Ruby master - Feature #15240

Set operations check for is_a?(Set), rather than allowing duck typing

10/21/2018 07:45 PM - ivoanjo (Ivo Anjo)

Status:	Open
Priority:	Normal
Assignee:	knu (Akinori MUSHA)
Target version:	

Description

Hello there □□

Ruby's Set, unlike Array or Hash, cannot easily interoperate with user-created classes as several operations (#==, #flatten, #flatten!, #intersect?, #disjoint?, #subset?, #proper_subset?, #superset?, #proper_superset?) check that the other class is_a?(Set), rather than allowing duck-typing.

Example:

```
require 'set'
class MySet
  include Enumerable
  def each(&block) [:my, :set].each(&block) end
  def size() to_a.size end
end
puts Set[:set].subset?(MySet.new)

=> Traceback (most recent call last):
        1: from testcase.rb:8:in `<main>'
set.rb:292:in `subset?': value must be a set (ArgumentError)
```

The only way I've found of going around this issue and looking at the Ruby sources, is to fake a response to is_a?:

```
require 'set'
class MySet
  include Enumerable
  def each(&block) [:my, :set].each(&block) end
  def size() to_a.size end
  def is_a?(klass) super || klass == Set end # <== Hack! □□
end
puts Set[:set].subset?(MySet.new)

=> true
```

This is a very ugly hack, and instead it would be nice if, instead, I could just provide a to_set method that Set could call to allow duck typing.

I'm willing to work on a patch to solve this (would be pretty nice to do my first contribution to Ruby core!), so hopefully we can discuss how this problem can be tackled.

Background / TL;DR

This issue came about as I am the creator of a gem called persistent-□□This gem provides immutable arrays, hashes and sets. Most of the hard work is delegated to another gem (hamster), but I've added a number of tweaks to allow the persistent-□□ variants to easily interoperate with their Ruby counterparts.

Because I wanted to allow Persistent□□::Set instances to be used together with Ruby's Set, I studied the set.rb implementation and came up with the is_a?(Set) hack above. This works on all Ruby versions the gem supports (1.9->2.6), but broke on JRuby 9.2 when a new optimized Set implementation was added, that did not do the is_a?(Set) check and thus broke the hack.

I've brought up this issue with the JRuby developers -- https://github.com/jruby/jruby/issues/5227 -- and from there we moved the discussion to ruby/spec -- https://github.com/ruby/spec/pull/629.

We ended up concluding that it would make sense to raise this on the Ruby tracker as something that should be fixed on Set itself, rather than codifying this hack as something that Ruby is expected to support.

Since Ruby sets already support an implicit conversion method -- to_set -- it seems natural to replace the is_a?(Set) with some kind of other.respond_to?(:to_set) && other = other.to_set in all places where the is_a?(Set) was being used. Note that his would be all that's needed to be able to use a Set duck-type --- the Persistent::Set specs are a pretty good proof of it.

Thanks for the time , and rock on !

History

#1 - 10/21/2018 08:51 PM - shevegen (Robert A. Heiler)

From my own experience there is often a very good chance for extending duck typing possibilities in ruby if there is a specific (or at the least potential rather than abstract) use case.

You suggested the replacement (e. g. from going to .is_a? towards .respond_to? and I believe if there is no reason against it, then this may have a high chance to be accepted. Of course I can not speak for anyone else; I merely base this on my own experience in regards to ruby's duck typing opportunities (matz has lots of duck-pictures in his presentations too).

I would suggest to have this issue take some days for others to comment on it and chime in; and then, if you would like to, to consider adding it towards the next developer meeting at:

https://bugs.ruby-lang.org/issues/15229

That way may be the best to get feedback from matz and perhaps also approval; and at the very least some discussion, which may help resolve other parts that may require more information (but I think in this case, it is a rather simple issue request, so I believe there is not that much that may have to be discussed e. g. compared to larger issue requests such as adding pattern matching into ruby and similar).

#2 - 10/22/2018 04:23 AM - duerst (Martin Dürst)

Having a patch (using .respond_to?, I'd guess) would probably make acceptance much easier.

#3 - 10/22/2018 08:19 AM - Hanmac (Hans Mackowiak)

i didn't looked yet how Set is implemented, is it ruby code only or does it has some C coded parts too?

if it has C-Coded parts, i think the best way would be to:

- Check if object is real Set, if yes do real set stuff
- this maybe: check if object has #each ... like maybe iterating the objects might be faster than building a new set
- Check for #to_set method, if not throw Exception

I did look, it is ruby only coded, maybe Set should be moved into implemented in C?

hm i especially hate such part of code:

```
case
when set.instance_of?(self.class) && @hash.respond_to?(:<)
  @hash < set.instance_variable_get(:@hash)
when set.is_a?(Set)
  size < set.size && all? { |o| set.include?(o) }
else
  raise ArgumentError, "value must be a set"
end
```

my problem with such code is the empty case part ... it just looks ugly because it isn't better than a if ... elseif ... else ... end construct

#4 - 10/22/2018 09:53 PM - ivoanjo (Ivo Anjo)

Thanks everyone for the feedback! As suggested, I'll wait a few more days for more feedback, and then come up with an experimental patch that can serve as a basis for further suggestion (and for adding this to the next developer meeting).

@Hans: There is definitely a lot of opportunity to clean up and optimize Set -- JRuby got a new Java port for 9.2.0.0 and as a result got a nice speed up.

#5 - 10/27/2018 09:15 AM - Eregon (Benoit Daloze)

Hanmac (Hans Mackowiak) wrote:

> hm i especially hate such part of code:
>
> ```
> case
> when set.instance_of?(self.class) && @hash.respond_to?(:<)
>   @hash < set.instance_variable_get(:@hash)
> when set.is_a?(Set)
>   size < set.size && all? { |o| set.include?(o) }
> else
>   raise ArgumentError, "value must be a set"
> end
> ```
>
> my problem with such code is the empty case part ... it just looks ugly because it isn't better than a if ... elseif ... else ... end construct

Existing code style is orthogonal to this issue.
Please make a separate issue or pull request if you would like to change it :)

#6 - 10/27/2018 09:19 AM - Eregon (Benoit Daloze)

- Assignee set to knu (Akinori MUSHA)

knu (Akinori MUSHA) Could you share your opinion?

I also wonder, why are these operations raising an error with non-Set arguments?
Is it to avoid e.g. Array which would have a slow O(n) include? check?
Otherwise any Enumerable would work with the above mentioned Set comparison methods if methods were called directly.

#7 - 12/24/2018 09:59 AM - knu (Akinori MUSHA)

I can't recall everything in details, but I think interaction with other types of objects, especially comparison operations, was out of the scope when I wrote Set, and there were some points to consider in my mind.

First, implicit conversion in comparison has not been adopted in Ruby unlike in PHP and JavaScript where programmers often have hard times with it, and that was why I took a conservative path.

e.g.

```
o = Object.new
def o.to_int; 1; end
def o.to_ary; [1,2]; end
1 == o #=> false
[1,2] == o #=> false
```

We'd still have gone with it in Set, but it seemed to me that the apply to_set and compare strategy wouldn't work well with arrays for example, because that'd make [1, 2, 2] a subset of and equal to Set[1, 2].

Comparison of both size and elements (as proposed above) could have worked, but in retrospect, there was something called String that was Enumerable but the size did not reflect the number of enumerated elements. For those who don't know, in Ruby 1.8, String#size would return a byte size and #each would enumerate lines.

Those above are just my excuses; maybe it's time to think and move forward.

#8 - 01/01/2019 10:13 PM - ivoanjo (Ivo Anjo)

Thanks for chiming in, knu (Akinori MUSHA)!

Interestingly in https://github.com/ruby/spec/pull/629 we had discussed and were considering that the existing #to_set could be seen as an *implicit* conversion, but as you've pointed out, the current Enumerable#to_set monkey patch that loading set.rb adds definitely makes #to_set look a lot more like an *explicit* conversion -- as it does not make any sense for Set[1, 2] == [1, 2, 2], and in general for any random Enumerable to suddenly be treated like set.

Notice however that there's something missing in your to_ary example that makes it work:

```
o = Object.new
def o.to_ary; [1, 2]; end
def o.==(other); super || other == to_ary; end

[1, 2] == o #=> true
```

What Array actually does is check for respond_to?(:to_ary) and in that case it calls the == on our custom object. Ideally a similar behavior would be added to Set.

So perhaps if we were to consider #to_set to be the *explicit* conversion, we could introduce another one to be an implicit counterpart, like #to_ary or

#to_hash. Unfortunately, I can't think of a great name for it, so as a **working title only** I'll refer to it as #to_st.

What do you think of this updated proposal?

#9 - 05/05/2019 07:27 PM - ivoanjo (Ivo Anjo)

Any news on this one? Can we get the ball rolling? :)

I really don't mind doing the work, but would like to get a bit of feedback on the correct direction to make sure that the contribution does get accepted.

#10 - 08/27/2019 08:12 PM - jeremyevans0 (Jeremy Evans)

- Backport deleted (2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN)

- ruby -v deleted (ruby 2.5.3p105 (2018-10-18 revision 65156) [x86_64-linux])

- Tracker changed from Bug to Feature

- File set-duck-typing-15240.patch added

I do not think the current behavior is a bug. However, supporting implicit conversion seems like a useful feature to add. I've implemented ivoanjo's proposal of to_st as an implicit conversion method in the attached patch.

Files

set-duck-typing-15240.patch	6.85 KB	08/27/2019	jeremyevans0 (Jeremy Evans)