Ruby master - Feature #12317

Name space of a module

04/26/2016 12:15 AM - sawa (Tsuyoshi Sawada)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

I want a method to return the name space of a module, something like:

```
class A; module B; module C end end end

A::B::C.namespace => [A, A::B, A::B::C]
```

There is nesting method that is similar, but that only returns the lexical nesting information.

There are also some known hacks for this, converting the module to the string representation using to_s or name, and then splitting it by ::. But that easily breaks if the module is anonymous, or is a singleton module. I would like a more robust, core method.

History

#1 - 04/26/2016 06:16 PM - shevegen (Robert A. Heiler)

Interesting. I am indifferent to it, but possibly lean towards giving a +1.

In some of my classes, I use a module called "opn", which stands short for
"output program name". I needed this for some of my ruby code, when it
gives output on the terminal, to know where exactly the output happens (since
I tend to use a lot of different scripts, in different projects).

Perhaps a year or so ago, I added a new method called "opnn". That name is
not really logical, but I simply repeat the last character, which helps
me identify that I still want to call "opn", but I will do so including
the namespace.

This may sound confusing, so here is what happens in ruby code:

```
class Foo

class Bar

  def initialize
    opnn; e 'Hello World!'
  end

end

end

Foo::Bar.new
```

And the above will output Hello World! (e is my alias for puts, I am
lazy, ruby allows me to be lazy) in grey ansi colours, prefixed with
the name of the class in question.

So the output will actually be:

Foo::Bar: Hello World!

So far, so good. All works fine.

opnn is a weird method though; I actually define it on the class itself.

The definition tends to be like so:

```
def opnn
  Opn.opn(namespace: NAMESPACE)
```

```
end
```

Opn is obviously a module, namespace, in the gem called opn. The argument is a hash. The constant called NAMESPACE is actually the namespace, and now here comes the relevant part.

I define that constant within the class itself like this:

```
NAMESPACE = self.inspect.to_s
```

Which I found has worked best so far.

All of this is not really ... awesome. In particular, that I have to manually define the namespace for each class, is not so great.

Perhaps a **NAMESPACE** identifier on that of that, which would be equivalent to self.inspect.to_s all the time? (Or perhaps some shorter way ... I just need a simple way to obtain the string representation of the namespace in question)

This suggestion is a bit different from Tsuyoshi Sawada, I don't want to hijack his suggestion. I just found it semi-fitting if I also detail a bit how I write ruby code and deal with namespaces (and this may all change as time passes by and better ways are found)

Thanks!

#2 - 05/07/2016 12:25 PM - sawa (Tsuyoshi Sawada)

I realized that the order of modules that I wrote in the expected output is not in accordance with methods like nesting or ancestors (which go from self to the modules that are farther). So perhaps, the following may be better:

```
A::B::C.namespace # => [A::B::C, A::B, A]
```

#3 - 05/15/2016 12:58 PM - dsferreira (Daniel Ferreira)

Hi Tsuyoshi.

A clear namespace definition is important for the implementation of my proposed feature: internal interface
In fact I was thinking in proposing such a similar method.

Following the challenge put by Jörg W Mittag internal - note#5

What in your opinion should be the output of #namespace for the challenges pointed out by Jörg?

Simple challenge example:

```
module Foo
  class Bar
  end
end

Baz = Foo

::Baz::Bar.ancestors
# => [Foo::Bar, Object, Kernel, BasicObject]

::Baz::Bar.namespace
# => ?
```

#4 - 05/15/2016 01:39 PM - sawa (Tsuyoshi Sawada)

Daniel Ferreira wrote:

> Simple challenge example:
>
> ```
> module Foo
>   class Bar
>   end
> end
> Baz = Foo
> ```

It should return

```
::Baz::Bar.namespace
# => [Foo::Bar, Foo]
```

I haven't read through the issue, but what is the issue here?

#5 - 05/15/2016 01:47 PM - dsferreira (Daniel Ferreira)

No issue in my point of view.
I agree with your reply.

Can you take a look at my proposal to see what you think about it?

#6 - 06/13/2016 09:54 AM - akr (Akira Tanaka)

I think lexical information is not obtainable in a method.

```
% ruby -e '
module A
  module B
  end
end

module C
  Z = 2
  module A::B
    p Z
  end
end

p A::B.namespaces
'
2
???
```

Z is refarenceable in B in C but not B in C.
Since B in C and B in A is same object, A::B.namespaces method cannot distinguish them.

#7 - 06/13/2016 10:17 AM - sawa (Tsuyoshi Sawada)

Akira Tanaka wrote:

> I think lexical information is not obtainable in a method.
>
> ```
> % ruby -e '
> module A
>   module B
>   end
> end
>
> module C
>   Z = 2
>   module A::B
>     p Z
>   end
> end
>
> p A::B.namespaces
> '
> 2
> ???
> ```
>
> Z is refarenceable in B in C but not B in C.
> Since B in C and B in A is same object, A::B.namespaces method cannot distinguish them.

I am not sure if I get the issue correctly. I expect:

```
module A; module B; end end
A::B.namespace # => [A::B, A]
module C
  A::B # => A::B
  A::B.namespace # => [A::B, A]
end
```

#8 - 06/15/2016 04:55 AM - shyouhei (Shyouhei Urabe)

"What is a namespace?" can be a fundamental issue here. I guess Akira's point at comment #6 is that a "namespace" is to do something with names. From such point of view, the "A::B" output is at least insufficient to be a "namespace". That cannot explain why Z is visible inside.

#9 - 09/04/2016 04:59 PM - jwmittag (Jörg W Mittag)

I guess the core question is "what is a namespace?"

For example, take this Ruby code:

```
foo = Class.new
bar = Class.new
baz = Module.new

baz.const_set(:Foo, foo)
baz.const_set(:Bar, bar)

quux = Module.new
quux.const_set(:Goobledigook, foo)
quux.const_set(:Blahdiblah, bar)

baz.send(:remove_const, :Bar)

module One; module Two; end end

One::Two::Three = quux::Goobledigook

One::Two::Three.namespace
#=> what should it return?

quux::Goobledigook.namespace
#=> what should it return?
```

Or really just the simple question:

```
module Foo; module Bar; end end
Baz = Foo::Bar

Foo::Bar.namespace
#=> what should it return?

Baz.namespace
#=> what should it return?
```