

Ruby master - Bug #11120

Unexpected behavior when mixing Module#prepend with method aliasing

05/04/2015 04:57 PM - pabloh (Pablo Herrero)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		
ruby -v:	ruby 2.2.2p95 (2015-04-13 revision 50295) [x86_64-linux]	Backport: 2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN

Description

I'm not completely sure myself if this should be considered a bug, but at least it should be up for discussion.

I stumbled upon this behavior when migrating some code using alias chains to Module#prepend.

Consider the following code:

```
# thingy.rb
class Thingy
  def thingy
    puts "thingy"
  end
end

# thingy_with_foo.rb
module ThingyWithFoo
  def thingy
    puts "thingy with foo"
    super
  end
end

Thingy.prepend(ThingyWithFoo)

# thingy_with_bar.rb
class Thingy
  alias_method :thingy_without_bar, :thingy
  # Wont't alias create an alias for Thingy#thingy but ThingyWithFoo#thingy instead

  def thingy_with_bar
    puts "thingy with bar"
    thingy_without_bar
  end
  # Expected to call original Thingy#thingy method but will call prepended method instead
end

  alias_method :thingy, :thingy_with_bar
end

# some_file.rb
Thingy.new.thingy # raises: stack level too deep (SystemStackError)
```

In a nutshell when calling super from ThingyWithFoo#foo it will call thingy_with_bar method, and this method will call back to ThingyWithFoo#foo by invoking thingy_without_bar, thus producing an endless loop.

This situation arises because alias_method is producing an alias not for the Thingy#thingy method the but for the upper method from ThingyWithFoo instead. May be this behavior could be considered correct, I'm still not sure, but it will probably became a problem for source code migrating from alias chains to use Modue#prepend, specially when other active gems could potentially still be using alias chains themselves without the user knowledge.

History

#1 - 05/08/2015 09:00 PM - pabloh (Pablo Herrero)

I gave some more thought to this but I can't really find a way to improve the migration path from aliases to prepend without creating new problems.

I think this issue should be closed.

#2 - 01/21/2016 01:18 PM - PSchambacher (Pierre Schambacher)

Adding my 2 cents here. I think that there's a big problem at the moment with Ruby, `Module.prepend` and `alias_method_chain`. Here is a sample of code:

```
module A
  def run
    puts 'A STARTS'
    super
    puts 'A ENDS'
  end
end

class B
  def run
    puts 'B STARTS'
    puts 'B ENDS'
  end
  prepend A
end

class B
  def run_with_chain
    puts 'CHAIN STARTS'
    run_without_chain
    puts 'CHAIN ENDS'
  end

  alias_method :run_without_chain, :run
  alias_method :run, :run_with_chain
end

B.new.run
```

Here is what I expect to happen:

`B.new.run`

- calls `A#run` because it's the first ancestor in the list
- calls `B#run` which is `B#run_with_chain`
- calls `B#run_without_chain`

Here is what actually happens

`B.new.run`

- calls `A#run` because it's the first ancestor in the list
- calls `B#run` which is `B#run_with_chain`
- calls `A#run` with `__callee__` being `run_without_chain`
- calls `B#run_with_chain`
- loop until stack too deep

I thought that `alias_method` was probably hooking into the `prepend` so that it would happen if you call one method or the other. This can be demonstrated with this code:

```
module A
  def run
    puts 'A STARTS'
    super
    puts 'A ENDS'
  end
end

class B
  def run
    puts 'B STARTS'
    puts 'B ENDS'
  end
  prepend A
  alias_method :run_without_chain, :run
end

B.new.run_without_chain
```

The output of this script is

```
A STARTS
B STARTS
B ENDS
A ENDS
```

This has probably been a problem since Ruby 2.0.0 and it going to become a bigger problem with Rails 5.0 dropping `alias_method_chain`. When people will start replacing `alias_method_chain` with `Module.prepend`, depending on the order of the gems, people might end up with stack too deep errors. Even more so if some people decide so simply replace `alias_method_chain` with 2 calls to `alias/alias_method` while others use `Module.prepend`.

I see 3 ways to make things safer:

1. Do not link the aliased method to the prepended module (meaning that calling `B.new.run_without_chain` would never call `A#run`).
2. Do the link, but remove it if the original method is redefined
3. Fix the super call in the module so it would call `B#run_without_chain` and not `B#run` (which is `B#run_with_chain`)

I personally don't think that solution 2 would be a good one. It's a bit specific and difficult to understand.

Solution 1 would be the most straightforward and I'd assume it would work for most people. There's a small chance that the code in the prepended module would not get executed in some situation because the aliased name is called rather than the original one. It's pretty easy to fix by aliasing the method as well in the module.

Solution 3 would also be a good solution. People in this situation at the moment have a stack too deep error. This would replace the stack too deep with a double call of `A#run` but this can be fixed with a guard (return super unless `__callee__ == :run`)

#3 - 01/21/2016 01:24 PM - PSchambacher (Pierre Schambacher)

Actually I really think that solution 1 is the good one. Here is another code sample:

```
module A
  def run
    puts 'RUN STARTS'
    super
    puts 'RUN ENDS'
  end

  def run_without_chain
    puts 'RUN_WITHOUT_CHAIN STARTS'
    super
    puts 'RUN_WITHOUT_CHAIN ENDS'
  end
end

class B
  def run
    puts 'B STARTS'
    puts 'B ENDS'
  end
  prepend A
  alias_method :run_without_chain, :run
end
```

```
B.new.run_without_chain
```

Here I would expect this output:

```
RUN_WITHOUT_CHAIN STARTS
B STARTS
B ENDS
RUN_WITHOUT_CHAIN ENDS
```

But this is the one I get:

```
RUN_WITHOUT_CHAIN STARTS
RUN STARTS
B STARTS
B ENDS
RUN ENDS
RUN_WITHOUT_CHAIN ENDS
```

To obtain the result I would like to get, I have to put the `alias_method` **before** the `prepend`. That doesn't feel really right for me since aliasing a method and prepending a module in the ancestors are pretty independent actions and they should not be order dependent.

#4 - 07/07/2019 05:11 AM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed