Hi,

In Ruby, the scope of a for loop variable is not limited in the for expression, which means that a for expression counts on side effects. This sometimes causes unexpected behavior when closing a for loop variable using a closure.

```ruby
$ cat test.rb
procs = []
for lang in ["Ruby", "Scala", "Haskell"]
  procs << -> { p lang }
end
procs.each(&:call)
```

Why not make a for loop variable local to the for expression like a block parameter?

In Ruby 1.8, a for expression is faster than a method call with a block, but it's not true in Ruby 1.9, so there is no reason to give a for expression special treatment.

The compatibility might be a problem, but I believe that code depending on the current behavior is evil. Ruby's for expression also allows a global variable and a method call as a for loop variable. However, I rarely see such code in real-world applications.

Furthermore, I've heard that the scope of a foreach loop variable is changed in C# 5.


Why should Ruby be more conservative than C#?

I've attached a patch for POC.

There is at least one problem in this patch. The problem is that it cannot handle the following code in mkmf.rb.

```ruby
for lib in libs = $libs.split
  ...
end
```

It's because libs is considered local to the for expression in parsing phase, but the variable can't be found in dyna vars in compiling phase. I'm wondering how this code should behave and how to implement it.

I suspected that this code was written by nobu, but it was written by Eric.

```ruby
for lib in libs = $libs.split
  ...
end
```

It's because libs is considered local to the for expression in parsing phase, but the variable can't be found in dyna vars in compiling phase. I'm wondering how this code should behave and how to implement it.
There is at least one problem in this patch. The problem is that it cannot handle the following code in mkmf.rb.

I suspected that this code was written by nobu, but it was written by Eric.

I apologize that it was wrong, Eric.
Eric just fixed indentation of code written by nobu.
I believe that no one else write such code.

--
Shugo Maeda

#2 - 02/24/2012 02:25 PM - shugo (Shugo Maeda)
- File for_loop_scope_fix_2.diff added

Hi,

Shugo Maeda wrote:

There is at least one problem in this patch. The problem is that it cannot handle the following code in mkmf.rb.

I've fixed my patch to fix the problem.
By the attached patch, the scope of variables in expr_value is not limited in the for expression.

I think it's reasonable because

for i in
end

is equivalent to

.each do |i|
end

, where i is outside the block.

#3 - 02/25/2012 03:23 AM - Anonymous

On Fri, Feb 24, 2012 at 02:25:57PM +0900, Shugo Maeda wrote:

Issue #6070 has been updated by Shugo Maeda.

File for_loop_scope_fix_2.diff added

Hi,

Shugo Maeda wrote:

There is at least one problem in this patch. The problem is that it cannot handle the following code in mkmf.rb.

I've fixed my patch to fix the problem.
By the attached patch, the scope of variables in expr_value is not limited in the for expression.

I think it's reasonable because

for i in
end

is equivalent to

.each do |i|
end

, where i is outside the block.
I agree with your opinion regarding the current side effect problems of for. I would love to see your patch in Ruby 2.0!

--
Aaron Patterson
http://tenderlovemaking.com/

#4 - 02/25/2012 11:42 PM - shugo (Shugo Maeda)
- File for_loop_scope_fix_0224.diff added

Hi,

I agree with your opinion regarding the current side effect problems of for. I would love to see your patch in Ruby 2.0!

Thanks for your agreement:)  
I've updated my patch, whose changes are:

- use NODE_MEMO to avoid memory leak when parsing of expr_value failed.
- removed unused code.
- fixed tests for new behavior of for expressions.
- fixed rss/parser.rb for new behavior of for expressions.

A good news is that rss/parser.rb is the only library affected by this change of for expressions.
The patch passes make test, and the result of test-all is 1F0E, which is the same as the version without the patch.

The version with the patch is a bit faster than the original version:

defiant:ruby$ time ruby-trunk -e 'for i in 1..10000000; end'
ruby-trunk -e 'for i in 1..10000000; end'  3.92s user 0.01s system 99% cpu 3.936 total
defiant:ruby$ time build/ruby-trunk -e 'for i in 1..10000000; end'
build/ruby-trunk -e 'for i in 1..10000000; end'  3.52s user 0.02s system 99% cpu 3.545 total

#5 - 03/01/2012 03:46 AM - kernigh (George Koehler)

=begin
This patch (for_loop_scope_fix_0224.diff) introduces a confusing change. With current Ruby, for loops and loops never have local variables. With this patch, for loops have local variables, but while loops never have them. Here is a brief example.

[nil].each do |i|
x = "x is string"
end
puts((x rescue "x not defined"))

for j in [nil]
y = "y is string"
end
puts((y rescue "y not defined"))

while true
z = "z is string"
break
end
puts((z rescue "z not defined"))

With current Ruby, y and z are not local to their loops.

$ ../tool/runruby.rb -v
ruby 2.0.0dev (2012-03-01 trunk 34861) [x86_64-openbsd5.0]
$ ../tool/runruby.rb ~/scratch.rb
x not defined
y is string
z is string

After applying for_loop_scope_fix_0224.diff, y is local to its for loop, but z is not local to its while loop.

$ ../tool/runruby.rb ~/scratch.rb
x not defined
y not defined
z is string

With for_loop_scope_fix_0224.diff, it is difficult to understand why for loops can restrict scope of variables, but while loops never do so.

09/05/2021
Hello,

2012/3/1 George Koehler xkernigh@netscape.net:

This patch (for_loop_scope_fix_0224.diff) introduces a confusing change. With current Ruby, for loops and loops never have local variables. With this patch, for loops have local variables, but while loops never have them. Here is a brief example.

Yes, it is what I have expected.

With for_loop_scope_fix_0224.diff, it is difficult to understand why for loops can restrict scope of variables, but while loops never do so.

Without the patch, the behavior of for loops is consistent with that of while loops; however, it is inconsistent with that of blocks. By contrast with the patch, the behavior of for loops is consistent with that of blocks; however, it is inconsistent with that of while loops. So there's a trade-off between these two design choices.

For loops are syntactically close to while loops, but for loops are semantically more close to blocks, so I think it's better to make for loops consistent with blocks.

Shugo Maeda

#7 - 03/05/2012 04:59 PM - ko1 (Koichi Sasada)

(2012/03/05 16:10), Shugo Maeda wrote:

For loops are syntactically close to while loops, but for loops are semantically more close to blocks, so I think it's better to make for loops consistent with blocks.

I like current behavior because of consistency with "while" statement :)

--
// SASADA Koichi at atdot dot net

#8 - 03/15/2012 05:53 PM - shugo (Shugo Maeda)

- Status changed from Open to Rejected

Koichi Sasada wrote:

(2012/03/05 16:10), Shugo Maeda wrote:

For loops are syntactically close to while loops, but for loops are semantically more close to blocks, so I think it's better to make for loops consistent with blocks.

I like current behavior because of consistency with "while" statement :)

Matz has the same opinion, so I'll close this ticket.

Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>Author</th>
</tr>
</thead>
<tbody>
<tr>
<td>for_loop_scope_fix.diff</td>
<td>2.64 KB</td>
<td>02/24/2012</td>
<td>shugo (Shugo Maeda)</td>
</tr>
<tr>
<td>for_loop_scope_fix_2.diff</td>
<td>2.81 KB</td>
<td>02/24/2012</td>
<td>shugo (Shugo Maeda)</td>
</tr>
<tr>
<td>noname</td>
<td>500 Bytes</td>
<td>02/25/2012</td>
<td>Anonymous</td>
</tr>
<tr>
<td>for_loop_scope_fix_0224.diff</td>
<td>8 KB</td>
<td>02/25/2012</td>
<td>shugo (Shugo Maeda)</td>
</tr>
</tbody>
</table>