Ruby trunk - Bug #9573

descendants of a module don't gain its future ancestors, but descendants of a class, do

02/27/2014 06:04 AM - rits (First Last)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.1.1p76 (2014-02-24 revision 45161) [i686-linux]	**Backport:**	1.9.3: UNKNOWN, 2.0.0: UNKNOWN, 2.1: UNKNOWN

Description

```
module Mod1
end

module Mod2
end

class Class1
end

class Class2 < Class1
end

p Class2.ancestors - Object.ancestors # [Class2, Class1]

Class1.include Mod1

p Class2.ancestors - Object.ancestors # [Class2, Class1, Mod1]

Mod1.include Mod2

p Mod1.ancestors - Object.ancestors # [Mod1, Mod2]

p Class2.ancestors - Object.ancestors # [Class2, Class1, Mod1]
```

---

note that descendants of a class do gain its future ancestors

so 2 issues:

1. It would seem natural that in dynamic language, dynamically added ancestors should propagate to descendants
2. Why is there a difference in ancestor propagation between modules and classes

History

#1 - 02/27/2014 06:23 AM - nobu (Nobuyoshi Nakada)

- Description updated

First Last wrote:

> so 2 issues:
>
> 1. It would seem natural that in dynamic language, dynamically added ancestors should propagate to descendants

It's a longstanding issue, a descendant knows its ancestors, but an ancestor doesn't know its descendants.

> 1. Why is there a difference in ancestor propagation between modules and classes

It is not between modules and classes, but caused by the order of inheritance and including.

#2 - 02/27/2014 06:43 AM - rits (First Last)

Nobuyoshi Nakada wrote:

> First Last wrote:
>
> > so 2 issues:
> >
> > 1. It would seem natural that in dynamic language, dynamically added ancestors should propagate to descendants
>
> It's a longstanding issue, a descendant knows its ancestors, but an ancestor doesn't know its descendants.

Is it the case that ancestors are cached in each descendant? So that it does not actually walk the ancestor tree each time. If so, is there any way to invalidate this cache for a given class or all, and have it reevaluate the ancestors?

> > 1. Why is there a difference in ancestor propagation between modules and classes
>
> It is not between modules and classes, but caused by the order of inheritance and including.

Please clarify. Mod1 is included in Class1 after Class2 extends Class1 and yet Class2 somehow learns of its new grandparent, Mod1. How does that happen if ancestors (Class1) do not know their descendants (Class2). So there is a difference, an ancestor added to a class, propagates to the descendant of this class, but an ancestor added to a module does not propagate to the descendant of this module.

#3 - 02/27/2014 08:30 AM - nobu (Nobuyoshi Nakada)

First Last wrote:

> > > 1. It would seem natural that in dynamic language, dynamically added ancestors should propagate to descendants
> >
> > It's a longstanding issue, a descendant knows its ancestors, but an ancestor doesn't know its descendants.
>
> Is it the case that ancestors are cached in each descendant? So that it does not actually walk the ancestor tree each time. If so, is there any way to invalidate this cache for a given class or all, and have it reevaluate the ancestors?

An included module is shared using an internal class (called as IClass), and IClasses are copied for each trees.
Now subclasses/submodules are maintained in each classes/modules for method cache validation, so it may be possible.

> > > 1. Why is there a difference in ancestor propagation between modules and classes
> >
> > It is not between modules and classes, but caused by the order of inheritance and including.
>
> Please clarify. Mod1 is included in Class1 after Class2 extends Class1 and yet Class2 somehow learns of its new grandparent, Mod1. How does that happen if ancestors (Class1) do not know their descendants (Class2). So there is a difference, an ancestor added to a class, propagates to the descendant of this class, but an ancestor added to a module does not propagate to the descendant of this module.

Class1 only knows Mod1, and its ancestor tree is copied into Class2.
And ditto for including a module.

#4 - 02/27/2014 10:12 AM - rits (First Last)

Nobuyoshi Nakada wrote:

> First Last wrote:
>
> > Please clarify. Mod1 is included in Class1 after Class2 extends Class1 and yet Class2 somehow learns of its new grandparent, Mod1. How does that happen if ancestors (Class1) do not know their descendants (Class2). So there is a difference, an ancestor added to a class, propagates to the descendant of this class, but an ancestor added to a module does not propagate to the descendant of this module.
>
> Class1 only knows Mod1, and its ancestor tree is copied into Class2.
> And ditto for including a module.

I am inferring that the ancestor tree is copied into Class2 when it extends Class1, yes?
But at this time Mod1 has not yet been included, so how does Class2 learn of Mod1?

and why is the situation different with modules?

Do you see what I am pointing out, an ancestor added to a class, propagates to the **past** descendant of this class (Class2 extends Class1 **before** Mod1 is included in Class1), but an ancestor added to a module does not propagate to the **past** descendant of this module.

#5 - 02/28/2014 12:11 AM - rits (First Last)

First Last wrote:

> Nobuyoshi Nakada wrote:
>
> > Class1 only knows Mod1, and its ancestor tree is copied into Class2.
> > And ditto for including a module.
>
> I am inferring that the ancestor tree is copied into Class2 when it extends Class1, yes?
> But at this time Mod1 has not yet been included, so how does Class2 learn of Mod1?
>
> and why is the situation different with modules?
>
> Do you see what I am pointing out, an ancestor added to a class, propagates to the **past** descendant of this class (Class2 extends Class1 **before** Mod1 is included in Class1), but an ancestor added to a module does not propagate to the **past** descendant of this module.

Can someone please explain this phenomenon.

#6 - 03/05/2014 04:07 PM - rits (First Last)

First Last wrote:

> First Last wrote:
>
> > Nobuyoshi Nakada wrote:
> >
> > > Class1 only knows Mod1, and its ancestor tree is copied into Class2.
> > > And ditto for including a module.
> >
> > I am inferring that the ancestor tree is copied into Class2 when it extends Class1, yes?
> > But at this time Mod1 has not yet been included, so how does Class2 learn of Mod1?
> >
> > and why is the situation different with modules?
> >
> > Do you see what I am pointing out, an ancestor added to a class, propagates to the **past** descendant of this class (Class2 extends Class1 **before** Mod1 is included in Class1), but an ancestor added to a module does not propagate to the **past** descendant of this module.
>
> Can someone please explain this phenomenon.

#7 - 03/13/2014 07:45 PM - rits (First Last)

First Last wrote:

> First Last wrote:
>
> > First Last wrote:
> >
> > > Nobuyoshi Nakada wrote:
> > >
> > > > Class1 only knows Mod1, and its ancestor tree is copied into Class2.
> > > > And ditto for including a module.
> > >
> > > I am inferring that the ancestor tree is copied into Class2 when it extends Class1, yes?
> > > But at this time Mod1 has not yet been included, so how does Class2 learn of Mod1?
> > >
> > > and why is the situation different with modules?
> > >
> > > Do you see what I am pointing out, an ancestor added to a class, propagates to the **past** descendant of this class (Class2 extends Class1 **before** Mod1 is included in Class1), but an ancestor added to a module does not propagate to the **past** descendant of this module.

> Can someone please explain this phenomenon.

What is the objection to explaining how this works?

#8 - 03/13/2014 08:34 PM - jeremyevans0 (Jeremy Evans)

First Last wrote:

> What is the objection to explaining how this works?

nobu explained how it works. However, as he is not a native English speaker, let me attempt to clarify.

In ruby, there exist pseudo-copies of modules called iclasses. These copies share the same variable(class variable/instance variable/constant) tables and the same method tables, but have a different super pointer in their C struct. iclasses are made when you attempt to include a module in another module or class.

When you do:

```
module M0; end
module M1
  include M0
end
```

This creates a module M1 that includes an iclass of M0 (notated below as i0M0) in its inheritance list. For a module, the inheritance list is the the super pointer in struct RClass).

When you do:

```
class A
  include M1
end
```

What happens is iclasses of M1 and i0M0 are made (notated below as i0M1, i1M0). So method lookup for an instance of A will be:

```
A -> i0M1 -> i1M0 -> Object 
```

Here's how the super pointers for the struct RClass should look:

```
M0: NULL
i0M0: NULL
M1: i0M0

i1M0: Object
i0M1: i1M0
A: i0M1
```

When you do:

```
module M2; end
M1.include M2
```

This creates an iclass of M2 (i0M2) and updates the super pointer in M1, as shown:

```
M1: i0M2
i0M2: i0M0
i0M0: NULL
```

However, it has no effect on any of the iclasses of M1 already created.

Ruby doesn't have multiple inheritance. Ruby method lookup uses a linked listed, not a tree. This is the reason for iclasses, and why including module B in module A after A has been included in class C does not include B in C.

Note that I am not an expert on ruby internals, so if there are errors in the above description, hopefully a more knowledgeable person can correct me.

#9 - 03/13/2014 09:29 PM - rits (First Last)

Jeremy Evans wrote:

> Ruby doesn't have multiple inheritance. Ruby method lookup uses a linked listed, not a tree. This is the reason for iclasses, and why including module B in module A after A has been included in class C does not include B in C.

Conceptually Ruby does have multiple inheritance, an object is_a?(all included modules).

Is MRI's iclass snapshotting an implementation detail? Can it theoretically be done differently (e.g. tree that you mentioned)

#10 - 06/05/2014 06:16 AM - hsbt (Hiroshi SHIBATA)

ref. https://github.com/ruby/ruby/pull/549