Allow calling a private method with `self.`

There is an inconsistency between calling a private attribute writer being allowed with self.value = syntax and self.value not being allowed on a private attribute writer.

Calling a private method in this way can be useful when trying to assign the return value of this private method to a local variable with the same name.

The attached patch handles this by compiling the calling into a function call by using the VM_CALL_FCALL flag, so it is as if the call were made without the self, prefix, except it won't be confused with local variables at the VM instruction level. It is also compiled like an assignment call, except I didn't use the COMPILE_RECV macro, since that would remove the CHECK macro usage around the COMPILE line.
I may not completely understand the issue description. What is the inconsistency? (That is a honest question, by the way; I am not fully understanding the issue domain.)

I am not even entirely sure what a private attribute writer is either; can we use these terms when we can use e.g. send() at all times? I may not understand this, but I assume you can get the value of any method via .send() and assign it to the local variable?

Here is a script to help demonstrate the inconsistency, where self.bar = 123 is allowed by self.bar is not.

```ruby
class Foo
  def foo
    self.bar = 123 # allowed
    self.bar # raises
  end

  private
  attr_accessor :bar
end

Foo.new.foo
```

By attribute writer, I was just referring to an assignment method like the one defined by attr_writer, although the same applies to any assignment method like def bar=(value); value; end. The inconsistency is just more obvious when dealing with the pair of methods defined by attr_accessor if they are private because self. works with one of them but not the other as shown above.

shevegen (Robert A. Heiler) wrote:

I may not understand this, but I assume you can get the value of any method via .send() and assign it to the local variable?

Yes, it can be easily worked around, it just doesn't seem like it should be necessary to workaround this limitation. The point of private is to keep things from being accessible from other objects, which we know isn't the case when a call is made on self. directly.

Why do you use private attr_accessor? You can access its instance variable directly.

Allowing self.private_method has a unpreferable side effect; it allows self.global_function, i.e., self.require "foo", self.lambda {}, self.fork, etc.

Accepted. But as I said in #11297, the document should be updated. After the patch, you can say self.puts("hello"), which can confuse you.

Allow calling a private method with self.

This makes it consistent with calling private attribute assignment methods, which currently is allowed (e.g. self.value =).
Calling a private method in this way can be useful when trying to assign the return value to a local variable with the same name.

[Feature #11297] [Feature #16123]

#9 - 06/10/2020 12:34 PM - nobu (Nobuyoshi Nakada)
- Related to Bug #16947: private method unexpected behavior added

Files

<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001-Allow-calling-a-private-method-with-self.diff.txt</td>
<td>3.28 KB</td>
<td>08/23/2019</td>
<td>dylants (Dylan Thacker-Smith)</td>
</tr>
</tbody>
</table>