Ruby trunk - Feature #6841

Shorthand for Assigning Return Value of Method to Self

08/07/2012 11:37 AM - wardrop (Tom Wardrop)

Status:	Assigned	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:	Next Major	

Description

=begin
Quite often in Ruby, I find myself doing something like: ((({my_var[:foo][:bar] = my_var[:foo][:bar].to_i}))) or ((({obj.foo.bar = obj.foo.bar.to_i}))). Realising this, I thought of what would be a fairly nice shorthand syntax for this, which could be: ((({my_var[:foo][:bar] .= to_i}))). How this works should be pretty self-explanatory. The ((({.=}))) operator works exactly like any other assignment operator of this nature.

Would be nice to see this in Ruby 2.0. Wondering what others think of this?
=end

History

#1 - 08/07/2012 11:39 AM - wardrop (Tom Wardrop)

=begin
The formatting obviously didn't work. Try this one:

Quite often in Ruby, I find myself doing something like: ((({my_var[:foo][:bar] = my_var[:foo][:bar].to_i}))) or ((({obj.foo.bar = obj.foo.bar.to_i}))). Realising this, I thought of what would be a fairly nice shorthand syntax for this, which could be: ((({my_var[:foo][:bar] .= to_i}))). How this works should be pretty self-explanatory. The ((({.=}))) operator works exactly like any other assignment operator of this nature.

Would be nice to see this in Ruby 2.0. Wondering what others think of this?
=end

#2 - 08/07/2012 12:37 PM - nobu (Nobuyoshi Nakada)

- Description updated

#3 - 08/07/2012 04:05 PM - trans (Thomas Sawyer)

=begin
Does . effectively become a "call dispatch operator" then? Could one write:

a.send('.', :foo)

I like the idea. But to complete the comparison to other operators, it made me think . would be some sort of method.
=end

#4 - 08/07/2012 08:29 PM - wardrop (Tom Wardrop)

Yeah, I thought about making "." a method, but I don't think that would be advantageous. If you were to overload ".", perhaps to intercept all method calls (can't think of any other reasons besides as hook method), it wouldn't really work as there are other means to call a method, such as #send and **#send**, unless they use "." internally? I'd prefer to see a new hook instead, something like #method_called(name) if that was your use case.

#5 - 08/08/2012 07:19 AM - kstephens (Kurt Stephens)

I like the idea, iff:

obj.foo.bar.baz .= to_i

behaves as:

begin
temp = obj.foo.bar
temp.baz = temp.baz.to_i
end

#6 - 08/08/2012 10:58 AM - wardrop (Tom Wardrop)

Yes, that's exactly how it would behave. To rewrite your example to make sure I understood it correctly:

`obj.foo.bar.baz = obj.foo.bar.baz.to_i`

becomes

`obj.foo.bar.baz .= to_i`

#7 - 08/09/2012 09:31 AM - kstephens (Kurt Stephens)

If we want lexical multiplicity to equal evaluation multiplicity...

Should:

obj.foo.bar[baz.daz] .= to_i

behave as?:

begin
temp1 = obj.foo.bar
temp2 = baz.daz
temp1[temp2] = temp1[temp2].to_i
end

... Since the following evaluates "obj.bar.baz" and "baz.daz" only once:

class Obj
def foo
puts "#{self}#foo"
@foo ||= Foo.new
end
end
class Foo
def bar
puts "#{self}#bar"
@bar ||= { }
end
end
class Baz
def daz
puts "#{self}#daz"
:x
end
end
obj = Obj.new
baz = Baz.new
obj.foo.bar[baz.daz] = 1

... Similarly for below:

obj.foo.bar[baz.daz] ||= 1

#8 - 08/29/2012 10:08 AM - wardrop (Tom Wardrop)

kstephens (Kurt Stephens), yes, that would be the expected result I believe. I don't think anyone would expect baz.daz to be called twice in that instance. Principle of least surprise applies here.

#9 - 10/27/2012 07:32 AM - ko1 (Koichi Sasada)

- Assignee set to mame (Yusuke Endoh)

mame-san, could you judge this ticket?

#10 - 10/27/2012 12:52 PM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee changed from mame (Yusuke Endoh) to matz (Yukihiro Matsumoto)

- Target version changed from 2.0.0 to Next Major

It requires matz's approval.

My personal comment: I agree, in fact I have thought the same thing
(and as I recall, unak implemented a patch).
But "..= to_i" resembles a variable named `to_i'. I wonder if matz
is interested or not.

--
Yusuke Endoh mame@tsg.ne.jp

#11 - 02/27/2013 09:07 AM - wardrop (Tom Wardrop)

=begin
If there are concerns about using an identifier after ".=", then perhaps a symbol could be used instead:

(({obj.foo.bar.baz .= :to_i}))

That would be somewhat consistant with the alternate block syntax (not sure what it's called):

(({['a', 'b', 'c'].each &:upcase!}))
=end

#12 - 02/27/2013 03:48 PM - duerst (Martin Dürst)

wardrop (Tom Wardrop) wrote:

> =begin
> Quite often in Ruby, I find myself doing something like: (({my_var[:foo][:bar] = my_var[:foo][:bar].to_i})) or (({obj.foo.bar = obj.foo.bar.to_i})).
> Realising this, I thought of what would be a fairly nice shorthand syntax for this, which could be: (({my_var[:foo][:bar] .= to_i})).

What about introducing to_i!. This would look more like Ruby.

#13 - 02/27/2013 08:32 PM - nobu (Nobuyoshi Nakada)

to_i! needs Object#become.

#14 - 02/28/2013 06:23 AM - funny_falcon (Yura Sokolov)

=begin
May be:

(({obj.foo.bar.baz = .to_i }))
(({obi.foo[bar.baz] = .to_s(16) }))

While this not consistent with ||= , it looks readable, imho.

Or combine both variants:

(({obj.foo.bar.baz .= .to_i }))
(({obi.foo[bar.baz] .= .to_s(16) }))

(looks like morse ;))
27.02.2013 4:07 пользователь "wardrop (Tom Wardrop)" tom@tomwardrop.com
написал:

> Issue #6841 has been updated by wardrop (Tom Wardrop).
>
> =begin
> If there are concerns about using an identifier after ".=", then perhaps a
> symbol could be used instead:
>
> (({obj.foo.bar.baz .= :to_i}))
>
> That would be somewhat consistant with the alternate block syntax (not
> sure what it's called):
>
> (({['a', 'b', 'c'].each &:upcase!}))
>
> # =end
>
> Feature #6841: Shorthand for Assigning Return Value of Method to Self
> https://bugs.ruby-lang.org/issues/6841#change-37122
>
> Author: wardrop (Tom Wardrop)
> Status: Assigned
> Priority: Normal
> Assignee: matz (Yukihiro Matsumoto)
> Category: core
> Target version: Next Major
>
> =begin

> Quite often in Ruby, I find myself doing something like:
> (({my_var[:foo][:bar] = my_var[:foo][:bar].to_i})) or (({obj.foo.bar =
> obj.foo.bar.to_i})). Realising this, I thought of what would be a fairly
> nice shorthand syntax for this, which could be: (({my_var[:foo][:bar] .=
> to_i})). How this works should be pretty self-explanatory. The (({.=}))
> operator works exactly like any other assignment operator of this nature.
>
> Would be nice to see this in Ruby 2.0. Wondering what others think of this?
> =end
>
> --
> http://bugs.ruby-lang.org/

#15 - 02/28/2013 11:23 AM - nobu (Nobuyoshi Nakada)

(13/02/28 6:16), Юрий Соколов wrote:

> =begin

You can't write in RD mode via e-mail.

> May be:
>
> (({obj.foo.bar.baz = .to_i }))
> (({obi.foo[bar.baz] = .to_s(16) }))
>
> While this not consistent with ||= , it looks readable, imho.

It's different from ||= etc, so it should not consistent with them.

Rather,

obj.foo.bar.baz += .to_i
obi.foo[bar.baz] *= .to_s(16)

might be useful.

--
Nobu Nakada

#16 - 02/28/2013 02:33 PM - Student (Nathan Zook)

http://en.wikipedia.org/wiki/Law_of_Demeter

The term that I've heard to describe obj.foo.bar.baz is "train wreck". Encouraging such seems to me to be problematic.

I am NOT saying that the transformation/normalization is wrong, just that it is being handled a the wrong level. Think about it. To make this call, you have to know about obj (which I'm assuming you should). You have to know foo about obj. You have to know bar about obj.foo. You have to know baz about obj.foo.bar. And you have to know that you want baz to be an integer. If that is not being overly intimate with your arguments, I wouldn't know what is.

If you are doing this a lot, then most likely you are normalizing a data structure. That is, your code looks like:

obj.foo.bar.baz = obj.foo.bar.baz.to_i
obj.for.bar.boz = obj.foo.bar.boz.to_f
...

If so, the to_i part is CERTAINLY not the problem! My first question would be "where does obj come from"? Most likely some sort of naive deserializing process. Many deserializers are in fact quite sophisticated, and perhaps a more informed usage would bypass ever creating the object in the undesirable form.

I have a hard time imagining any other way that one would ever get to the point that obj.foo.bar.baz = obj.foo.bar.baz.to_i would make sense. But perhaps I have missed something.

#17 - 04/03/2013 03:55 PM - wardrop (Tom Wardrop)

=begin
(({Regarding obj.foo.bar.baz = .to_i})) syntax, with the dot prefix on the ((|to_i|)), another proposal has been made further down the discussion for issue #8191. The idea is that an expression beginning with a dot could be an inferred method call on the result of the last expression.

Just consider that before using the dot prefix syntax for anything else.

```
=end
```