Ruby - Bug #11878

Comparison of prepended modules

12/26/2015 02:44 PM - sawa (Tsuyoshi Sawada)

Status:	Rejected		
Priority:	Normal		
Assignee:	matz (Yukihiro Matsumoto)		
Target version:			
ruby -v:	2.3.0p0 (2015-12-25 revision 53290) [x86_64-linux]	**Backport:**	2.0.0: REQUIRED, 2.1: REQUIRED, 2.2: REQUIRED, 2.3: REQUIRED

Description

Including module B to class/module A gives the following results (as expected):

```
module A; end
module B; end
A.include B
A < B # => true
B < A # => false
A <=> B # => -1
```

And prepending module C to A gives the following results:

```
module C; end
A.prepend C
A < C # => true
C < A # => nil
A <=> C # => -1
```

It looks like including and prepending almost do not make difference with respect to module comparison, i.e., A < B and A < C are the same, and A <=> B and A <=> C are the same. However, then, the difference between B < A and C < A stands out unexplained. I suppose this is a bug. If C < A were to return false, then it would be at least consistent.

However, if that was what was intended, then at least to me, it is strange. In that case, I would like to make this a feature request. I would rather expect:

```
A < C # => false
C < A # => true
A <=> C # => 1
```

Associated revisions

Revision a974041b0ef744c102b2b15ff30b91b4b0ea8a1c - 12/30/2015 12:58 AM - nobu (Nobuyoshi Nakada)

object.c: fix prepend cmp

- object.c (rb_class_inherited_p): search the corresponding
  ancestor to prepended module from prepending class itself.
  [ruby-core:72493] [Bug #11878]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@53380 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision a974041b - 12/30/2015 12:58 AM - nobu (Nobuyoshi Nakada)

object.c: fix prepend cmp

- object.c (rb_class_inherited_p): search the corresponding
  ancestor to prepended module from prepending class itself.
  [ruby-core:72493] [Bug #11878]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@53380 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 12/28/2015 06:46 PM - marcandre (Marc-Andre Lafortune)

- Assignee set to nobu (Nobuyoshi Nakada)

Indeed.

#2 - 12/30/2015 12:59 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset r53380.

---

object.c: fix prepend cmp

- object.c (rb_class_inherited_p): search the corresponding
  ancestor to prepended module from prepending class itself.
  [ruby-core:72493] [Bug #11878]

#3 - 12/30/2015 01:01 AM - nobu (Nobuyoshi Nakada)

- Description updated

- Backport changed from 2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN to 2.0.0: REQUIRED, 2.1: REQUIRED, 2.2: REQUIRED, 2.3: REQUIRED

#4 - 12/30/2015 04:28 AM - sawa (Tsuyoshi Sawada)

As far as I see the revision r53380 by Nakada san, it looks like my feature proposal part was rejected. Is this the case?

#5 - 12/30/2015 07:36 AM - nobu (Nobuyoshi Nakada)

Making prepending class an ancestor of prepended module?
It feels strange a little, to me.

#6 - 12/30/2015 09:16 AM - sawa (Tsuyoshi Sawada)

I thought that the ordering relation among modules/classes represents the method call priority. A < B means that method look-up first looks in A, then B; smaller module/class has higher priority. If so, since a module prepended to a class has higher priority than the class, the module should be smaller (<) than the class. Is my interpretation wrong?

#7 - 04/13/2016 07:02 AM - naruse (Yui NARUSE)

- Status changed from Closed to Assigned

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

#8 - 12/23/2019 03:57 PM - mame (Yusuke Endoh)

- Status changed from Assigned to Rejected

The current behavior is consistent. The rdoc of Module#< says:

```
call-seq:
  mod < other   ->  true, false, or nil

Returns true if <i>mod</i> is a subclass of <i>other</i>.
```

Note that subclass is not directly related to the method lookup order. Consider the following example:

```
module M; end
class C; prepend M; end

C.new.is_a?(M) #=> true
```

This means C is a subclass of M. So, C < M should return true. Note that, in terms of method lookup, M has a higher priority than 'C'.

#9 - 12/23/2019 04:38 PM - Eregon (Benoit Daloze)

- Status changed from Rejected to Open

At the very least it's inconsistent with the order of Module#ancestors:

```
module M; end
class C; prepend M; end

> C.ancestors
```

```
=> [M, C, Object, JSON::Ext::Generator::GeneratorMethods::Object, PP::ObjectMixin, Kernel, BasicObject]
> C < Object
=> true
> M < C
=> false
> C < M
=> true
```

I think no user expects that the "subclass relation" is different than the order of ancestors, isn't it? (and why would it need to be?)
Also, the documentation says nothing about modules or this ad-hoc order which nothing else seems to use.

I think this is a bug and I'd like matz's ruling.

#10 - 12/23/2019 04:41 PM - Eregon (Benoit Daloze)

Re obj.is_a?(mod) we can consider it as obj.singleton_class.ancestors.include?(mod) or even obj.class.ancestors.include?(mod) in this case.
That's consistent with ancestors.

#11 - 12/24/2019 07:19 AM - mame (Yusuke Endoh)

> I think no user expects that the "subclass relation" is different than the order of ancestors, isn't it?

Regardless whether users know or not, they are actually different. Consider:

```
module M; end
class C; prepend M; end
class D; include M; end
```

If M is a subclass of C, D is a subclass of C. Both A.prepend(B) and A.include(B) make A to be a subclass of B.

Note that the order of Module#ancestors is not specified; the rdoc says nothing.

(IMO, Module#prepend is a very bad thing that makes the object system complicated.)

#12 - 01/16/2020 08:43 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

For the code like below:

```
module A; end

module I
  include A
end

p A < I #=> false
p A > I #=> true

module P
  prepend A
end

# current: same as include
p A < P #=> false
p A > P #=> true
```

A > P does not mean P is a subclass of A, but P *includes* the method sets defined in A. So the current behavior should not be changed.

Matz.