Ruby master - Bug #15027

When Struct#each method is overriden Struct#select and Struct#to_a use wrong collections

08/25/2018 05:29 PM - bruno (Bruno Sutic)

Status:	Feedback		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	2.6.0dev	**Backport:**	2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN

Description

Bug

Here's the code snippet that should reproduce the problem:

```
class Foo < Struct.new(:bar)
  def each(&block)
    [:baz, :qux].each(&block)
  end
end

foo = Foo.new(:foo)
foo.map(&:itself) # => [:baz, :qux]  # OK

foo.to_a # => [:foo]  # NOT OK, expected [:baz, :qux]
foo.select(&:itself) # => [:foo]  # NOT OK, expected [:baz, :qux]
```

As you can see, even tho we defined another collection for use by overriding #each, the to_a and select still use Struct's original collection.

The problem seem to be with Struct#to_a and Struct#select methods from struct.c file that are defined unnecessarily.

Proposed solution

The attached solution simply deletes Struct#select and Struct#to_a. A couple tests are added to show everything still works as before.

Please let me know if I can provide any more info and I'll be ready to do so.

History

#1 - 08/26/2018 01:56 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

The proposed behavior is more consistent but slower. I am not sure it's a good idea to hinder performance by supporting consistency in the rare case. Any opinion?

Matz.

#2 - 08/26/2018 02:46 AM - jeremyevans0 (Jeremy Evans)

I'm against changing the current behavior. If you can override each, you can override other methods. If we wanted to be consistent about making this change, we would probably need to remove most the following additional core methods so that the Enumerable implementation would be used:

```
Array: any?, collect, count, cycle, drop, drop_while, find_index, first, include?, map, max, min, reject, reve
rse_each, select, sort, sum, take, take_while, to_a, to_h, uniq, zip
Hash: any?, include?, member?, to_a, to_h
Range: first, include?, max, member?, min
Struct: to_h
```

Above results generated with the following code (which includes some additional classes and methods):

```
classes = []
```

```
meths = Enumerable.instance_methods(false) - [:each]

ObjectSpace.each_object(Class){|c| classes << c if c < Enumerable && c.name}

classes.sort_by(&:name).each do |c|
  same = c.instance_methods(false) & meths
  next if same.empty?
  puts "#{c}: #{same.sort.join(', ')}"
end
```

I'm guessing the standard library would need additional changes (e.g. set).

If anyone really wants the default Enumerable behavior for Struct:

```
Struct.remove_method(:to_a, :to_h, :select)
```

#3 - 08/26/2018 06:53 AM - shevegen (Robert A. Heiler)

Since matz asked for feedback, just a comment - I have no particular pro or con opinion per se, mostly because I very rarely use the Struct/OpenStruct family; I usually just end up writing a "real" class instead and adapt it to what is necessary. The only suggestion I would like to make in regards to Struct is to mention this in the documentation of Struct (https://ruby-doc.org/core-2.5.1/Struct.html) so that it may not surprise ruby users who use Struct; could also mention Jeremy's example of using the default Enumerable behaviour for Struct.

#4 - 08/26/2018 12:36 PM - Hanmac (Hans Mackowiak)

as a middle way, can't we just do the "is overwritten by user" check?
i think i have seen it on other classes like Array or Hash

so it checks if the each method is overwritten, in that case, call the Enumerable method,
if not, call the Struct own one.

#5 - 08/29/2018 02:12 PM - nobu (Nobuyoshi Nakada)

```
diff --git a/struct.c b/struct.c
index 7de46980aa..e4c875b5be 100644
--- a/struct.c
+++ b/struct.c
@@ -860,6 +860,9 @@ rb_struct_inspect(VALUE s)
 static VALUE
 rb_struct_to_a(VALUE s)
 {
+    if (!rb_method_basic_definition_p(CLASS_OF(s), idEach)) {
+        return rb_call_super(0, 0);
+    }
     return rb_ary_new4(RSTRUCT_LEN(s), RSTRUCT_CONST_PTR(s));
 }

@@ -1077,6 +1080,9 @@ rb_struct_select(int argc, VALUE *argv, VALUE s)

     rb_check_arity(argc, 0, 0);
     RETURN_SIZED_ENUMERATOR(s, 0, 0, struct_enum_size);
+    if (!rb_method_basic_definition_p(CLASS_OF(s), idEach)) {
+        return rb_call_super(argc, argv);
+    }
     result = rb_ary_new();
     for (i = 0; i < RSTRUCT_LEN(s); i++) {
    if (RTEST(rb_yield(RSTRUCT_GET(s, i)))) {
```

#6 - 09/01/2018 04:14 PM - bruno (Bruno Sutic)

- File struct_enumerable_fix2.patch added

Thank you for the code @nobu. I have applied Nobu's suggestion, please find updated patch in the attach.

> If you can override each, you can override other methods.

This is true, but doing this is very un-ruby-ish. Also, I don't think an average ruby developer will be patient enough to investigate why is this happening. A more realistic scenario is that a developer will just implement a custom #select or #to_a method.

#7 - 09/03/2018 09:07 AM - Hanmac (Hans Mackowiak)

matz (Yukihiro Matsumoto): is the patch from Nobu good enough?

#8 - 09/26/2018 10:24 AM - bruno (Bruno Sutic)

Hi,

let me know if you have any further feedback on this one?

Files

struct_enumerable_fix.patch	2.8 KB	08/25/2018	bruno (Bruno Sutic)
struct_enumerable_fix2.patch	1.51 KB	09/01/2018	bruno (Bruno Sutic)