Ruby trunk - Bug #13257

Symbol#singleton_class should be undef

02/27/2017 09:46 AM - tagomoris (Satoshi TAGOMORI)

Status:	Feedback		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.4.0p0 (2016-12-24 revision 57164) [x86_64-darwin15]	**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

Objects of some classes doesn't have singleton classes (e.g., Symbol, Integer, Float...).
Symbol#singleton_class raises TypeError.

But Symbol#respond_to?(:singleton_class) returns true, and we cannot know when #singleton_class returns a valid class or raise errors in any way (except for calling it).

I think that such #singleton_class methods should be undef.

History

#1 - 03/02/2017 09:27 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

Fundamentally, #respond_to? is a method to check method existence. The class Symbol does have #singleton_class so that it's natural for it to return true, even though #singleton_class always fails.

It seems you want to check the availability of functionality. That means you want to change the meaning of the method.
I do not reject the proposal immediately, but at least it's controversial. Persuade us if you really want to change it.

Matz.

#2 - 03/02/2017 10:13 AM - tagomoris (Satoshi TAGOMORI)

I understood that #respond_to? doesn't matter in fact.
What I really want is to undef Symbol#singleton_class (and Integer, Float too).

#3 - 03/02/2017 10:15 AM - tagomoris (Satoshi TAGOMORI)

- Subject changed from Symbol#respond_to?(:singleton_class) should be false (but true) to Symbol#singleton_class should be undef

#4 - 03/02/2017 10:20 AM - tagomoris (Satoshi TAGOMORI)

- Description updated

#5 - 03/02/2017 11:16 AM - knu (Akinori MUSHA)

What's the motivation behind this? You cannot tell if an object has no singleton class just because it doesn't define or respond to #singleton_class anyway. BasicObject is an exception, and I could imagine certain kinds of objects (for delegate/proxy/mocking) might have their reasons they shouldn't undefine #singleton_method. If you have an object, then (class << object; true; end rescue false) would be the best way to test if it has a singleton class.

#6 - 03/02/2017 11:34 AM - tagomoris (Satoshi TAGOMORI)

My motivation is to know whether there is an class/module in ancestors or not.

I needed it at this patch: https://github.com/msgpack/msgpack-ruby/pull/132/files
The feature implemented in this patch serialize/deserialize objects in a configured format if the object is an kind of registered classes or modules.
It should consider about the order of inheritance, so it can't use #is_a? method, and it should consider about modules added by #extend, so it should check ancestors of singleton classes.

#7 - 03/02/2017 11:47 AM - tagomoris (Satoshi TAGOMORI)

After chat with narse, I found that what I really want is the list of classes and modules on method search path of objects (nearly equal to obj.singleton_class.ancestors).
If I can get it without creating/allocating singleton classes, it's nice to me.

I missed to think about BasicObject instances, but it's out of our scope in this use case.

#8 - 03/02/2017 03:08 PM - knu (Akinori MUSHA)

Is that a use case where you need a method for checking without allocating a singleton class?
As I read it diagonally, it looks like you are always accessing and using the singleton class when an object has one.

#9 - 04/17/2017 06:30 AM - akr (Akira Tanaka)

I guess a some method such as Kernel#ancestor_modules, which returns all class/modules to search methods, supports your needs directly than undef'ing singleton_class.