Ruby - Feature #16615

Group style access scope for macros

02/07/2020 12:18 PM - ted (Ted Johansson)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Given a method .bar, which defines an instance method #baz on a class, and returns the defined method's name as a symbol (:baz).

```
class Foo

  private

  # On evaluation defines a method and returns its name.
  # In current Ruby, that method will be public. The suggested
  # behaviour is to make it private, since the class method
  # which defines the instance method is in the private scope.
  #
  bar :baz
end
```

it would be neat if the dynamically defined instance method respected the scope in which its definition originated. (In this particular case private.)

Essentially the request is to extend the special powers of attr_* (being able to define methods that honour visibility scopes) to any method.

Note: I am aware that inline access scopes already work for dynamically defined methods, as they merely accept a symbol as an argument.

Edit: Changed macro to bar so people don't get hung up on the name of the method (which has no importance to the proposal.)

History

#1 - 02/07/2020 12:48 PM - shevegen (Robert A. Heiler)

The suggestion is fairly short; might help to expand a bit, in particular why it would be necessary/useful.

Two comments from me in regards to the proposal:

1. Is there a defining difference towards e. g. the attr* family? Perhaps I missed this in the proposal, but it should be remembered, even more so as this may become a question to newcomers for ruby - see the old questio about "Symbol versus String" and strange add-ons such as HashWithIndifferentAccess.

2. I believe the name "macro" is an awkward name. I am not sure that name should be added, but even more importantly the relatedness to 1) should be considered. (The public versus private distinction in ruby is not a strong one, due to ruby's dynamic nature and philosophy. I understand why the distinction is there, but personally I very, very rarely use public/private ever; if then mostly just to portray intention to others in a libary, but even then I often wonder whether this is even necessary. I think it then comes down a lot to the personal preferences of a given ruby user more than anything else.).

#2 - 02/11/2020 06:15 PM - shan (Shannon Skipper)

The idea of attrs returning a Symbol was rejected in: https://bugs.ruby-lang.org/issues/9453

Your example doesn't seem to be a macro. How does this differ from attrs?

#3 - 03/06/2020 06:43 AM - ted (Ted Johansson)

- Description updated

#4 - 03/06/2020 06:45 AM - ted (Ted Johansson)

- Description updated

#5 - 03/06/2020 06:50 AM - ted (Ted Johansson)

shevegen (Robert A. Heiler) wrote in #note-1:

> Two comments from me in regards to the proposal:
>
> 1. Is there a defining difference towards e. g. the attr* family? Perhaps I missed this in the proposal, but it should be remembered, even more so as this may become a question to newcomers for ruby - see the old questio about "Symbol versus String" and strange add-ons such as HashWithIndifferentAccess.

The attr_* family of methods currently enjoys special treatment here, as they are the only methods which define instance methods that will obey this style of visibility scope.

> 1. I believe the name "macro" is an awkward name. I am not sure that name should be added, but even more importantly the relatedness to 1) should be considered.

I changed the name to a meta-syntactic variable, since it's irrelevant to the proposal itself.

> (The public versus private distinction in ruby is not a strong one, due to ruby's dynamic nature and philosophy. [...] I think it then comes down a lot to the personal preferences of a given ruby user more than anything else.).

I think you might be projecting your personal opinion onto "Ruby in general". I can't think of any library which doesn't make extensive use of encapsulation. (For example, Rails has 1,073 uses of private.) Not saying you have todo it in your projects, but it is a thing, and in my experience working with large code bases, it's also a Good Thing™. :-)

#6 - 03/09/2020 02:39 PM - Dan0042 (Daniel DeLorme)

- Description updated

Maybe a more concrete example would be helpful:

```
def defc
  define_method(:c){ }
end
class Foo
  private
  def a; end
  attr_accessor :b
  defc
end
Foo.new.a #=> NoMethodError (private method `a' called)
Foo.new.b #=> NoMethodError (private method `b' called)
Foo.new.c #=> nil
```

Having c be declared as private may be desirable... but there's sure to be a compatibility impact.

Is that why you added the condition that this should only happen if the meta-method (bar / defc) returns the instance method name (baz / c) as a symbol?