Ruby - Feature #6682

Add a method to return an instance attached by a singleton class

07/01/2012 07:22 PM - ryoqun (Ryo Onodera)

Status:	Assigned
Priority:	Normal
Assignee:	shyouhei (Shyouhei Urabe)
Target version:	

Description

=begin
Currently, there is no easy way to get the attached instance from a singleton class. For MRI, we have to resort to writing an C extension. So it'll be useful to add an instance method to Class to return the attached instance if the given class object is a singleton class.

I'll show what I want in the code-wise with the following code snippet:

text = "I love Ruby."
klass = text.singleton_class

=> #<Class:#String:0x000000027383e8>

klass.singleton_instance # <= This is the new method.

=> "I love Ruby."

String.singleton_instance # <= This should return nil because String isn't a singleton class and there is no singleton instance, rather there will be many instances.

=> nil

As for use cases, in my case, I wanted to create a module to add class methods. And it has some state, so must be initialized properly. And it can equally be used by Class#extend and Class#include like this:

module Countable
attr_reader(:count)

```
class << self
  def extended(extended_class)
    p("extending #{extended_class}")
    super

    initialize_state(extended_class)
  end

  def included(included_class)
    p("including #{included_class}")
    super

    if included_class.singleton_instance # <= Currently, I can't do this.
      initialize_state(included_class.singleton_instance)
    end
  end

  private
  def initialize_state(object)
    p("initializing state of #{object}")
    object.instance_variable_set(:@count, 0)
  end
end
```

end

class Person
extend(Countable)
end

class Book
class << self
include(Countable)
end
end

p(Person.count)
p(Book.count)

=> "extending Person"

=> "initializing state of Person"

=> "including #Class:Book"

=> "initializing state of Book"

=> 0

=> 0

Others wanted this functionality as shown by ((<this stackoverflow page|URL:http://stackoverflow.com/questions/7053455/given-a-ruby-metaclass-how-do-i-get-the-instance-to-which-it-is-attached>)). Also, I found several actual C-extensions for this kind of functionality on the wild browsing ((<a search result|URL:https://github.com/search?q=rb_iv_get+__attached__&repo=&langOverride=&start_value=1&type=Code&language=C>)) on github.

- ((<eigen|URL:https://github.com/elliottcable/refinery/blob/853dcc2254557200d1d6be4cb9c105e8fa9d01a9/ext/eigen/eigen.c#L12>))
- ((<mult|URL:https://github.com/banister/mult/blob/6a1d0bdd383e7e231c5b7c2c718204dfb6ba28ca/ext/mult/mult.c#L43>))

Thanks for creating a great language. Especially I love its meta-programming capability. I'd wish this feature to lead to better meta-programming capability of Ruby.
=end

History

#1 - 07/09/2012 07:57 AM - ryoqun (Ryo Onodera)

I opened a pull request for this feature: https://github.com/ruby/ruby/pull/142

#2 - 09/25/2012 09:06 PM - ryoqun (Ryo Onodera)

=begin
There is a problem in the original proposal. It is that we can't determine whether a given class is singleton or not by checking an object returned from (({Class#singleton_instance})) in some cases. Consider this exapmle:

String IS NOT singleton

String.singleton_instance => nil

NilClass IS singleton

NilClass.singleton_instance => nil

(({NilClass})) is a singleton class and returning (({nil})) from (({Class#singleton_instance})) is completely legitimate.

Thus, I refined the behavior of Class#singleton_instance a bit.

String.singleton_instance => raises TypeError
NilClass.singleton_instance => nil

#3 - 11/19/2012 08:43 AM - zzak (zzak _)

- File class_singleton_instance.patch added

- Assignee set to shyouhei (Shyouhei Urabe)

I've added Ryo's patch from github: https://github.com/ruby/ruby/pull/142

Shyouhei, could you review this?

Thanks

#4 - 11/19/2012 06:04 PM - ryoqun (Ryo Onodera)

=begin
zzak, thanks for updating this feature request.

I add more explanation.

First of all, I'll clarify my intention: I want any kind of modules to be interchangeably used in either of the following 2 ways:

(1) Extend inside a class (This is used preferably when there is no class method for (({Person})))

```
class Person
  extend(Countable)

  def foo
    ..
  end

  ...
end
```

(2) Include inside a singleton class (This is used preferably when there are class methods for (({Person}))). This is used to group all of code related to class methods for readability)

```
class Person
  def foo
    ..
  end

  ...

  class << self
    include(Countable)

    def foo
      ..
    end

    ...
  end
end
```

As a library author, I want my library users to be able to choose how to use my library modules as above.

I'm assuming that extending inside a class is functionally equivalent with including inside a singleton class. And I think it should be. If this assumption isn't valid and I'm wrong, I'll close this feature request.

For ordinary modules, there is no issue. However, for state-full modules, currently, we can only use such a module by extending inside a class not by including inside a singleton class.

To accomplish this, there are two approaches.

(1) Add (({Class#singleton_instance})) (this feature request)

(2) Add a hook like (({singleton_included})) akin to (({singleton_method_added})) (proposed by n0kada)

I don't case which approach is adapted.

I personally discussed about this with n0kada in the past. This is summary of that discussion.

=end

#5 - 11/24/2012 11:12 AM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Target version set to 2.6

#6 - 12/25/2017 06:15 PM - naruse (Yui NARUSE)

- Target version deleted (2.6)

Files

class_singleton_instance.patch	4.29 KB	11/19/2012	zzak (zzak _)