Ruby master - Bug #16983

RubyVM::AbstractSyntaxTree.of(method) returns meaningless node if the method is defined in eval

06/24/2020 12:57 PM - pocke (Masataka Kuwabara)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.8.0dev (2020-06-23T13:58:26Z master dc351ff984) [x86_64-linux]	**Backport:**	2.5: UNKNOWN, 2.6: UNKNOWN, 2.7: UNKNOWN

Description

Problem

RubyVM::AST.of(method) returns a meaningless node if the method is defined in eval.
For example:

```
p 'blah'

eval <<~RUBY, binding, __FILE__, __LINE__ + 1
  def foo
  end
RUBY

method = method(:foo)
pp RubyVM::AbstractSyntaxTree.of(method)
# => (STR@3:5-3:12 "def foo\n" + "end\n")
```

I expect the node of foo method, or nil. But it returns a STR node.

It becomes a big problem when AST.of receives arbitrary methods.
Because we can't distinguish a method is defined in eval or not.
It means we can't believe the returned value of AST.of if the method may receive a method defined in eval.

For example:

```
def do_something_for_each_method_ast(klass)
  klass.instance_methods(false).each do |m|
    ast = RubyVM::AbstractSyntaxTree.of(klass.instance_method(m))
    next unless ast

    do_something ast
  end
end

class A
  eval <<~RUBY, binding, __FILE__, __LINE__ + 1
    def foo
    end
  RUBY
end

do_something_for_each_method_ast A
```

In the example, I expect the do_something method receives only node for a method definition,
but it may pass a wrong node if any method is defined in eval.

Cause (I guess)

I guess the cause is misleading node number.
In and out of an eval block uses different sequences of node number.
So if I specify __FILE__ to eval, the actual file and code in eval may have the same node number.

For example

```
p 'blah' # Node number for 'blah' is 1, file name is "test.rb"

eval <<~RUBY, binding, __FILE__, __LINE__ + 1
  def foo # Node number for `def` is also 1, file name is also "test.rb"
  end
RUBY

method = method(:foo)
# It finds a node from node number 1 by reading "test.rb", so it get the str node.
pp RubyVM::AbstractSyntaxTree.of(method)
# => (STR@3:5-3:12 "def foo\n" + "end\n")
```

History

#1 - 08/28/2020 09:35 PM - jeremyevans0 (Jeremy Evans)

I'm not sure if this is a bug, but it does seem like a fundamental and significant limitation with the design of RubyVM::AbstractSyntaxTree.of. RubyVM::AbstractSyntaxTree.of reparses the file the method is defined in and cannot handle any cases where eval or similar are used. You'll get a node completely different from what you would expect. Here's another example:

```
eval DATA.read, binding, __FILE__, 14
method = method(:foo)
pp RubyVM::AbstractSyntaxTree.of(method)

__END__
  def foo
  end
```

Output:

```
(VCALL@1:16-1:23 :binding)
```

Because it reparses the file, you'll also get the wrong result if the file is modified:

```
def bar
end

File.write(__FILE__, File.read(__FILE__).gsub('def bar', "def foo\nbar"))

method = method(:bar)
pp RubyVM::AbstractSyntaxTree.of(method)
```

Output:

```
(SCOPE@1:0-3:3
 tbl: []
 args:
   (ARGS@1:7-1:7
    pre_num: 0
    pre_init: nil
    opt: nil
    first_post: nil
    post_num: 0
    post_init: nil
    rest: nil
    kw: nil
    kwrest: nil
    block: nil)
 body: (VCALL@2:0-2:3 :bar))
```

And if the interpreter can no longer access the file (chroot, file deletion, permission change, or other file system access limiting), you get an error.

I can't think of a way to fix this without all iseq methods holding a reference to the string used to parse them, and having RubyVM::AbstractSyntaxTree.of work off that string. I'm not sure how much extra memory use that would cause, or if such an approach is considered acceptable.