CommonRuby - Feature #11541

Let attr_accessor, _reader & _writer return symbols of the defined methods

09/20/2015 03:37 PM - iGEL (Johannes Barre)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

Since Ruby 2.1, def returns a symbol with the name of the just defined method, so you can easily pass it to visibility modifiers like private, protected, and public. Why not let attr_reader & friends return an array with the names of the defined methods, so we can easily write:

```
private attr_reader :method1, :method2
```

To fully support the example above, private would be required to accept also arrays with method names. Without it, it would require the star syntax, which would already be an improvement:

```
private *attr_reader :method1, :method2
```

I wrote two test cases to better illustrate the impact:

```
test/ruby/test_module.rb:

  def test_attr_return_value
    c = Class.new

    assert_equal(%i(reader1 reader2), c.class_eval { attr_reader(:reader1, :reader2) })
    assert_equal(%i(writer1= writer2=), c.class_eval { attr_writer(:writer1, :writer2) })
    assert_equal(%i(accessor1 accessor1= accessor2 accessor2=), c.class_eval { attr_accessor(:acce
ssor1, :accessor2) })
  end

test/ruby/test_method.rb:

  def test_visibility_modifier_with_array
    c = Class.new do
      def m1; end
      def m2; end
    end

    c.class_eval { private %i(m1 m2) }
    assert(c.private_method_defined?(:m1))
    assert(c.private_method_defined?(:m2))

    c.class_eval { protected %w(m1 m2) }
    assert(c.protected_method_defined?(:m1))
    assert(c.protected_method_defined?(:m2))

    c.class_eval { public :m1, [:m2] } # Not sure if this should be allowed.
    assert(c.public_method_defined?(:m1))
    assert(c.public_method_defined?(:m2))

    assert_raise(NameError) do
      c.class_eval { private %i(m1 m2 m3) }
    end
    assert(c.private_method_defined?(:m1))
    assert(c.private_method_defined?(:m2))

    assert_raise(TypeError) do
      c.class_eval { protected [:m1, 2] }
    end
    assert(c.private_method_defined?(:m1))
```

```
    assert_raise(TypeError) do
      c.class_eval { public [:m1, [:m2]] } # Not sure about this case. Should it be allowed?
    end
    assert(c.public_method_defined?(:m1))
  end
```

WDYT? Thank you!

Related issues:

Related to CommonRuby - Feature #12019: Better low-level support for writing ...	**Open**
Related to Ruby master - Feature #14397: public, protected and private should...	**Assigned**
Related to Ruby master - Feature #6470: Make attr_accessor return the list of...	**Open**
Related to Ruby master - Feature #9453: Return symbols of defined methods for...	**Rejected**
Related to Ruby master - Feature #13560: Module#attr_ methods return reasonab...	**Open**

History

#1 - 01/25/2016 07:55 PM - pitr.ch (Petr Chalupa)

+1, this is also very useful for low level concurrency proposals. It allows to write public attr :status, atomic: true where attr returns array of 4 defined atomic helper methods. The array is accepted by public method which modifies the visibility of the helpers.

#2 - 01/25/2016 10:20 PM - Eregon (Benoit Daloze)

- Related to Feature #12019: Better low-level support for writing concurrent libraries added

#3 - 01/25/2018 12:20 PM - Eregon (Benoit Daloze)

- Related to Feature #14397: public, protected and private should return their arguments instead of self added

#4 - 03/16/2018 06:12 AM - gfx (Goro FUJI)

+1

I'm about to issue the same feature request

#5 - 11/28/2018 10:51 AM - shevegen (Robert A. Heiler)

I don't have a big pro or con opinion (am neutral here), but perhaps we could move this
to an upcoming developer meeting anyway?

#6 - 01/10/2019 09:16 AM - Eregon (Benoit Daloze)

- Related to Feature #6470: Make attr_accessor return the list of generated method added

#7 - 08/19/2019 01:54 AM - znz (Kazuhiro NISHIYAMA)

- Related to Feature #9453: Return symbols of defined methods for `attr` and friends added

#8 - 08/19/2019 01:55 AM - znz (Kazuhiro NISHIYAMA)

- Related to Feature #13560: Module#attr_ methods return reasonable values added