Ruby - Feature #5148

Constant Namespace/Scoping in Dynamic Classes/Modules

08/03/2011 12:01 AM - trans (Thomas Sawyer)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

When defining a dynamic class or module, the constants defined within it are not kept within it's namespace, but are tied to the outer scope's namespace.

```
c = Class.new do
  module M; end
end
#=> #<Class:0xa601300>
c::M  #=> M
M     #=> M
```

This presents serious limitations in designing domain specific languages that take advantage of Ruby's class system. For instance in creating a test framework that uses blocks to defines testcases it is not possible to isolate fixture constants from those of other testcases. E.g.

```
describe "fancy mixin" do
  class Foo
    include FancyMixin
  end
  ...
end

describe "unfancy mixin" do
  class Foo
    include UnfancyMixin
  end
  ...
end
```

Foo in the unfancy testcase will use Foo in the fancy testcase, causing unexpected issues --especially as these testcases may be defined in different files and coded by different people.

History

#1 - 08/03/2011 12:12 AM - nobu (Nobuyoshi Nakada)

- Tracker changed from Bug to Feature

#2 - 08/07/2011 03:31 AM - Eregon (Benoit Daloze)

Hi,
On 2 August 2011 17:01, Thomas Sawyer transfire@gmail.com wrote:

> Bug #5148: Constant Namespace/Scoping in Dynamic Classes/Modules
> http://redmine.ruby-lang.org/issues/5148
>
> When defining a dynamic class or module, the constants defined within it are not kept within it's namespace, but are tied to the outer scope's namespace.
>
> c = Class.new do
> module M; end
> end
> #=> #Class:0xa601300
> c::M #=> M
> M   #=> M

I also noticed that recently.

It seems inconsistent to me.
Let's have some code:
https://gist.github.com/1129567

Constant scoping seems to depend on the the fact it is defined in a "dynamic" scope or not.
But one can force to define on a specific scope by doing scope::Const = value (like self::C = value in a Class.new block).

I think the constant definition scope should always be the current scope (as you would always add "self::" before the constant name).
Also, this should be true for constant resolution, I should not need self.class::CONST, in a parent class to resolve a child class's constant.

To "solve" your problem, you could use self::

describe "unfancy mixin" do
class self::Foo # not at toplevel
include UnfancyMixin
end

Foo # uninitialized constant Foo (NameError)

self::Foo # => #Class:0x00000100857ba8::Foo
const_get :Foo # => #Class:0x00000100857ba8::Foo
end

But it has an immediate pitfall: you can not reference it by 'Foo',
you need 'self::Foo' or 'const_get :Foo'.

Variables and method calls are fully dynamic for definition/resolution, in opposition to constants.

Can someone explain the rationale behind this logic ?

#3 - 03/31/2012 10:18 AM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to matz (Yukihiro Matsumoto)

Hello,

Matz, do you think it should be fixed?
Nobu, do you think it can be fixed?

--
Yusuke Endoh mame@tsg.ne.jp

#4 - 03/31/2012 06:19 PM - nobu (Nobuyoshi Nakada)

I don't think it will be accepted.

#5 - 04/01/2012 12:33 AM - matz (Yukihiro Matsumoto)

- Assignee changed from matz (Yukihiro Matsumoto) to mame (Yusuke Endoh)

I think this change is a good idea, basically. But I am not sure how much influence it would have to existing Ruby programs, and implementations. For example, Nobu has already shown his concern. We have to experiment before accepting this proposal.

Matz.

#6 - 04/23/2012 11:55 PM - mame (Yusuke Endoh)

- Status changed from Assigned to Rejected

- Assignee deleted (mame (Yusuke Endoh))

Okay.

Thomas, or anyone interested, please create a patch yourself, experiment it, show an objective evaluation of the influence of this feature, and then, persuade matz again.

--
Yusuke Endoh mame@tsg.ne.jp