Ruby master - Bug #11189

alias prepended module

05/27/2015 07:58 PM - ko1 (Koichi Sasada)

Status:	Open		
Priority:	Normal		
Assignee:	matz (Yukihiro Matsumoto)		
Target version:			
ruby -v:		**Backport:**	2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN

Description

```
module P
  def m1
    p :P_m1
    super
  end

  def m2
    p :P_m2
    super
  end
end

class C0
  def m1
    p :C0_m1
  end
end

class C1 < C0
  def m1
    p :C1_m1
    super
  end
  prepend P
  alias m2 m1
end

C1.new.m2
```

このプログラムは、

```
:P_m2
:P_m1
:C0_m1
```

という結果になります。superで辿っているはずなのに、同じモジュールのメソッドが2回呼ばれます。superで辿っていったら、必ず継承元へ行く、2度同じクラスは来ない、という常識があると思っていたので、Ruby的にはかなり驚きました。

この挙動は、設定時に:

- aliasでC1のメソッドテーブルに、C1#m2をP#m1へ飛ばすように設定する

次に、呼び出された時に:

- m2が呼ばれる
- P#m2を実行する
- P#m2のsuperを呼ぶ
- C1#m2を見ると、P#m1へ飛ぶようにaliasが設定されている
- P#m1を実行する、superする
- （なんでかC1#m1をスキップする□なんで？）
- C0#m1が呼ばれる

となっておりました。P#m2 -> P#m1 という流れがとても気持ち悪いですが、これは意図された挙動でしょうか。

C1#foo が呼ばれないのも気持ち悪いですが。

ちなみに、2.1 では C1#m1 が呼ばれていました。

```
* ruby 2.1.5p312 (2015-03-10 revision 49912) [i386-mswin32_110]
:P_m2
:P_m1
:C1_m1
:C0_m1
```

たしか、alias + prepend の議論が以前あったと思うのですが。
あ、[Bug #7842] だ。

ちなみに、私の予想は

```
:P_m1
:C1_m1
:C0_m1
```

かなぁ。

Related issues:

Related to Ruby master - Bug #7842: An alias of a "prepend"ed method skips th...	**Closed**	**02/13/2013**

History

#1 - 05/27/2015 08:03 PM - ko1 (Koichi Sasada)

今、こんなコードだと、

- C1 (m_tbl 空)
- T_ICLASS（m_tbl 空） -> P
- T_ICLASS（C1 の m_tbl） -> C1
- C0
- Object...

という継承関係（下が親）が作られるけど、alias のために C1 に m_tbl 作らないといかんね、って話な気がします。が、なんか以前この話をしていたことがあったような。結局 C1 にはテーブル作れないんでしたっけ。

メソッド定義については、

- alias の時は、C1::m_tbl へ
- メソッド定義の時は T_ICLASS(C1 の m_tbl）へ

と分けなきゃいけないのも、なんかかっこ悪いですね。

#2 - 05/28/2015 12:33 AM - usa (Usaku NAKAMURA)

- Related to Bug #7842: An alias of a "prepend"ed method skips the original method when calling super added

#3 - 08/12/2019 10:59 PM - jeremyevans0 (Jeremy Evans)

ko1 (Koichi Sasada) wrote:

> ちなみに、私の予想は
>
> ```
> :P_m1
> :C1_m1
> :C0_m1
> ```
>
> かなぁ。

This output would be against my expection. If C1 prepends P, then methods in P must be considered before methods in C1. Consider C1.ancestors:

```
[P, C1, C0, Object, Kernel, BasicObject]
```

P#m2 must be called before C1#m2, and alias m2 m1 in C1 only modifies the method table in C1, not in P.

All Ruby versions I tested have the same behavior as Ruby 2.1:

```
:P_m2
```

```
:P_m1
:C1_m1
:C0_m1
```

There does appear to be a related bug in super_method, though:

```
C1.new.method(:m2)
# => #<Method: C1(P)#m2>

C1.new.method(:m2).super_method
# => #<Method: C1(P)#m2(m1)>

C1.new.method(:m2).super_method.super_method
# => nil
# Should be #<Method: C1#m1>
```

Note that because of the way alias works with prepend, you can get an infinite loop in method lookup with aliases and only calling super:

```
module P
  def m1
    super
  end

  def m2
    super
  end
end

class C
  prepend P
  alias m2 m1
  alias m1 m2
end

C.new.m2
# SystemStackError
```