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 semantics

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

Associated revisions

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
b9d29603375d17c3d1d609d9662f50bea9e41fa1 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 if 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
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.