SyntaxError in endless method

12/16/2020 07:52 PM - zverok (Victor Shepelev)

Status: Closed
Priority: Normal
Assignee:
Target version:

Description
This works:

```ruby
def foo() = puts("bar")
```

This does not:

```ruby
def foo() = puts "bar"
# syntax error, unexpected string literal, expecting `do' or '{' or '('
```

Is this intentional or accidental? Not sure how it is reasoned.

Associated revisions
Revision 31794d2e - 05/12/2021 03:14 PM - mame (Yusuke Endoh)
parse.y: Allow "command" syntax in endless method definition
This change allows def hello = puts "Hello" without parentheses.
Note that private def hello = puts "Hello" does not parse for
technical reason.
[Feature #17398]

Revision 1da1d747 - 12/13/2021 05:01 AM - mame (Yusuke Endoh)
NEWS.md: def foo = puts "Hello" is now allowed [Feature #17398]

History
#1 - 12/17/2020 12:49 AM - mame (Yusuke Endoh)
The body of an endless method must be an expression (called "arg" in the syntax rules of parse.y). puts("bar") is an expression, but puts "bar" is a
statement (called "command" in the syntax rules).
I think it could be a bit confusing, but I have no idea whether we can/should allow a statement as a body of an endless method.

#2 - 12/17/2020 09:39 AM - zverok (Victor Shepelev)
mame (Yusuke Endoh) Hmm, haven't thought about it from this perspective... Can you please explain a bit? As far as I can see, in, say, assignment
context it behaves like an expression:
```ruby
result = puts 'foo'
# prints "foo", result = nil
```
I am just trying to describe the behavior in full for the next installment of my changelog and this aspect is quite confusing for me... Though, it is not
endless-method specific, as far as I can see:
```ruby
y = sin x # OK
y = 1 + sin x
# ^ unexpected local variable or method, expecting `do' or '{' or '('
```
What's the "rule of thumb" to understand this better?

#3 - 12/17/2020 01:52 PM - Eregon (Benoit Daloze)
Conceptually, according to the typical definition in computer science, both puts("bar") and puts "bar" are expressions (i.e., they return a value, and if it
was some other method than puts it would also not always be nil).
It might be slightly less clear for e.g. a = 42 (it's still an expression, it still returns a value), but I think puts "bar" is clear that it should be the same as
puts("bar"), except for precedence.
So it's probably going to be very difficult to explain the actual condition, other than showing specific examples.

Endless methods definitions don't support poetry mode?

The following patch allows def foo() = puts "bar". It brings no parser conflict.

https://gist.github.com/mame/0773bf3938e046e2b608de5f5b2a826c8

However, it is not perfect. private def foo() = puts "foo" does not parse.
private var = puts "bar" is not allowed neither, so I have no idea how to allow this.

If I were young, I would add mame (Yusuke Endoh)'s patch. I did similar decisions many times in the past. But Ruby has been mature and complex, I now feel reluctant. Let us consider this idea for a while.

Matz.

I'm attaching an updated patch with a test.

I have considered this issue for a while and concluded it should be merged to be consistent with the assignment statement.

Matz.

Applied in changeset git|31794d2e733e081e4e22127aff6380393981681.

parse.y: Allow "command" syntax in endless method definition

This change allows def hello = puts "Hello" without parentheses.

Note that private def hello = puts "Hello" does not parse for technical reason.

[Feature #17398]

Files

allow-command-style-endless-method-def.patch 3.44 KB 04/19/2021 mame (Yusuke Endoh)