Ruby master - Feature #11505

Module#=== should call #kind_of? on the object rather than rb_obj_is_kind_of which only searches the ancestor heirarchy

09/03/2015 06:43 PM - lamont (Lamont Granquist)

Status:	Rejected
Priority:	Normal
Assignee:	
Target version:	

Description

In trying to implement a Decorator pattern which wraps Hashes and/or Arrays, I would like case equality to work such that the decorator would be handled by case statements on Array or Hash:

```
case
when 'Array'
  # handle Arrays
when 'Hash'
  # handle Hashes
end
```

My decorators delegate #is_a? and #kind_of? to the decorated object so that #is_a?(Hash) or #kind_of?(Hash) return true even though the decorator does not inherit from Array or Hash.

When trying to use the decorator in case statements, however, the case-equality is handled by Array#=== or Hash#===. This is a method inherited from Module#=== which calls rb_obj_is_kind_of:

https://github.com/ruby/ruby/blob/f830ace8a831a954db7a6aae280a530651a5b58a/object.c#L1519-L1533

The implementation of rb_obj_is_kind_of does not call arg#kind_of?(mod) as its name might suggest, but instead searches the ancestor hierarchy:

https://github.com/ruby/ruby/blob/f830ace8a831a954db7a6aae280a530651a5b58a/object.c#L616-L651

The result of this is that even though my decorator implements the Hash contract, even though it responds to #is_a?(Hash) and #kind_of?(Hash) with true, it cannot be considered a Hash in a case statement short of monkeypatching the core Hash#=== operator.

A proposed fix would be to simply replace the rb_obj_is_kind_of call with:

rb_funcall(arg, rb_intern("kind_of?"), mod)

A more aggressive fix might be to modify the implementation of rb_obj_is_kind_of but that is called at multiple other places in the source code.

This is likely only going to be acceptable for ruby-3.0 and be considered a potentially breaking change I would imagine.

History

#1 - 10/10/2018 06:01 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

I understand your needs but the receiver of === is a class/module, not the decorator. So if you want to override the behavior of the case statement, you need some complex mechanism like coercing. I am not positive about adding that kind of complex system since it could slow down every case statement.

Matz.