Ruby - Bug #12706

Hash#each yields inconsistent number of args

08/25/2016 06:43 PM - bughit (bug hit)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description

```
def foo(a, b)
  p [a, b]
end

def bar(a, b = 2)
  p [a, b]
end

foo_lambda = method(:foo).to_proc
bar_lambda = method(:bar).to_proc

{a: 1}.each(&foo_lambda)
{a: 1}.each(&bar_lambda)
```

From #12705, yielding to method lambdas uses lambda/method arg semnatics

the yield to foo produces [:a, 1] suggesting that each is yielding two values yield key, value
but yield to bar produces [[:a, 1], 2] suggesting that each is yielding one value yield [key, value]

it would be better if you always knew what to expect from it

Related issues:

Related to Ruby - Bug #16948: hash.each(&method(:something)) behavior changed...	**Closed**
Related to Ruby - Bug #17197: Some Hash methods still have arity 2 instead of 1	**Rejected**

Associated revisions

Revision 47141797bed55eb10932c9a722a5132f50d4f3d8 - 03/16/2020 02:17 PM - mame (Yusuke Endoh)

hash.c: Do not use the fast path (rb_yield_values) for lambda blocks

As a semantics, Hash#each yields a 2-element array (pairs of keys and
values). So, { a: 1 }.each(&->(k, v) { }) should raise an exception
due to lambda's arity check.
However, the optimization that avoids Array allocation by using
rb_yield_values for blocks whose arity is more than 1 (introduced at
b9d29603375d17c3d1d609d9662f50beaec61fa1 and some commits), seemed to
overlook the lambda case, and wrongly allowed the code above to work.

This change experimentally attempts to make it strict; now the code
above raises an ArgumentError. This is an incompatible change; if the
compatibility issue is bigger than our expectation, it may be reverted
(until Ruby 3.0 release).

[Bug #12706]

Revision 47141797bed55eb10932c9a722a5132f50d4f3d8 - 03/16/2020 02:17 PM - mame (Yusuke Endoh)

hash.c: Do not use the fast path (rb_yield_values) for lambda blocks

As a semantics, Hash#each yields a 2-element array (pairs of keys and
values). So, { a: 1 }.each(&->(k, v) { }) should raise an exception
due to lambda's arity check.
However, the optimization that avoids Array allocation by using
rb_yield_values for blocks whose arity is more than 1 (introduced at

b9d29603375d17c3d1d609d9662f50beaec61fa1 and some commits), seemed to
overlook the lambda case, and wrongly allowed the code above to work.

This change experimentally attempts to make it strict; now the code
above raises an ArgumentError. This is an incompatible change; if the
compatibility issue is bigger than our expectation, it may be reverted
(until Ruby 3.0 release).

[Bug #12706]

Revision 47141797 - 03/16/2020 02:17 PM - mame (Yusuke Endoh)

hash.c: Do not use the fast path (rb_yield_values) for lambda blocks

As a semantics, Hash#each yields a 2-element array (pairs of keys and
values). So, { a: 1 }.each(&->(k, v) { }) should raise an exception
due to lambda's arity check.
However, the optimization that avoids Array allocation by using
rb_yield_values for blocks whose arity is more than 1 (introduced at
b9d29603375d17c3d1d609d9662f50beaec61fa1 and some commits), seemed to
overlook the lambda case, and wrongly allowed the code above to work.

This change experimentally attempts to make it strict; now the code
above raises an ArgumentError. This is an incompatible change; if the
compatibility issue is bigger than our expectation, it may be reverted
(until Ruby 3.0 release).

[Bug #12706]

History

#1 - 03/16/2020 08:30 AM - matz (Yukihiro Matsumoto)

It was caused by the optimization introduced in 2.1. It should check if a block is a lambda before making optimization.
We worry about compatibility but let's fix it in 2.8(3.0) and see it can cause problems. Please mark the change as **experimental** .

Matz.

#2 - 03/16/2020 02:17 PM - mame (Yusuke Endoh)

- Status changed from Open to Closed

Applied in changeset git|47141797bed55eb10932c9a722a5132f50d4f3d8.

---

hash.c: Do not use the fast path (rb_yield_values) for lambda blocks

As a semantics, Hash#each yields a 2-element array (pairs of keys and
values). So, { a: 1 }.each(&->(k, v) { }) should raise an exception
due to lambda's arity check.
However, the optimization that avoids Array allocation by using
rb_yield_values for blocks whose arity is more than 1 (introduced at
b9d29603375d17c3d1d609d9662f50beaec61fa1 and some commits), seemed to
overlook the lambda case, and wrongly allowed the code above to work.

This change experimentally attempts to make it strict; now the code
above raises an ArgumentError. This is an incompatible change; if the
compatibility issue is bigger than our expectation, it may be reverted
(until Ruby 3.0 release).

[Bug #12706]

#3 - 03/19/2020 01:27 AM - Eregon (Benoit Daloze)

Does this cause any issue in practice?

AFAIK it's not worth the incompatibility and could break many things.
We had to follow MRI behavior here for Hash#each and Hash#map in TruffleRuby, e.g., https://github.com/oracle/truffleruby/issues/1944

IMHO the right thing to do is to yield 2 values here, and having an Array for backward compatibility if arity != 2 seems OK.

#4 - 06/10/2020 05:26 PM - jeremyevans0 (Jeremy Evans)

- Related to Bug #16948: hash.each(&method(:something)) behavior changed without warning on master added

#5 - 06/10/2020 07:33 PM - marcandre (Marc-Andre Lafortune)

Interesting. Does it intend to fix just this case, or any inconsistencies I listed in https://bugs.ruby-lang.org/issues/14015 ?

@Eregon (Benoit Daloze): given the current state of affairs, I consider lambdas & multiple yield a very bad idea until things make some sense.

#6 - 11/16/2020 05:18 AM - nobu (Nobuyoshi Nakada)

- Related to Bug #17197: Some Hash methods still have arity 2 instead of 1 added

#7 - 08/05/2021 02:10 PM - mk (Matthias Käppler)

I just ran into this since I hadn't been aware of this change.

What I find odd about this change is that it introduces a new inconsistency: the behavior of related Enumerable methods such as map is now different to that of each (in fact, isn't map implemented in terms of each at the VM level?)

Using the bug author's example, I find the following behavior in Ruby 3 at least equally surprising:

```
irb(main):058:0> {a: 1}.each(&foo_lambda)
(irb):44:in `foo': wrong number of arguments (given 1, expected 2) (ArgumentError)
 from (irb):58:in `each'
 from (irb):58:in `<main>'
 from /home/mk/.rbenv/versions/3.0.2/lib/ruby/gems/3.0.0/gems/irb-1.3.5/exe/irb:11:in `<top (required)>'
 from /home/mk/.rbenv/versions/3.0.2/bin/irb:23:in `load'
 from /home/mk/.rbenv/versions/3.0.2/bin/irb:23:in `<main>'
irb(main):059:0>
irb(main):060:0> {a: 1}.map(&foo_lambda)
[:a, 1]
=> [[:a, 1]]
```

Why does each fail but map succeeds?