Ruby master - Bug #12036

Enumerator's automatic rewind behavior

01/29/2016 09:30 PM - slash_nick (Ryan Hosford)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	2.0.0: UNKNOWN, 2.1: UNKNOWN, 2.2: UNKNOWN, 2.3: UNKNOWN

Description

When enumerating an enumerator, the enumerator automatically rewinds when #next raises an error. The concern here is that someone may need to handle that error and continue processing the rest of the enumerator.

12:22 < Ox0dea> I think tjohnson found where the StopIteration was being raised?
12:23 < tjohnson> A friend in #projecthydra pointed me here: https://github.com/ruby/ruby/blob/trunk/enumerator.c#L655-L660
12:24 < tjohnson> seems quite deliberate. compare: https://github.com/ruby/ruby/blob/trunk/enumerator.c#L925-L930
12:25 < Ox0dea> Yeah, Line 651 in there is the one birthing the new Fiber, and since dead ones don't maintain context, the surrounding Enumerator can't pick up where it left off.
12:26 < Ox0dea> It's almost certainly intentional, and most likely even The Right Thing, but "make easy things easy and hard things *possible*".

tjohnson first demonstrated the behavior in a gist: https://gist.github.com/no-reply/4b38f26b3fe32ad266a7#gistcomment-1683794
lucasb also produced a snippet that clearly demonstrates the behavior: https://eval.in/509874

Thanks:
0x0dea, lucasb, tjohnson, rthbound

History

#1 - 09/06/2017 04:29 AM - kernigh (George Koehler)

I don't want Ruby to change this behavior. Enumerable#next can't continue the iteration if it raises an exception. That's why it has the automatic rewind behavior.

Here's a quick example. In this script, we handle the oops and expect the last e.next to return 44, but there is automatic rewind so it returns 11.

```
def peach
  yield 11
  yield 22
  fail 'oops'
  yield 44
end

e = enum_for(:peach)
puts e.next               #=> 11
puts e.next               #=> 22
(e.next rescue puts $!)   #=> oops
puts e.next               #=> 11
```

Now we want to change the behavior of Enumerable#next so it would return 44. This seems impossible, because #peach always raises an error and never reaches yield 44. To make this change, we would need to modify Ruby to divert some errors between fibers. When the fiber running #peach tries to raise an error, Ruby would switch to the fiber running Enumerable#next before raising the error. Later, if the program switches back to the fiber of #peach, *then #peach would continue as if it had never raised an error.* With this change, the caller of #next can rescue the error, and #peach can reach yield 44.

In our modified Ruby, Enumerable#next would divert errors that should never be diverted. The next script might crash our modified Ruby.

```
def teach
  "str" + 2
end

e = enum_for(:teach)
(e.next rescue puts $!)
#=> no implicit conversion of Integer into String
e.next
#=> crash Ruby???
```

We rescue a TypeError from String#+, then tell Ruby to continue the iteration. Our modified Ruby wouldn't have automatic rewind, so the fiber running #teach would continue the execution of String#+ as if it never raised TypeError. Then the C code implementing String#+ might use 2 as a String and crash Ruby! This is why I want Ruby to keep automatic rewind.

#2 - 07/24/2019 05:19 PM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Closed