Marc-Andre Lafortune)

matz (Yukihiro Matsumoto) wrote:

So I think this change is good but we need to make it in proper timing.

mame (Yusuke Endoh) wrote:

When is proper time? Ruby 3.0? :-)

As soon as Nobu can code it?

I'll note that we can already prepend a module more than once:

```
Base = Class.new
Foo = Class.new(Base)
M = Module.new; N = M.dup
Base.prepend M; Base.include N
Foo.prepend M; Foo.include N
Foo.ancestors # => [M, Foo, M, Base, N, Object, Kernel, BasicObject]
```

#29 - 11/29/2017 07:14 AM - matz (Yukihiro Matsumoto)

- *Status changed from Assigned to Rejected*

The incompatibility that will be caused by the change is intolerable. Any attempt to address the change will make the language far more complex than it currently is.

I have to reject.

Matz.

#30 - 11/29/2017 04:30 PM - marcandre (Marc-Andre Lafortune)

Thank you for the consideration.

May I add two observations?

1. prepend had similar behavior (ignoring already prepended modules) in 2.0 and 2.1. It was changed in 2.2 and I don't remember incompatibility problems.
2. If incompatibility is actually an issue, could we detect cases where we ignore wrongly an include and issue a warning, and make the changes in the next version?

Files

inclusion.pdf	56.4 KB	07/01/2012	claytrump (Clay Trump)
---------------	---------	------------	------------------------