Ruby trunk - Bug #15404

Endless range has inconsistent chaining behaviour

12/12/2018 01:08 PM - valich (Valentin Fondaratov)

Status:	Open		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	ruby 2.6.0rc1 (2018-12-06 trunk 66253) [x86_64-linux]	**Backport:**	2.4: UNKNOWN, 2.5: UNKNOWN

Description

Everything below is tested on Ruby 2.6.0-rc1. Particular sexp column coordinates are wrong because I've had some leading spaces in the file, sorry.

The essence of the bug

Syntactically, chaining normal ranges is prohibited. For example,
(1..1)..1 produces the following sexp output:

```
[:program,
 [[:dot2,
   [:paren, [[:dot2, [:@int, "1", [1, 16]], [:@int, "1", [1, 19]]]]],
   [:@int, "1", [1, 23]]]]]
```

while
1..1..1 is a syntax error (compiler output: syntax error, unexpected ..)

New endless ranges break this behaviour and allow chaining.
There are two bugs.

1.

Chaining is possible on one line:
1.. ..1 is parsed as

```
[:program,
 [[:dot2, [:dot2, [:@int, "1", [1, 15]], nil], [:@int, "1", [1, 21]]]]]
```

I think this is inconsistent compared to the previous case.

2.

Chaining works even with newline between two parts:

```
1..
..1
```

```
[:program,
 [[:dot2, [:dot2, [:@int, "1", [1, 15]], nil], [:@int, "1", [2, 17]]]]]
```

This behaviour is completely counterintuitive because 1.. on the first line is a complete statement. Even if it continues to the next line with the search for the right part of expression (end range), it should break because ..1 is not a syntactically valid range end. So, in the search for the end range parser decides to complete the first range and use it as a beginning. It contradicts older

```
1
..2
```

behaviour which effectively meant that a range could not be continued to the next line.

Why it's important

All the code above will break on runtime because it leads to bad value for range (ArgumentError). However, if the code is located in some method (or branch) which is executed rarely, developer might miss the problem.

History

#1 - 12/12/2018 01:09 PM - valich (Valentin Fondaratov)

- ruby -v changed from 2.6.0-rc1 to ruby 2.6.0rc1 (2018-12-06 trunk 66253) [x86_64-linux]

#2 - 12/12/2018 03:44 PM - mame (Yusuke Endoh)

Thank you for your report.

valich (Valentin Fondaratov) wrote:

> ## Why it's important
>
> All the code above will break on runtime because it leads to bad value for range (ArgumentError). However, if the code is located in some method (or branch) which is executed rarely, developer might miss the problem.

I understand the problem. But, it is not specific to endless range, is it? In fact, (1..1)..1 does parse, and fails to run.

It is possible to prohibit all nested ranges, including a non-endless range (1..1)..1 and an endless range (1..)..1. This brings very small incompatibility which is acceptable IMO.

Nobu's patch:

```
diff --git i/parse.y w/parse.y
index 278c5b0296..1c76af541a 100644
--- i/parse.y
+++ w/parse.y
@@ -380,6 +380,8 @@ static void void_expr(struct parser_params*,NODE*);
 static NODE *remove_begin(NODE*);
 static NODE *remove_begin_all(NODE*);
 #define value_expr(node) value_expr_gen(p, (node) = remove_begin(node))
+static int range_expr_gen(struct parser_params*,NODE*);
+#define range_expr(node) range_expr_gen(p, (node) = remove_begin(node))
 static NODE *void_stmts(struct parser_params*,NODE*);
 static void reduce_nodes(struct parser_params*,NODE**);
 static void block_dup_check(struct parser_params*,NODE*,NODE*);
@@ -1909,8 +1911,8 @@ arg       : lhs '=' arg_rhs
         | arg tDOT2 arg
             {
             /*%%%*/
-            value_expr($1);
-            value_expr($3);
+            range_expr($1);
+            range_expr($3);
             $$ = NEW_DOT2($1, $3, &@$);
             /*% %*/
             /*% ripper: dot2!($1, $3) %*/
@@ -1918,8 +1920,8 @@ arg       : lhs '=' arg_rhs
         | arg tDOT3 arg
             {
             /*%%%*/
-            value_expr($1);
-            value_expr($3);
+            range_expr($1);
+            range_expr($3);
             $$ = NEW_DOT3($1, $3, &@$);
             /*% %*/
             /*% ripper: dot3!($1, $3) %*/
@@ -1931,7 +1933,7 @@ arg       : lhs '=' arg_rhs
                         loc.beg_pos = @2.end_pos;
                         loc.end_pos = @2.end_pos;

-            value_expr($1);
+            range_expr($1);
             $$ = NEW_DOT2($1, new_nil(&loc), &@$);
             /*% %*/
             /*% ripper: dot2!($1, Qnil) %*/
@@ -1943,7 +1945,7 @@ arg       : lhs '=' arg_rhs
                         loc.beg_pos = @2.end_pos;
```

```
                        loc.end_pos = @2.end_pos;

-           value_expr($1);
+           range_expr($1);
            $$ = NEW_DOT3($1, new_nil(&loc), &@$);
            /*% %*/
            /*% ripper: dot3!($1, Qnil) %*/
@@ -9540,6 +9542,18 @@ value_expr_gen(struct parser_params *p, NODE *node)
     return TRUE;
 }

+static int
+range_expr_gen(struct parser_params *p, NODE *node)
+{
+    switch (nd_type(node)) {
+      case NODE_DOT2:
+      case NODE_DOT3:
+        yyerror1(&node->nd_loc, "nested range");
+        return FALSE;
+    }
+    return value_expr_gen(p, node);
+}
+
 static void
 void_expr(struct parser_params *p, NODE *node)
 {
```

matz (Yukihiro Matsumoto) and naruse (Yui NARUSE), can we commit it? Is it okay to reject (1..1)..1 as a parse error?

#3 - 01/30/2019 09:54 PM - kddeisz (Kevin Deisz)

Just want to add to this that

```
Ripper.sexp("1..\n1..5")
```

breaks and returns nil