Ruby - Feature #11537

Introduce "Safe navigation operator"

09/18/2015 09:29 AM - hsbt (Hiroshi SHIBATA)

Status:	Closed	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

I sometimes write following code with rails application:

```
u = User.find(id)
if u && u.profile && u.profile.thumbnails && u.profiles.thumbnails.large
  ...
```

or

```
# Use ActiveSupport
if u.try!(:profile).try!(:thumbnails).try!(:large)
 ...
```

I hope to write shortly above code. Groovy has above operator named "Safe navigation operator" with "?." syntax.
Ruby can't use "?." operator.

Can we use ".?" syntax. like this:

```
u = User.find(id)
u.?profile.?thumbnails.?large
```

Matz. How do you think about this?

Related issues:

Related to Ruby - Feature #8191: Short-hand syntax for duck-typing	**Closed**	
Related to Ruby - Feature #8237: Logical method chaining via inferred receiver	**Closed**	
Related to Ruby - Feature #11034: Nil Conditional	**Closed**	
Related to Ruby - Feature #1122: request for: Object#try	**Rejected**	**02/07/2009**

Associated revisions

Revision a356fe1c3550892902103f66928426ac8279e072 - 10/22/2015 06:30 AM - nobu (Nobuyoshi Nakada)

Safe navigation operator

- compile.c (iseq_peephole_optimize): peephole optimization for branchnil jumps.
- compile.c (iseq_compile_each): generate save navigation operator code.
- insns.def (branchnil): new opcode to pop the tos and branch if it is nil.
- parse.y (NEW_QCALL, call_op, parser_yylex): parse token '.?'. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52214 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision a356fe1c - 10/22/2015 06:30 AM - nobu (Nobuyoshi Nakada)

Safe navigation operator

- compile.c (iseq_peephole_optimize): peephole optimization for branchnil jumps.
- compile.c (iseq_compile_each): generate save navigation operator code.
- insns.def (branchnil): new opcode to pop the tos and branch if it is nil.
- parse.y (NEW_QCALL, call_op, parser_yylex): parse token '.?'. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52214 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 7e730322ee714b607c94389911f5b86704cc8ed4 - 10/23/2015 01:47 AM - nobu (Nobuyoshi Nakada)

compile.c: just insert

- compile.c (compile_massign_lhs): just insert topn insn instead
  of popping and adding. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52233 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 7e730322 - 10/23/2015 01:47 AM - nobu (Nobuyoshi Nakada)

compile.c: just insert

- compile.c (compile_massign_lhs): just insert topn insn instead
  of popping and adding. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52233 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 0b7d473734d0dec8520afe7a36540aa1f40d2532 - 10/23/2015 01:49 AM - nobu (Nobuyoshi Nakada)

safe navigation attrset

- compile.c (iseq_compile_each): support safe navigation of simple
  attribute assignment. [Feature #11537]
- parse.y (mlhs_node, lhs, attrset_gen): ditto. keep mid
  non-attrset as the sign of safe navigation.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52234 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 0b7d4737 - 10/23/2015 01:49 AM - nobu (Nobuyoshi Nakada)

safe navigation attrset

- compile.c (iseq_compile_each): support safe navigation of simple
  attribute assignment. [Feature #11537]
- parse.y (mlhs_node, lhs, attrset_gen): ditto. keep mid
  non-attrset as the sign of safe navigation.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52234 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision ae8f8fddb0f8e89e86d27a7710a9c07addf7901b - 10/23/2015 02:58 AM - nobu (Nobuyoshi Nakada)

compile.c: optimize method chain

- compile.c (iseq_peephole_optimize): optimize lengthy safe
  navigation method chain. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52237 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision ae8f8fdd - 10/23/2015 02:58 AM - nobu (Nobuyoshi Nakada)

compile.c: optimize method chain

- compile.c (iseq_peephole_optimize): optimize lengthy safe
  navigation method chain. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52237 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 1a98528cb7afc8f8de5af50e5d51c83b251ea860 - 10/25/2015 01:12 AM - nobu (Nobuyoshi Nakada)

symbol.c: dotq in ripper

- symbol.c (op_tbl): add DOTQ for ripper. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52276 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 1a98528c - 10/25/2015 01:12 AM - nobu (Nobuyoshi Nakada)

symbol.c: dotq in ripper

- symbol.c (op_tbl): add DOTQ for ripper. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52276 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 603b000dd034c0ce52e2f8afd0e43c4969e609a4 - 10/26/2015 03:55 AM - nobu (Nobuyoshi Nakada)

parse.y: call_op2

- parse.y (call_op2): separate from call_op and also allow "::",
  while dot_or_colon should not allow ".?". [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52282 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 603b000d - 10/26/2015 03:55 AM - nobu (Nobuyoshi Nakada)

parse.y: call_op2

- parse.y (call_op2): separate from call_op and also allow "::",
  while dot_or_colon should not allow ".?". [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52282 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 506e50b430aef83680e9f7b02c75299c7bbcfa53 - 10/26/2015 08:11 AM - nobu (Nobuyoshi Nakada)

parse.y: fix ripper

- parse.y (call_op, call_op2): fix values on ripper. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52284 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 506e50b4 - 10/26/2015 08:11 AM - nobu (Nobuyoshi Nakada)

parse.y: fix ripper

- parse.y (call_op, call_op2): fix values on ripper. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52284 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 107b0dc8cdd3c49563bd057c631c1d662d6dfd89 - 10/28/2015 04:29 AM - nobu (Nobuyoshi Nakada)

parse.y: fix op_assign type

- parse.y (new_attr_op_assign): fix op_assign type, which is
  already an ID since r52284. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52318 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 107b0dc8 - 10/28/2015 04:29 AM - nobu (Nobuyoshi Nakada)

parse.y: fix op_assign type

- parse.y (new_attr_op_assign): fix op_assign type, which is
  already an ID since r52284. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52318 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision dbf73f6fc2a4b0414110eea64a9f98d33c2cf930 - 11/02/2015 11:07 AM - nobu (Nobuyoshi Nakada)

parse.y: lbracket

- parse.y (lbracket): support .? before aref. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52422 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision dbf73f6f - 11/02/2015 11:07 AM - nobu (Nobuyoshi Nakada)

parse.y: lbracket

- parse.y (lbracket): support .? before aref. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52422 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision a3ee54f8a653f2e7c8ea88ef5ee9f3a49a248447 - 11/03/2015 12:27 AM - nobu (Nobuyoshi Nakada)

parse.y: revert lbracket

- parse.y (lbracket): remove .? before aref. [Feature #11537]
  revert r52422 and r52424

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52430 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision a3ee54f8 - 11/03/2015 12:27 AM - nobu (Nobuyoshi Nakada)

parse.y: revert lbracket

- parse.y (lbracket): remove .? before aref. [Feature #11537]
  revert r52422 and r52424

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52430 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 837babd56459aafc1232a12fbfa783025d619b98 - 11/06/2015 03:39 AM - nobu (Nobuyoshi Nakada)

change DOTQ

- defs/id.def (token_ops), parse.y (parser_yylex): change DOTQ
  from ".?" to "&.". [ruby-core:71363] [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52462 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 837babd5 - 11/06/2015 03:39 AM - nobu (Nobuyoshi Nakada)

change DOTQ

- defs/id.def (token_ops), parse.y (parser_yylex): change DOTQ
  from ".?" to "&.". [ruby-core:71363] [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52462 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 613737eee9168db483206bcac65c83413be02f42 - 12/05/2015 07:58 AM - nobu (Nobuyoshi Nakada)

node.c: NODE_QCALL

- node.c (dump_node): dump NODE_QCALL. [Feature #11537]
  http://twitter.com/watson1978/status/673042429931446272

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52895 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 613737ee - 12/05/2015 07:58 AM - nobu (Nobuyoshi Nakada)

node.c: NODE_QCALL

- node.c (dump_node): dump NODE_QCALL. [Feature #11537]
  http://twitter.com/watson1978/status/673042429931446272

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@52895 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision bfd34365b6b01898649cfd570306e062c170628b - 12/16/2015 01:49 AM - nobu (Nobuyoshi Nakada)

parse.y: fix block_call&.call

- parse.y (block_command, block_call): fix &. calls after
  block_call. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@53138 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision bfd34365 - 12/16/2015 01:49 AM - nobu (Nobuyoshi Nakada)

parse.y: fix block_call&.call

- parse.y (block_command, block_call): fix &. calls after
  block_call. [Feature #11537]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@53138 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 09/18/2015 09:52 AM - akr (Akira Tanaka)

- Related to Feature #8191: Short-hand syntax for duck-typing added

#2 - 09/18/2015 09:52 AM - akr (Akira Tanaka)

- Related to Feature #8237: Logical method chaining via inferred receiver added

#3 - 09/18/2015 09:58 AM - akr (Akira Tanaka)

- Related to Feature #11034: Nil Conditional added

#4 - 09/18/2015 02:34 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

+1. Besides Groovy, CoffeeScript also allows that and I've used this feature a lot in my past days with Groovy and still use it all the time with CoffeeScript and have been missing this feature in Ruby for a long time.

#5 - 09/19/2015 12:42 AM - nobu (Nobuyoshi Nakada)

- Description updated

#6 - 09/19/2015 04:09 AM - phluid61 (Matthew Kerwin)

As per the discussion a few years ago in #8191, I'm +1 for .?foo syntax.

I still have a lingering question about the following; does this print "Hello" twice?

```
a = Object.new
def a.b
  puts "Hello"
end
a.?b.?c

a && a.b && a.b.c    # this does
a.try!(:b).try!(:c)  # this doesn't (?)
```

I think it should NOT call a.b twice.

#7 - 09/19/2015 05:10 AM - akr (Akira Tanaka)

- Related to Feature #1122: request for: Object#try added

#8 - 09/23/2015 05:04 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Matthew, it shouldn't be implemented as a && a.b && a.b.c, but something like (temp1 = a) && (temp2 = temp1.b) && temp2.c

#9 - 10/07/2015 02:30 PM - matz (Yukihiro Matsumoto)

I like the idea. My remaining concern is ".?" is too similar to "?." which is chosen by other languages.
We cannot use "?." since it conflicts with a method call with a "*?" predicate method.

Matz.

#10 - 10/07/2015 04:51 PM - sawa (Tsuyoshi Sawada)

The && and try are different. I am considering the && version.

Since we already have:

```
a &&= b
```

which means

```
a = a && b
```

By analogy from the above, and given that we want

```
a && a.b
```

what about:

```
a.&&b
```

or more shortly:

```
a.&b
```

#11 - 10/21/2015 07:45 AM - matz (Yukihiro Matsumoto)

In several languages (Groovy Swift etc.), use ?. but we cannot use it in Ruby, because foo? is a valid method name.
Thus .? is a reasonable alternative for Ruby, I think.

Accepted.

Matz.

#12 - 10/21/2015 08:05 AM - matz (Yukihiro Matsumoto)

Oh, I made mistake. We will introduce .? (typo fixed already).

#13 - 10/21/2015 10:20 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

Great news! Thanks! Is this going to be released on next minor or on Ruby 3 only?

#14 - 10/21/2015 07:29 PM - marcandre (Marc-Andre Lafortune)

Great!

Just to be clear, this will check only for nil, right?

```
nil.?foo # => nil
false.?foo # => NoMethodError
```

#15 - 10/21/2015 11:33 PM - nobu (Nobuyoshi Nakada)

Marc-Andre Lafortune wrote:

> Just to be clear, this will check only for nil, right?

I think so, and my implementation too.

#16 - 10/22/2015 06:30 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset r52214.

Safe navigation operator

- compile.c (iseq_peephole_optimize): peephole optimization for branchnil jumps.
- compile.c (iseq_compile_each): generate save navigation operator code.
- insns.def (branchnil): new opcode to pop the tos and branch if it is nil.
- parse.y (NEW_QCALL, call_op, parser_yylex): parse token '.?'. [Feature #11537]

#17 - 10/22/2015 04:29 PM - treznick (Tom Reznick)

Hi,

I think we may have found some unexpected behavior with the .? operator.

If I call the following:

```
s = Struct.new(:x)
o = s.new()
o.x #=> nil
o.x.nil? #=> true
o.x.?nil? #=> nil
o.x.kind_of?(NilClass) #=> true
o.x.?kind_of?(NilClass) #=> nil
o.x.methods.include?(:nil?) #=> true
```

While it's arguably a bit peculiar to try to check that nil is nil, in a nil-safe way, .?kind_of?(NilClass) could reasonably return true.

Also this is clearly a bit of a contrived example, it does highlight some of the more unexpected behaviors of the .? operator.

Any chance of clarification?

All best!

Tom Reznick
Software Engineer
Continuity
@threznick

Nobuyoshi Nakada wrote:

> Applied in changeset r52214.

Safe navigation operator

- compile.c (iseq_peephole_optimize): peephole optimization for branchnil jumps.
- compile.c (iseq_compile_each): generate save navigation operator code.
- insns.def (branchnil): new opcode to pop the tos and branch if it is nil.
- parse.y (NEW_QCALL, call_op, parser_yylex): parse token '.?'. [Feature #11537]

#18 - 10/22/2015 04:34 PM - asterite (Ary Borenszweig)

I know this is already decided and the commit is out there, but since you are adding new syntax and a new feature to the language, I suggest you reconsider https://bugs.ruby-lang.org/issues/9076

With that change, instead of adding special syntax for safe nil traversing, you get generalized syntax for the implicit block argument. Instead of this:

```
obj.?bar(x, y)
```

You do:

```
obj.try &.bar(x, y)
```

The try method is simply:

```
class Object
  # But obviously implemented in C for performance reasons
  def try
    yield self unless self.is_a?(NilClass)
  end
end
```

As I mention in the original issue, foo &.bar simply gets translated *by the parser* to something like foo { |x| x.bar } (where x doesn't conflict with any other identifier in the method). So it's just a change in the parser, no need to change compile.c, insns.def, etc (although I can understand that nil checking might be optimized with a VM instruction).

My main worry is code like this:

```
obj.?empty?
```

That looks confusing to the eye. I associate "?" next to method names to "query" methods, and now when reading an expression "?" can have several meanings (in addition to the ternary operator).

We already have this syntax in Crystal and it's working really well.

#19 - 10/22/2015 04:45 PM - jeremyevans0 (Jeremy Evans)

Tom Reznick wrote:

> Hi,
>
> I think we may have found some unexpected behavior with the .? operator.
>
> If I call the following:
>
> ```
> s = Struct.new(:x)
> o = s.new()
> o.x #=> nil
> o.x.nil? #=> true
> o.x.?nil? #=> nil
> o.x.kind_of?(NilClass) #=> true
> o.x.?kind_of?(NilClass) #=> nil
> o.x.methods.include?(:nil?) #=> true
> ```
>
> While it's arguably a bit peculiar to try to check that nil is nil, in a nil-safe way, .?kind_of?(NilClass) could reasonably return true.

I think it's completely expected that nil.?kind_of?(NilClass) returns nil and not true. The whole point of .? is to return nil without calling the method if the receiver is nil. I'm not sure if .? is a good idea syntax-wise, but if you are going to have it, it shouldn't have special cases for specific methods.

#20 - 10/22/2015 09:28 PM - phluid61 (Matthew Kerwin)

On 23/10/2015 2:46 AM, merch-redmine@jeremyevans.net wrote:

> Issue #11537 has been updated by Jeremy Evans.
>
> Tom Reznick wrote:
>
> > Hi,
> >
> > I think we may have found some unexpected behavior with the .?
> > operator.
> >
> > If I call the following:
> >
> > ```
> > s = Struct.new(:x)
> > o = s.new()
> > o.x #=> nil
> > o.x.nil? #=> true
> > o.x.?nil? #=> nil
> > o.x.kind_of?(NilClass) #=> true
> > o.x.?kind_of?(NilClass) #=> nil
> > o.x.methods.include?(:nil?) #=> true
> > ```
> >
> > While it's arguably a bit peculiar to try to check that nil is nil,
> > in a nil-safe way, .?kind_of?(NilClass) could reasonably return true.
>
> I think it's completely expected that nil.?kind_of?(NilClass) returns
> nil and not true. The whole point of .? is to return nil without
> calling the method if the receiver is nil. I'm not sure if .? is a
> good idea syntax-wise, but if you are going to have it, it shouldn't have
> special cases for specific methods.

I agree, or put another way: if you're testing for nil in two ways, .? has higher priority. That makes it a programmer issue, not a ruby one.

#21 - 10/23/2015 03:58 AM - treznick (Tom Reznick)

Thanks for the thoughtful replies guys! That definitely helps clarify the .? operator
Matthew Kerwin wrote:

> On 23/10/2015 2:46 AM, merch-redmine@jeremyevans.net wrote:
>
> > Issue #11537 has been updated by Jeremy Evans.
> >
> > Tom Reznick wrote:
> >
> > > Hi,
> > >
> > > I think we may have found some unexpected behavior with the .?
> > > operator.
> > >
> > > If I call the following:
> > >
> > > ```
> > > s = Struct.new(:x)
> > > o = s.new()
> > > o.x #=> nil
> > > o.x.nil? #=> true
> > > o.x.?nil? #=> nil
> > > o.x.kind_of?(NilClass) #=> true
> > > o.x.?kind_of?(NilClass) #=> nil
> > > o.x.methods.include?(:nil?) #=> true
> > > ```
> > >
> > > While it's arguably a bit peculiar to try to check that nil is nil,
> > > in a nil-safe way, .?kind_of?(NilClass) could reasonably return true.
> >
> > I think it's completely expected that nil.?kind_of?(NilClass) returns
> > nil and not true. The whole point of .? is to return nil without
> > calling the method if the receiver is nil. I'm not sure if .? is a
> > good idea syntax-wise, but if you are going to have it, it shouldn't have
> > special cases for specific methods.

> I agree, or put another way: if you're testing for nil in two ways, .? has
> higher priority. That makes it a programmer issue, not a ruby one.

#22 - 10/26/2015 05:10 AM - trans (Thomas Sawyer)

Yukihiro Matsumoto wrote:

> I like the idea. My remaining concern is ".?" is too similar to "?." which is chosen by other languages.
> We cannot use "?." since it conflicts with a method call with a "*?" predicate method.

Maybe there is a better syntax by requiring a space:

```
u ? .profile ? .thumbnails ? .large
```

In this way is more an extension of the ternary operator -- the initial dot on the method signals the difference. This should also allow:

```
u ? .profile ? .thumbnails ? .large : default
```

#23 - 10/26/2015 12:25 PM - wycats (Yehuda Katz)

Yukihiro Matsumoto wrote:

> I like the idea. My remaining concern is ".?" is too similar to "?." which is chosen by other languages.
> We cannot use "?." since it conflicts with a method call with a "*?" predicate method.
>
> Matz.

I agree with this concern. I also agree with the idea of trying to make it more like a traditional infix operator by encouraging spaces.

What about:

```
u && .profile && .thumbnails && .large
```

As shorthand for:

```
u && u.profile && u.profile.thumbnails && u.profile.thumbnails.large
```

It reads nicely to me and its meaning is clear. It also doesn't conflict with Ruby's method predicates (? suffix).

Matz, what do you think?

#24 - 10/26/2015 01:28 PM - nobu (Nobuyoshi Nakada)

What does sole .profile do?
If it requires && just before it, a space should not be between them.

#25 - 10/26/2015 01:58 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Yehuda Katz wrote:

> As shorthand for:
>
> ```
> u && u.profile && u.profile.thumbnails && u.profile.thumbnails.large
> ```

Actually, it would be a shorthand for something like:

```
u && (tmp1 = u.profile) && (tmp2 = tmp1.thumbnails) && tmp2.large
```

But I still prefer the syntax implemented in trunk as it's similar to other languages and will be more familiar for those used to the operator in Groovy and CoffeeScript, for example.

#26 - 10/27/2015 09:13 AM - DerKobe (Philip Claren)

Thomas Sawyer wrote:

> In this way is more an extension of the ternary operator -- the initial dot on the method signals the difference. This should also allow:
>
> ```
> u ? .profile ? .thumbnails ? .large : default
> ```

Although I agree that .? is not that intuitive to read, the extended ternary operator would have a problem: boolean false passes for the safe navigator (because it's a valid value) but not for the ternary operator.

#27 - 10/27/2015 11:47 AM - rosenfeld (Rodrigo Rosenfeld Rosas)

On the other hand, the problem with using "u?.profile" is that "u?" is a valid method, but if Ruby required an space for using the new operator, we could get it working very similarly to Groovy and CoffeeScript:

```
u ?.profile ?.thumbnails ?.large || default
```

#28 - 10/27/2015 12:01 PM - mame (Yusuke Endoh)

Rodrigo Rosenfeld Rosas wrote:

> ```
> u ?.profile ?.thumbnails ?.large || default
> ```

Consider a method +.

str.?+"foo" causes no conflict, but str ?.+"foo" will be parsed as str(?. + "foo").

--
Yusuke Endoh mame@ruby-lang.org

#29 - 10/27/2015 12:07 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

I believe you are referring to implementation details and in this case I can't really opinionate, but with regards to syntax, since "?." is not a valid code, it should be interpreted as "?.+" and "foo" would be parsed as its argument.

But if this is too complicate to implement, I'm also fine with using ".?".

#30 - 10/27/2015 12:24 PM - mame (Yusuke Endoh)

"?." is a valid code. Try: p ?. + "foo"

--
Yusuke Endoh mame@ruby-lang.org

#31 - 10/27/2015 12:31 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Thanks for pointing me that. Everyday learning a new hidden feature from Ruby ;)

#32 - 10/27/2015 12:45 PM - mame (Yusuke Endoh)

Rodrigo Rosenfeld Rosas wrote:

> Everyday learning a new hidden feature from Ruby ;)

Me too. Ruby has infinite possibilities.

--
Yusuke Endoh mame@ruby-lang.org

#33 - 10/27/2015 02:31 PM - matz (Yukihiro Matsumoto)

I don't think u && .profile && .thumbnails && .large is acceptable for some reasons:

- it's longer
- unclear

than proposed .?. Instead, &. or .& can be alternatives. But my concern for the idea is && does handle false values (false and nil), where proposed behavior only accepts nil.

Matz.

#34 - 10/29/2015 07:32 PM - Anonymous

The same discussion happens to be on TypeScript and ES6 worlds.
Using .. instead of ?. or .? because it's way more clear when you are using the ternary ? : operator on the same line.

If it's not a conflict in Ruby syntax perhaps worths looking at

https://esdiscuss.org/topic/existential-operator-null-propagation-operator#content-65

https://github.com/Microsoft/TypeScript/issues/16#issuecomment-152275052

Posting the markdown info from there:

This would be amazing operator!! Especially for ES6/ES7/TypeScript - and why not Ruby ?

```
var error = a.b.c.d; //this would fail with error if a, b or c are null or undefined.
var current = a && a.b && a.b.c && a.b.c.d; // the current messy way to handle this
var currentBrackets = a && a['b'] && a['b']['c'] && a['b']['c']['d']; //the current messy way to handle this
var typeScript = a?.b?.c?.d; // The typescript way of handling the above mess with no errors
var typeScriptBrackets = a?['b']?['c']?['d']; //The typescript of handling the above mess with no errors
```

However I propose a more clear one - as not to confuse ? from the a ? b : c statements with a?.b statements:

```
var doubleDots = a..b..c..d;
//this would be ideal to understand that you assume that if any of a, b, c is null or undefined the result wil
l be null or undefined.
var doubleDotsWithBrackets = a..['b']..['c']..['d'];
```

For the bracket notation, I recommend two dots instead of a single one as it's consistent with the others when non brackets are used. Hence only the property name is static or dynamic via brackets.

Two dots, means if its null or undefined stop processing further and assume the result of expression is null or undefined. (as d would be null or undefined).

Two dots make it more clear, more visible and more space-wise so you understand what's going on.

This is not messing with numbers too - as is not the same case e.g.

```
1..toString(); // works returning '1'
var x = {};
x.1 = {y: 'test' }; //fails currently
x[1] = {y: 'test' }; //works currently
var current = x[1].y; //works
var missing= x[2].y; //throws exception
var assume= x && x[2] && x[2].y; // works but very messy
```

About numbers two options: Your call which one can be adopted, but I recommend first one for compatibility with existing rules!

1. Should fail as it does now (x.1.y == runtime error)

```
var err = x..1..y;
// should fail as well, since 1 is not a good property name, nor a number to call a method, since it's after x
 object.
```

1. Should work since it understands that is not a number calling a property from Number.prototype

```
var err = x..1..y; // should work as well, resulting 'test' in this case
var err = x..2..y; // should work as well, resulting undefined in this case
```

With dynamic names:

```
var correct1 = x..[1]..y; //would work returning 'test'
var correct2 = x..[2]..y; //would work returning undefined;
```

What do you think folks?

P.S. foo?.bar and foo?['bar'] syntax would work too.

However the using both current ? : operator and ?. might be very confusing on the same line.

e.g. using ?. and ?['prop']

```
var a = { x: { y: 1 } };
var b = condition ? a?.x.?y : a?.y?.z;
var c = condition ? a?['x']?['y'] : a?['y']?['z'];
```

as opposed to double dots .. and ..['prop']

```
var a = { x: { y: 1 } };
var b = condition ? a..x..y : a..y..z;
var c = condition ? a..['x']..['y'] : a..['y']..['z'];
```

Which one does look more clear to you?

#35 - 10/29/2015 09:08 PM - phluid61 (Matthew Kerwin)

a..b already means something in ruby

I'm happy with Matz's acceptance of .? especially in light of all the
discussions that lead up to it.

#36 - 11/04/2015 12:01 AM - trans (Thomas Sawyer)

Laurentiu Macovei wrote:

> The same discussion happens to be on TypeScript and ES6 worlds.
> Using .. instead of ?. or .? because it's way more clear when you are using the ternary ? : operator on the same line.
>
> If it's not a conflict in Ruby syntax perhaps worths looking at

.. wouldn't work. But that reminds me. Was ! ever considered?

```
u!profile!thumbnails!large
```

#37 - 11/04/2015 12:07 AM - trans (Thomas Sawyer)

Philip Claren wrote:

> Thomas Sawyer wrote:
>
> > In this way is more an extension of the ternary operator -- the initial dot on the method signals the difference. This should also allow:
> >
> > ```
> > u ? .profile ? .thumbnails ? .large : default
> > ```
>
> Although I agree that .? is not that intuitive to read, the extended ternary operator would have a problem: boolean false passes for the safe navigator (because it's a valid value) but not for the ternary operator.

Is allowing false necessary/useful? On the other hand, if it is, then might a nil-only ternary operator be useful too (regardless of this issue)?

#38 - 11/04/2015 12:32 AM - nobu (Nobuyoshi Nakada)

Thomas Sawyer wrote:

> .. wouldn't work. But that reminds me. Was ! ever considered?
>
> `u!profile!thumbnails!large`

No, it's a unary operator.

#39 - 11/04/2015 12:44 AM - matz (Yukihiro Matsumoto)

Thomas, that reminds me of old UUCP addresses (grin).
But ,as Nobu pointed out, it can be parsed as u(!profile(!thumbnails(!large))) unfortunately.

Matz.

#40 - 11/05/2015 11:01 PM - enebo (Thomas Enebo)

How about "? We can pay homage to Windows file delimeter?

a\b\c

I just scanned lexer and I cannot think of a reason off the top of my head why not...what does \ mean? who knows...what for .? mean? I don't know that either. This suggestion is twice as efficient though since it only uses one character.

-Tom

#41 - 11/05/2015 11:48 PM - normalperson (Eric Wong)

tom.enebo@gmail.com wrote:

> How about "? We can pay homage to Windows file delimeter?
>
> a\b\c
>
> I just scanned lexer and I cannot think of a reason off the top of my head why not...what does \ mean? who knows...what for .? mean? I don't

> know that either. This suggestion is twice as efficient though since it only uses one character.

It's already used for continuing long lines.

#42 - 11/06/2015 01:25 AM - matz (Yukihiro Matsumoto)

I think about this for a while, and thinking of introducing &. instead of .?, because:

- .? is similar to ?. in Swift and other languages, but is different anyway.
- Since ? is a valid suffix of method names in Ruby, we already see a lot of question marks in our programs.
- u&.profile reminds us as short form of u && u.profile.

But behavior of &. should be kept, i.e. it should skip nil but recognize false.

Matz.

#43 - 11/06/2015 01:50 AM - enebo (Thomas Enebo)

Eric Wong wrote:

> tom.enebo@gmail.com wrote:
>
> > How about "? We can pay homage to Windows file delimeter?
> >
> > a\b\c
> >
> > I just scanned lexer and I cannot think of a reason off the top of my head why not...what does \ mean? who knows...what for .? mean? I don't know that either. This suggestion is twice as efficient though since it only uses one character.
>
> It's already used for continuing long lines.

well it is but that is not really an issue since it only will acknowledge " if right before end of line (lexer just says spaceSeen and loops back up to top of lexer for its next token). What is really weird to me is if it isn't at the end of the line it returns as the token '\'. I see nowhere in the grammar where ruby acknowledges this as a valid token. Any other literal escaping happens within lexing. Did I just find a vestigial organ or am I missing something simple?

In any case it looks like Matz might have picked something other than .?, which was main reason I came up with \ as an idea (I feel reversing order from other languages will confuse them more than it will help -- so pick something different altogether).

-Tom

#44 - 11/06/2015 02:21 AM - nobu (Nobuyoshi Nakada)

Binary operators implicitly continue the next line, without a backslash.

e.g.,

```
foo +
1
```

is same as foo + 1.

If \ were become a binary operator, there is an ambiguity.

```
foo \
1
```

is foo(1) (current interpretation) or foo\1?

#45 - 11/06/2015 04:16 AM - shugo (Shugo Maeda)

- Status changed from Closed to Open

Yukihiro Matsumoto wrote:

> I think about this for a while, and thinking of introducing &. instead of .?, because:

It sounds nice because &. is more friendly to syntax highlighting of my favorite editor,
but le me play devil's advocate here.

- .? is similar to ?. in Swift and other languages, but is different anyway.

This also applies to ->, but -> is accepted now, so it's not a strong reason.

- Since ? is a valid suffix of method names in Ruby, we already see a lot of question marks in our programs.

& also has other roles such as a binary operator and the prefix for block arguments.
The current syntax allows x.&y as a valid expression, whose behavior is the same as x & y,
so it may be confusing.

#46 - 11/06/2015 06:06 PM - rosenfeld (Rodrigo Rosenfeld Rosas)

Yukihiro Matsumoto wrote:

> But behavior of &. should be kept, i.e. it should skip nil but recognize false.

I'm not sure I understood exactly what you meant by this.

Did you mean .? and &. would be implemented in the same way?

For example: false.?inexistent will raise. Should false&.inexisting raise or return false?

I'd like to see it always raising no matter you decide for ".?" or "&.". I don't see any reasons why one would like to call a method conditionally in the false object. If this happens it's most likely to be a bug from the method returning false.

#47 - 11/07/2015 12:49 AM - nobu (Nobuyoshi Nakada)

Rodrigo Rosenfeld Rosas wrote:

> Did you mean .? and &. would be implemented in the same way?

Yes.

> For example: false.?inexistent will raise. Should false&.inexisting raise or return false?

It will raise a NoMethodError.

#48 - 11/07/2015 02:32 AM - ko1 (Koichi Sasada)

I have weak objection because foo&.bar seems check nil *and* false.

Maybe this is because I read this expression as "foo and foo.bar", which expression checks also false.

#49 - 11/09/2015 02:39 PM - enebo (Thomas Enebo)

Nobuyoshi Nakada wrote:

> Binary operators implicitly continue the next line, without a backslash.
>
> e.g.,
>
> ```
> foo +
> 1
> ```
>
> is same as foo + 1.
>
> If \ were become a binary operator, there is an ambiguity.
>
> ```
> foo \
> 1
> ```
>
> is foo(1) (current interpretation) or foo\1?

I am willing to withdraw this proposal and it is potentially ambiguous (although still able to be properly parsed) but I can count the number of times I have seen a rubyist use a line continuation with zero fingers :) So I do not think they confusion would be very large.

-Tom

#50 - 11/13/2015 10:46 AM - uwe@kubosch.no (Uwe Kubosch)

Yukihiro Matsumoto wrote:

I think about this for a while, and thinking of introducing &. instead of .?, because:

- .? is similar to ?. in Swift and other languages, but is different anyway.
- Since ? is a valid suffix of method names in Ruby, we already see a lot of question marks in our programs.
- u&.profile reminds us as short form of u && u.profile.

But behavior of &. should be kept, i.e. it should skip nil but recognize false.

I must say I am very happy with this change. "&." is much easier to read than ".?" .

#51 - 12/05/2015 07:58 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset r52895.

---

node.c: NODE_QCALL

- node.c (dump_node): dump NODE_QCALL. [Feature #11537]
  http://twitter.com/watson1978/status/673042429931446272