Ruby master - Feature #4288
Allow invoking arbitrary method names with foo."something" syntax

01/18/2011 02:50 PM - headius (Charles Nutter)

| Status:    | Closed    |
| Priority:  | Normal    |
| Assignee:  | matz (Yukihiro Matsumoto) |
| Target version: | 3.0 |

Description
=begin
This is one Groovy feature I tend to like.

For non-standard or arbitrary method names, it would be nice to have a way to invoke them directly that doesn't require the parser to be made more complex nor require an intermediate "send" call. Groovy does this by allowing the following form:

foo."asdf"()

This syntax would make it easier to integrate with other languages that have different naming rules. For example, -|=@++ is a valid operator in Scala. With this syntax, you could invoke it as

foo."-|=@++" bar

The alternative in JRuby is that we have to map such names as eq_bang_at_plus_eq, which is certainly not as elegant, or force people to use send (and force them to use :"" anyway, since :=|=++ is not a valid symbol).

It's left up for debate whether string interpolation should be allowed in this syntax.
=end

Related issues:
Related to Ruby master - Feature #5394: Anonymous Symbols, Anonymous Methods
Rejected 10/04/2011

History
#1 - 01/19/2011 03:45 AM - naruse (Yui NARUSE)
- Status changed from Open to Assigned
- Assignee set to matz (Yukihiro Matsumoto)

=begin

=end

#2 - 01/19/2011 11:13 PM - austin (Austin Ziegler)
=begin
On Tue, Jan 18, 2011 at 12:51 AM, Charles Nutter redmine@ruby-lang.org wrote:

This is one Groovy feature I tend to like.

[...]

This syntax would make it easier to integrate with other languages that have different naming rules. For example, -|=@++ is a valid operator in Scala. With this syntax, you could invoke it as

foo."-|=@++" bar

How does one define this sort of named method, or is it something that we declare must be handled in method_missing (or an equivalent for JRuby/Scala, for example) if it's not an otherwise legal Ruby method name?

-austin
On Wed, Jan 19, 2011 at 12:51 AM, Charles Nutter redmine@ruby-lang.org wrote:

This syntax would make it easier to integrate with other languages that have different naming rules. For example, `=@+=` is a valid operator in Scala. With this syntax, you could invoke it as

```ruby
foo.="@+=" bar
```

How does one define this sort of named method, or is it something that we declare must be handled in method_missing (or an equivalent for JRuby/Scala, for example) if it's not an otherwise legal Ruby method name?

```ruby
Class.new {  
define_method ':=!@+=' do  
  puts 'I have no idea what this method is supposed to do, though'
  end
}.new.send ':=!@+='
```

And, as demonstrated, isn't #send good enough?

On Tue, Jan 18, 2011 at 12:51 AM, Charles Nutter redmine@ruby-lang.org wrote:

For non-standard or arbitrary method names, it would be nice to have a way to invoke them directly that doesn't require the parser to be made more complex nor require an intermediate "send" call. Groovy does this by allowing the following form:

```ruby
foo."asdf"()
foo.#{some_variable}()
```

I've always found the use of Kernel#send for dynamic method dispatch to be a bit awkward and would prefer something that was syntactically similar to static method dispatch rather than the current dependency on the 'magic' behavior of Kernel#send.

Could you explain how foo.#{some_variable}() is less magic than foo.send(some_variable)? I am currently of the opposite opinion.

On Jan 18, 2011, at 12:51 AM, Charles Nutter wrote:

Maybe I'm crazy, but could I suggest something a little more general?

```ruby
foo.#{expression}(arg1)
```

to have the same semantics as

```ruby
foo.send(expression, arg1)
```
I don't want something that just resolves to doing "send" under the covers. I want this code:

foo."something"()

to parse identically to:

foo.something()

What I want is purely a syntax-level quoting mechanism for method invocation.

I've always found the use of Kernel#send for dynamic method dispatch to be a bit awkward and would prefer something that was syntactically similar to static method dispatch rather than the current dependency on the 'magic' behavior of Kernel#send.

I agree, but "send" also changes things in subtle ways. I just want a way to specify a syntactically "unfriendly" method call and have it actually resolve to a plain old CALL, FCALL, or VCALL (in MRI AST parlance).

FWIW, there have been proposals to add this sort of "symbolic freedom" to Java method invocations using a similar form:

foo.#"something"

- Charlie

Perhaps a syntactic approach to dynamic dispatch would enable more interesting optimizations since it would be clear that something 'interesting' was going on rather than just a generic method call to a magic method like Kernel#send.

If you mean that the above syntax would actually parse to those literal call names (rather than bouncing through send) then I have no objections. I feel like foo."something" is cleaner, though, and the {} immediately says "block" to me. If that was your intent...I think that's going a bit too far :)

- Charlie

---

#6 - 01/20/2011 08:13 AM - headius (Charles Nutter)

=begin

On Wed, Jan 19, 2011 at 8:53 AM, Nikolai Weibull now@bitwi.se wrote:

And, as demonstrated, isn't #send good enough?

- It's longer :)
- It defeats optimization, since you have to bounce through a "send" call to get to the eventual call. In essence, send typically forces a slow-path dynamic dispatch all the time. foo."something" would be a literal dispatch to the target method.
- The default form of send routes around visibility. That's not always what you want.
- On 1.8, send deepens the exception backtrace.

The syntax proposed is basically a quoting mechanism for literal method calls. It doesn't replace send, it just expands what you can do with direct call syntax. send changes the dispatch logic in subtle ways.

- Charlie

=end

#7 - 01/21/2011 06:39 AM - headius (Charles Nutter)

=begin

On Wed, Jan 19, 2011 at 6:03 PM, Gary Wright gwtmp01@mac.com wrote:

I probably wasn't clear. I also don't want syntactic sugar for #send but instead a way to dynamically name a method without the extra step
of looking up the implementation of #send.

Ok, cool.

I was trying to expand on your idea a bit to produce a more
general approach that still addressed your use case:

```ruby
foo.('something')()
```
can still be optimized to the same parse tree as

```ruby
foo.something()
```

but it also admits the use of an arbitrary expression to generate
the name of the method to call.

It does, but my concern with it is that (in the same way as
string-interpolated ."" syntax) it can't be made 100% parse-time. So I
think the questions to be answered are (in this order):

1. Do we feel it's useful to have a quoting mechanism for method
   invocation (that does not use send)?
2. Do we want that mechanism to allow runtime determination of the
   method to be called (and how does this water down the value of a
   pure-syntactical quoting mechanism)?
3. Do we want the runtime determination to allow arbitrary code

Given that an interpolated mechanism could basically introduce any
code, your syntax and string-interpolated syntax are largely the same
feature (i.e. they both go all the way to (3) above.

I definitely want (1). I'm far more dubious on the value of (2) and
(3) weighed against the benefit...

BTW, I suspect that any of these ideas (single quotes, double quotes, or braces) is going to complicate the parser...

Perhaps not as much as you might expect; a " is not a valid token
after a ".", so there's no ambiguity.

- Charlie

#8 - 01/21/2011 05:50 PM - sdsykes (Stephen Sykes)

It does, but my concern with it is that (in the same way as
string-interpolated ."" syntax) it can't be made 100% parse-time. So I
think the questions to be answered are (in this order):

1. Do we feel it's useful to have a quoting mechanism for method
   invocation (that does not use send)?
2. Do we want that mechanism to allow runtime determination of the
   method to be called (and how does this water down the value of a
   pure-syntactical quoting mechanism)?
3. Do we want the runtime determination to allow arbitrary code

Given that an interpolated mechanism could basically introduce any
code, your syntax and string-interpolated syntax are largely the same
feature (i.e. they both go all the way to (3) above.

I definitely want (1). I'm far more dubious on the value of (2) and
(3) weighed against the benefit...

Yes, agree, if you want interpolation you can use send.

But using double quotes leads to an expectation of interpolation does it not?
So would not single quotes be a better proposal in that case?

-Stephen
On Fri, Jan 21, 2011 at 2:50 AM, Stephen Sykes sdsykes@gmail.com wrote:

Yes, agree, if you want interpolation you can use send.

But using double quotes leads to an expectation of interpolation does it not? So would not single quotes be a better proposal in that case?

Yes, you are correct :)

foo.'something'()

Charlie

Actually, I think this is easy discoverable by the interpreter. For instance, the interpreter can convert foo.'something' or foo."something" or foo."#{'some'}thing" to foo.something, but foo."#{something}" would be converted to "foo.send :something".

So, I would vote for the interpolation to be allowed... Additionally, I think that differently from "send", foo."#{something}" should not be allowed if "something" is a private method.

Rodrigo.

I agree, you'd want to support double-quotes and string interpolation, otherwise the introduction of the new syntax loses half its benefit. I don't see how interpolation could increase parser complexity, given that you could use the exact same string interpolation logic that already exists, and just map the result to as method invocation. The proposed syntax should be a direct replacement/alternative to #send, like how string interpolation, e.g. "The time is #{Time.now}" is an alternative to "The time is + Time.now.to_s.

I thought I have closed this issue long ago. Use send.

Matz.