Ruby trunk - Bug #12834

`prepend` getting prepended even if it already exists in the ancestors chain

10/13/2016 10:48 AM - ndn (Nikola Nenkov)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	2.3.1	**Backport:**	2.3: UNKNOWN

Description

```
module M; end
class A; prepend M; end
class B < A; prepend M; end

B.ancestors # => [M, B, M, A, Object, Kernel, BasicObject]
```

Even though I find this behaviour to be more intuitive, it is inconsistent with Module#include and is potentially breaking.
I didn't see a mention on the [release notes] and the [documentation] is now outdated.

History

#1 - 10/13/2016 03:26 PM - znz (Kazuhiro NISHIYAMA)

Module#include can make multiple times module existence in ancestors.

```
module M; end
class A; end
class B < A; include M; end
class A; include M; end
B.ancestors #=> [B, M, A, M, Object, Kernel, BasicObject]
```

#2 - 10/13/2016 03:42 PM - ndn (Nikola Nenkov)

My point was that whether or not a module will get prepended can have an actual impact, yet it got changed between versions without an explicit notice. Given that the behaviour is documented as opposed to left undefined, I thought that this is a regression.

#3 - 03/02/2017 01:14 AM - ioquatix (Samuel Williams)

I feel like this is a bug, and should be fixed. Prepending the same thing multiple times should be a warning IMHO. I'm not even sure what situation this is useful.

```
2.4.0 :001 > module M
2.4.0 :002?>   end
 => nil
2.4.0 :003 > module N
2.4.0 :004?>   end
 => nil
2.4.0 :005 > class C
2.4.0 :006?>   prepend M, N, M
2.4.0 :007?>   end
 => C
2.4.0 :008 > C.included_modules
 => [N, M, Kernel]
2.4.0 :009 > D = C.dup.prepend(M)
 => D
2.4.0 :010 > D.included_modules
 => [N, M, Kernel]
2.4.0 :011 > class E < D
2.4.0 :012?>   prepend M
2.4.0 :013?>   end
 => E
2.4.0 :014 > E.included_modules
 => [M, N, M, Kernel]
2.4.0 :015 >
```

#4 - 04/18/2017 04:54 AM - shyouhei (Shyouhei Urabe)

We looked at this issue in yesterday's developer meeting.

While discussing, we learned that prepended ancestors are not linearizable using the C3 algorithm. When you prepend something, you normally want to override a method in a specific order. So a prepending module and its prepended counterpart must be adjacent in a superclass resolution. Thus, matz started thinking that if a module is prepended multiple times, that should appear more than once in a method chain.

#5 - 11/29/2017 04:40 PM - marcandre (Marc-Andre Lafortune)

shyouhei (Shyouhei Urabe) wrote:

> Thus, matz started thinking that if a module is prepended multiple times, that should appear more than once in a method chain.

Could you clarify?

For inclusion/prepending, there are two questions in my mind:
1) what happens if a module is included/prepended multiple times at different places in the hierarchy (e.g. in A and B, in the example above)
2) what happens if it is included/prepended multiple times at the same level (e.g. twice in A)

Was the discussion about (b) allowing multiple times at the same level?

#6 - 11/30/2017 12:53 AM - shyouhei (Shyouhei Urabe)

marcandre (Marc-Andre Lafortune) wrote:

> Could you clarify?
>
> For inclusion/prepending, there are two questions in my mind:
> 1) what happens if a module is included/prepended multiple times at different places in the hierarchy (e.g. in A and B, in the example above)
> 2) what happens if it is included/prepended multiple times at the same level (e.g. twice in A)
>
> Was the discussion about (b) allowing multiple times at the same level?

I don't remember there were discussions about inclusion in this topic. There were ones for prepending only. From what I understand the intention of prepending a module more than once is:

1)'s situation:

```
module X              def x; "X<%s>" % super end end
module Y prepend ::X; def x; "Y<%s>" % super end end
module Z prepend ::Y; def x; "Z<%s>" % super end end
class W
  prepend ::X
  prepend ::Z
  def x
    'W'
  end
end

W.new.x # => "X<Y<Z<X<W>>>>"
```

2)'s situation:

```
module X              def x; "X<%s>" % super end end
module Y prepend ::X; def x; "Y<%s>" % super end end
module Z prepend ::X; def x; "Z<%s>" % super end end
class W
  prepend ::Y
  prepend ::Z
  def x
    'W'
  end
end

W.new.x # => "X<Z<X<Y<W>>>>"
```

Note: current trunk does not behave this way.