Ruby - Bug #16736

Prepending blank module breaks super call in aliased method

03/24/2020 02:09 PM - tycooon (Yuri Smirnov)

Status:	Closed		
Priority:	Normal		
Assignee:	jeremyevans0 (Jeremy Evans)		
Target version:			
ruby -v:	2.7.0p0 (2019-12-25 revision 647ee6f091) [x86_64-darwin19]	**Backport:**	2.5: DONTNEED, 2.6: DONTNEED, 2.7: REQUIRED

Description

Here is the test script:

```
class A
  def key
    ["some_key"]
  end
end

module M
  prepend Module.new

  def self.included(base)
    base.alias_method :base_key, :key
  end

  def key
    super + ["new_key"]
  end

  def generate
    base_key
  end
end

class B < A
  include M
end

x = B.new
p x.generate
```

In Ruby 2.7 I get the following error:

```
Traceback (most recent call last):
 2: from test.rb:28:in `<main>'
 1: from test.rb:19:in `generate'
test.rb:15:in `key': super: no superclass method `key' for #<B:0x00007fbc1704d028> (NoMethodError)
```

If I remove the prepend Module.new line or switch to Ruby 2.6, I get the expected result:

```
["some_key", "new_key"]
```

Associated revisions

Revision c745a60634260ba2080d35af6fdeaaae86fe5193 - 05/22/2020 02:36 PM - jeremyevans (Jeremy Evans)

Fix origin iclass pointer for modules

If a module has an origin, and that module is included in another module or class, previously the iclass created for the module had an origin pointer to the module's origin instead of the iclass's origin.

Setting the origin pointer correctly requires using a stack, since
the origin iclass is not created until after the iclass itself.
Use a hidden ruby array to implement that stack.

Correctly assigning the origin pointers in the iclass caused a
use-after-free in GC. If a module with an origin is included
in a class, the iclass shares a method table with the module
and the iclass origin shares a method table with module origin.

Mark iclass origin with a flag that notes that even though the
iclass is an origin, it shares a method table, so the method table
should not be garbage collected. The shared method table will be
garbage collected when the module origin is garbage collected.
I've tested that this does not introduce a memory leak.

This also includes a fix for Module#included_modules to skip
iclasses with origins.

Fixes [Bug #16736]

Revision c745a60634260ba2080d35af6fdeaaae86fe5193 - 05/22/2020 02:36 PM - jeremyevans (Jeremy Evans)

Fix origin iclass pointer for modules

If a module has an origin, and that module is included in another
module or class, previously the iclass created for the module had
an origin pointer to the module's origin instead of the iclass's
origin.

Setting the origin pointer correctly requires using a stack, since
the origin iclass is not created until after the iclass itself.
Use a hidden ruby array to implement that stack.

Correctly assigning the origin pointers in the iclass caused a
use-after-free in GC. If a module with an origin is included
in a class, the iclass shares a method table with the module
and the iclass origin shares a method table with module origin.

Mark iclass origin with a flag that notes that even though the
iclass is an origin, it shares a method table, so the method table
should not be garbage collected. The shared method table will be
garbage collected when the module origin is garbage collected.
I've tested that this does not introduce a memory leak.

This also includes a fix for Module#included_modules to skip
iclasses with origins.

Fixes [Bug #16736]

Revision c745a606 - 05/22/2020 02:36 PM - jeremyevans (Jeremy Evans)

Fix origin iclass pointer for modules

If a module has an origin, and that module is included in another
module or class, previously the iclass created for the module had
an origin pointer to the module's origin instead of the iclass's
origin.

Setting the origin pointer correctly requires using a stack, since
the origin iclass is not created until after the iclass itself.
Use a hidden ruby array to implement that stack.

Correctly assigning the origin pointers in the iclass caused a
use-after-free in GC. If a module with an origin is included
in a class, the iclass shares a method table with the module
and the iclass origin shares a method table with module origin.

Mark iclass origin with a flag that notes that even though the
iclass is an origin, it shares a method table, so the method table
should not be garbage collected. The shared method table will be
garbage collected when the module origin is garbage collected.
I've tested that this does not introduce a memory leak.

This also includes a fix for Module#included_modules to skip
iclasses with origins.

Fixes [Bug #16736]

Revision ad729a1d11c6c57efd2e92803b4e937db0f75252 - 05/23/2020 03:31 AM - jeremyevans (Jeremy Evans)

Fix origin iclass pointer for modules

If a module has an origin, and that module is included in another
module or class, previously the iclass created for the module had
an origin pointer to the module's origin instead of the iclass's
origin.

Setting the origin pointer correctly requires using a stack, since
the origin iclass is not created until after the iclass itself.
Use a hidden ruby array to implement that stack.

Correctly assigning the origin pointers in the iclass caused a
use-after-free in GC. If a module with an origin is included
in a class, the iclass shares a method table with the module
and the iclass origin shares a method table with module origin.

Mark iclass origin with a flag that notes that even though the
iclass is an origin, it shares a method table, so the method table
should not be garbage collected. The shared method table will be
garbage collected when the module origin is garbage collected.
I've tested that this does not introduce a memory leak.

This change caused a VM assertion failure, which was traced to callable
method entries using the incorrect defined_class. Update
rb_vm_check_redefinition_opt_method and find_defined_class_by_owner
to treat iclass origins different than class origins to avoid this
issue.

This also includes a fix for Module#included_modules to skip
iclasses with origins.

Fixes [Bug #16736]

Revision ad729a1d11c6c57efd2e92803b4e937db0f75252 - 05/23/2020 03:31 AM - jeremyevans (Jeremy Evans)

Fix origin iclass pointer for modules

If a module has an origin, and that module is included in another
module or class, previously the iclass created for the module had
an origin pointer to the module's origin instead of the iclass's
origin.

Setting the origin pointer correctly requires using a stack, since
the origin iclass is not created until after the iclass itself.
Use a hidden ruby array to implement that stack.

Correctly assigning the origin pointers in the iclass caused a
use-after-free in GC. If a module with an origin is included
in a class, the iclass shares a method table with the module
and the iclass origin shares a method table with module origin.

Mark iclass origin with a flag that notes that even though the
iclass is an origin, it shares a method table, so the method table
should not be garbage collected. The shared method table will be
garbage collected when the module origin is garbage collected.
I've tested that this does not introduce a memory leak.

This change caused a VM assertion failure, which was traced to callable
method entries using the incorrect defined_class. Update
rb_vm_check_redefinition_opt_method and find_defined_class_by_owner
to treat iclass origins different than class origins to avoid this
issue.

This also includes a fix for Module#included_modules to skip
iclasses with origins.

Fixes [Bug #16736]

Revision ad729a1d - 05/23/2020 03:31 AM - jeremyevans (Jeremy Evans)

Fix origin iclass pointer for modules

If a module has an origin, and that module is included in another module or class, previously the iclass created for the module had an origin pointer to the module's origin instead of the iclass's origin.

Setting the origin pointer correctly requires using a stack, since the origin iclass is not created until after the iclass itself. Use a hidden ruby array to implement that stack.

Correctly assigning the origin pointers in the iclass caused a use-after-free in GC. If a module with an origin is included in a class, the iclass shares a method table with the module and the iclass origin shares a method table with module origin.

Mark iclass origin with a flag that notes that even though the iclass is an origin, it shares a method table, so the method table should not be garbage collected. The shared method table will be garbage collected when the module origin is garbage collected. I've tested that this does not introduce a memory leak.

This change caused a VM assertion failure, which was traced to callable method entries using the incorrect defined_class. Update rb_vm_check_redefinition_opt_method and find_defined_class_by_owner to treat iclass origins different than class origins to avoid this issue.

This also includes a fix for Module#included_modules to skip iclasses with origins.

Fixes [Bug #16736]

History

#1 - 03/24/2020 03:25 PM - mame (Yusuke Endoh)

According to bisect, the behavior was changed by 5069c5f5214ce68df8b3954321ad9114c5368dc3.

@jeremyevans0 (Jeremy Evans) Could you please check it out?

#2 - 03/24/2020 03:25 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to jeremyevans0 (Jeremy Evans)

#3 - 03/24/2020 05:30 PM - jeremyevans0 (Jeremy Evans)

- Backport changed from 2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN to 2.5: DONTNEED, 2.6: DONTNEED, 2.7: REQUIRED

This is due to the origin pointers on the module iclasses being incorrectly set to the module's origin instead of the iclass origin. Setting the origin pointers correctly requires using a stack, as the origin iclasses are created after the iclasses themselves. I already did part of the work in the prepend patch in #9573. I've merged the necessary parts of the prepend patch locally and am running tests now. Assuming no errors I'll submit a pull request.

#4 - 03/24/2020 09:16 PM - jeremyevans0 (Jeremy Evans)

Pull request submitted: https://github.com/ruby/ruby/pull/2978

This includes a GC change to prevent a use-after-free, but I'm not sure if the change introduces a memory leak. Someone that knows more about GC and in what cases iclasses share method tables should probably review.

This pull request also fixes an issue in Module#included_modules to handle origin iclasses for modules correctly.

#5 - 05/22/2020 10:28 PM - jeremyevans (Jeremy Evans)

- Status changed from Assigned to Closed

Applied in changeset git|c745a60634260ba2080d35af6fdeaaae86fe5193.

Fix origin iclass pointer for modules

If a module has an origin, and that module is included in another module or class, previously the iclass created for the module had an origin pointer to the module's origin instead of the iclass's

origin.

Setting the origin pointer correctly requires using a stack, since
the origin iclass is not created until after the iclass itself.
Use a hidden ruby array to implement that stack.

Correctly assigning the origin pointers in the iclass caused a
use-after-free in GC. If a module with an origin is included
in a class, the iclass shares a method table with the module
and the iclass origin shares a method table with module origin.

Mark iclass origin with a flag that notes that even though the
iclass is an origin, it shares a method table, so the method table
should not be garbage collected. The shared method table will be
garbage collected when the module origin is garbage collected.
I've tested that this does not introduce a memory leak.

This also includes a fix for Module#included_modules to skip
iclasses with origins.

Fixes [Bug #16736]