Ruby master - Bug #4136

Enumerable#reject should not inherit the receiver's instance variables

12/09/2010 03:36 AM - hasari (Hiro Asari)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 1.9.3dev (2010-11-28 trunk 29965) [x86_64-darwin10.5.0]	**Backport:**	

Description

=begin
re
Below, you see that a.reject returns a copy of the receiver, which inherits the instance variable @foo. This is not the case with Array#select.

```
irb(main):001:0> a=[]
=> []
irb(main):002:0> a.instance_variable_set "@foo", "bar"
=> "bar"
irb(main):003:0> a.reject {}.instance_variable_get "@foo"
=> "bar"
irb(main):004:0> a.select {}.instance_variable_get "@foo"
=> nil
```

1.8.x behaves the same way.
=end

Related issues:

Related to Ruby master - Bug #7625: Array□□□□□□□□□□□□compact□Array□□□	**Closed**	**12/26/2012**
Related to Ruby master - Bug #7768: Inherited Array class missing	**Closed**	

History

#1 - 12/09/2010 04:45 AM - headius (Charles Nutter)

=begin
I find this behavior unintuitive. #reject returns a *new* array, which I would not expect to have instance variables from the old array. It could, in fact, drag along data I don't intend for it to drag along, with no obvious way to scrub that data out other than manually *removing* instance vars on the new object.

Also note that this unnecessarily impacts the perf of reject when ivars are present, and if people want this behavior, .dup.reject! is an easy way to get it.
=end

#2 - 12/09/2010 07:15 AM - hasari (Hiro Asari)

=begin
In the similar vein, if you subclass Array, that class's #reject returns an object of that subclass, rather than an Array.

```
$ irb
irb(main):001:0> RUBY_DESCRIPTION
=> "ruby 1.8.7 (2009-06-12 patchlevel 174) [universal-darwin10.0]"
irb(main):002:0> class A < Array; def foo; return "yo"; end; end
=> nil
irb(main):003:0> a=A.new
=> []
irb(main):004:0> a.reject {}.foo
=> "yo"
irb(main):005:0> a.collect { true }.foo
NoMethodError: undefined method `foo' for []:Array
from (irb):5
from :0
```

This is counterintuitive, at the least.
=end

#3 - 12/09/2010 03:47 PM - matz (Yukihiro Matsumoto)

- Status changed from Open to Closed

- % Done changed from 0 to 100

=begin
This issue was solved with changeset r30148.
Hiro, thank you for reporting this issue.
Your contribution to Ruby is greatly appreciated.
May Ruby be with you.

=end

#4 - 12/09/2010 04:30 PM - zenspider (Ryan Davis)

=begin

On Dec 8, 2010, at 10:37 , Hiro Asari wrote:

```
    irb(main):001:0> a=[]
    => []
    irb(main):002:0> a.instance_variable_set "@foo", "bar"
    => "bar"
    irb(main):003:0> a.reject {}.instance_variable_get "@foo"
    => "bar"
    irb(main):004:0> a.select {}.instance_variable_get "@foo"
    => nil
```

that's an awesome find.

=end

#5 - 12/09/2010 07:16 PM - mudge (Paul Mucur)

=begin
I have attempted to codify this new behaviour in RubySpec in https://github.com/rubyspec/rubyspec/pull/31 and would appreciate any feedback.
=end

#6 - 12/14/2010 07:42 AM - marcandre (Marc-Andre Lafortune)

- Category set to core

=begin
This changes the behavior for subclasses of Array. Should the other cases also be modified in the same way?

If I check the list I had in my blog (see the quiz at bottom of http://blog.marc-andre.ca/2009/05/schizo-ruby-puzzle.html), the following (at least) are remaining:

```
Sub = Class.new(Array)
x = Sub.new
(x * 2).class # => Sub
x.flatten.class # => Sub
x[0...0].class # => Sub
```

=end

#7 - 12/14/2010 09:32 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:33704] [Ruby 1.9-Bug#4136] Enumerable#reject should not inherit the receiver's instance variables"
on Tue, 14 Dec 2010 07:42:26 +0900, Marc-Andre Lafortune redmine@ruby-lang.org writes:

|This changes the behavior for subclasses of Array. Should the other cases also be modified in the same way?

If a method is originally defined in Enumerable, i.e. its return value (Array)
is a collection of values from enumerable.

|If I check the list I had in my blog (see the quiz at bottom of http://blog.marc-andre.ca/2009/05/schizo-ruby-puzzle.html), the following (at least) are remaining:
|
|Sub = Class.new(Array)

|x = Sub.new
|(x * 2).class # => Sub
|x.flatten.class # => Sub
|x[0...0].class # => Sub

I don't think so. #flatten is not an enumerable method. Please point
out if we missed some other methods.

```
                                                        matz.
```

=end

#8 - 12/15/2010 08:36 PM - Eregon (Benoit Daloze)

=begin
On 15 December 2010 05:32, Marc-Andre Lafortune
ruby-core-mailing-list@marc-andre.ca wrote:

> Or similarly, why does:
>
> (x.slice(0,0)).class # => Sub
> # while...
> (x.slice!(0,0)).class # => Array
>
> ## Thanks
>
> Marc-André

I suppose you already know this, but it might help others:

x.slice!(0, 1).class # => Sub
x.slice!(0..0).class # => Sub

x.slice!(0, 0).class # => Array
x.slice!(1, 0).class # => Array
x.slice!(1..0).class # => Array

So, it seems any arguments to #slice which return some kind of Array
except slice(n, 0) and slice(n, <n) return a Sub.

This is inconsistent IMHO, because it should return the same class,
even when the array is empty (for the cases it returns some kind of
Array).
Also, a mutating method should (most of the time) not change an object's class.

Regards,
B.D.

=end