Ruby master - Bug #1393

Multiple assignment does not call to_a but to_ary

04/21/2009 10:39 AM - scritch (Vincent Isambart)

Status:	Rejected		
Priority:	Normal		
Assignee:	matz (Yukihiro Matsumoto)		
Target version:	1.9.2		
ruby -v:	1.9.1p0, 1.9.2dev (2009-04-19 trunk 23210)	**Backport:**	

Description

=begin
In Ruby 1.9, when Ruby is expecting an array and does not get one (for example in splat arguments), it calls the to_a method on the object.
For example, if x is not an array, "y = *x" tries to call x.to_a.
However, in multiple assignments, it does not try to call to_a but to_ary.
For example, "*y = x" tries to call x.to_ary.
Should not Ruby try to call to_a in both cases? It looks like a bug to me.
Is there a particular reason this was done?
=end

History

#1 - 07/16/2009 04:30 PM - yugui (Yuki Sonoda)

- Assignee set to ko1 (Koichi Sasada)

=begin

=end

#2 - 09/07/2009 04:51 PM - yugui (Yuki Sonoda)

- Status changed from Open to Assigned

- Target version set to 1.9.2

=begin

=end

#3 - 03/25/2010 01:23 AM - mame (Yusuke Endoh)

- Category set to core

- Assignee changed from ko1 (Koichi Sasada) to matz (Yukihiro Matsumoto)

=begin
Hi,

2009/4/21 Vincent Isambart redmine@ruby-lang.org:

> In Ruby 1.9, when Ruby is expecting an array and does not get one (for example in splat arguments), it calls the to_a method on the object.
> For example, if x is not an array, "y = *x" tries to call x.to_a.
> However, in multiple assignments, it does not try to call to_a but to_ary.
> For example, "*y = x" tries to call x.to_ary.

This issue is still reproducable:

```
obj = Object.new
def obj.to_a; [1, 2]; end
def obj.to_ary; [3, 4]; end
def foo(x, y); p [x, y]; end

a, b = *obj; p [a, b]  #=> [1, 2] on 1.9, [3, 4] on 1.8
foo(*obj)              #=> [1, 2] on 1.9, [3, 4] on 1.8
```

> Should not Ruby try to call to_a in both cases? It looks like a bug to me.

Agreed. I think that to_ary should be called because the both cases do
implicit conversion.

> Is there a particular reason this was done?

I guess that it is just a slip-up when removing to_splat (r14063).

Matz, is my guess correct?
Yugui, do you allow the spec change (fix) in 1.9.2?

FYI: if this is fixed, 21 failures/errors occur on RubySpec.
They depend on the current behavior implicitly and explicitly, such as:

```
[*nil].should == []

a = loop do break *nil; end; a.should == []

obj = Object.new
def obj.to_a
[1, 2]
end
return *obj
```

--
Yusuke ENDOH mame@tsg.ne.jp
=end

#4 - 04/14/2010 09:11 PM - mame (Yusuke Endoh)

- Priority changed from Normal to 5

=begin
Hi, Matz

Could you please answer this ticket?

> For example, if x is not an array, "y = *x" tries to call x.to_a.
> However, in multiple assignments, it does not try to call to_a but to_ary.
> For example, "*y = x" tries to call x.to_ary.

This is philosophical, but really significant bug against
convention of Ruby's type conversion, I think. I can't
ignore this without your response.

--
Yusuke Endoh mame@tsg.ne.jp
=end

#5 - 04/27/2010 11:32 AM - matz (Yukihiro Matsumoto)

=begin
Hi,

In message "Re: [ruby-core:29509] [Bug #1393] Multiple assignment does not call to_a but to_ary"
on Wed, 14 Apr 2010 21:11:35 +0900, Yusuke Endoh redmine@ruby-lang.org writes:

```
|Could you please answer this ticket?
|
|> For example, if x is not an array, "y = *x" tries to call x.to_a.
|> However, in multiple assignments, it does not try to call to_a but to_ary.
|> For example, "*y = x" tries to call x.to_ary.
|
|This is philosophical, but really significant bug against
|convention of Ruby's type conversion, I think. I can't
|ignore this without your response.
```

OK, to_a means an explicit array conversion, so it should not be used
for implicit conversion a in "*y = x*". *On the other hand, I consider
"*x" as a form of explicit conversion, i.e. shorthand for* "(x.to_a)",
so that the current 1.9 behavior is intentional.

```
matz.
```

=end

#6 - 04/27/2010 11:33 AM - matz (Yukihiro Matsumoto)

- Status changed from Assigned to Rejected

=begin

=end

#7 - 04/27/2010 12:45 PM - mame (Yusuke Endoh)

=begin
Hi,

I see. Thank you for your clarifying!

--
Yusuke Endoh mame@tsg.ne.jp
=end