Ruby - Feature #7701

Non-optional (required) keyword args

01/16/2013 08:57 AM - headius (Charles Nutter)

Status:	Closed
Priority:	Normal
Assignee:	nobu (Nobuyoshi Nakada)
Target version:	

Description

=begin
I would like to see keyword args expanded to include a non-optional form, to force callers to pass in keyword arguments.

Currently, we have required, optional, and rest positional args but only optional and rest keyword args. Consistency is one small reason to add required keyword args.

They would likely take the form of keyword with no default value:

```
def foo(a:, b:)
...
end

foo(a: 1, b: 2) # ok
foo(a: 1) # ArgumentError
```

Justifications:

- Consistency with positional args. A weak justification, I know.
- Avoiding a lot of boilerplate code by users wishing to enforce keywords being passed in. Example from tenderlove:

  ```
  def foo(a: raise('pass a'), b: raise('pass b'))
  ```

- Building a rich API atop keyword args would be easier (i.e. require fewer manual checks) if you could force some keywords to be passed in. Having to check everywhere when you require a keyword argument is unpleasant.
- Keyword args already enforces that no *additional* keyword args can be passed (without **), and it seems lopsided to have no way to enforce a minimum set of keyword args.
  =end

Associated revisions

Revision 34a95669dad8843e3e9a4af683682ab2f50856dd - 03/12/2013 01:20 PM - nobu (Nobuyoshi Nakada)

required keyword arguments

- compile.c (iseq_set_arguments, iseq_compile_each): support required keyword arguments. [ruby-core:51454] [Feature #7701]
- iseq.c (rb_iseq_parameters): ditto.
- parse.y (f_kw, f_block_kw): ditto. this syntax is still experimental, the notation may change.
- vm_core.h (rb_iseq_struct): ditto.
- vm_insnhelper.c (vm_callee_setup_keyword_arg): ditto.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@39735 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 34a95669 - 03/12/2013 01:20 PM - nobu (Nobuyoshi Nakada)

required keyword arguments

- compile.c (iseq_set_arguments, iseq_compile_each): support required keyword arguments. [ruby-core:51454] [Feature #7701]
- iseq.c (rb_iseq_parameters): ditto.
- parse.y (f_kw, f_block_kw): ditto. this syntax is still experimental, the notation may change.

- vm_core.h (rb_iseq_struct): ditto.
- vm_insnhelper.c (vm_callee_setup_keyword_arg): ditto.

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@39735 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 02/16/2013 01:18 AM - nobu (Nobuyoshi Nakada)

- File 0001-required-keyword-arguments.patch added

- Description updated

Just implemented to escape from reality.

#2 - 02/16/2013 03:23 AM - Anonymous

On Sat, Feb 16, 2013 at 01:18:03AM +0900, nobu (Nobuyoshi Nakada) wrote:

> Issue #7701 has been updated by nobu (Nobuyoshi Nakada).
>
> File 0001-required-keyword-arguments.patch added
> Description updated
>
> Just implemented to escape from reality.

Sounds like fun. How can I escape?

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

#3 - 02/26/2013 01:09 AM - marcandre (Marc-Andre Lafortune)

- Assignee set to matz (Yukihiro Matsumoto)

+1 Seems great (the feature, not the escaping!)

#4 - 02/26/2013 04:02 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Assigned

- Assignee changed from matz (Yukihiro Matsumoto) to nobu (Nobuyoshi Nakada)

Accepted.

Matz.

#5 - 02/26/2013 09:53 AM - ko1 (Koichi Sasada)

(2013/01/16 8:57), headius (Charles Nutter) wrote:

> def foo(a:, b:)
> ...
> end

One (trivial) question:

Now,

```
def foo x, a:
1
end
```

is parsed as:

```
def foo x, a: 1
end
```

What happen after this change?

Yes, I see [ruby-trunk - Bug #7942].

--
// SASADA Koichi at atdot dot net

#6 - 02/26/2013 06:47 PM - nobu (Nobuyoshi Nakada)

It would be a method which has a required keyword argument and returns 1always.
Yes, it'll cause an incompatibility, but we have no real code with keyword argument yet.

#7 - 03/09/2013 05:12 PM - headius (Charles Nutter)

There's also probably no code out there that relies on the behavior ko1 pointed out. I think it's safe.

#8 - 03/09/2013 05:12 PM - headius (Charles Nutter)

Target version would be 2.1 now?

#9 - 03/10/2013 05:16 PM - mame (Yusuke Endoh)

What should Method#arity and Method#parameter return?

--
Yusuke Endoh mame@tsg.ne.jp

#10 - 03/11/2013 10:21 AM - marcandre (Marc-Andre Lafortune)

Method#parameter should return a different symbol for those, say :keyreq.

For arity, I opened #8072 regarding what I believe is a problem with the current behavior.

I would expect the arity of the following two methods to be equivalent:

```
def new_way(req, named_req1:, named_req2:, named_opt: 42, **named_rest); end
def old_way(req, options); end
```

#11 - 03/11/2013 10:53 AM - ko1 (Koichi Sasada)

(2013/03/09 17:12), headius (Charles Nutter) wrote:

> There's also probably no code out there that relies on the behavior ko1 pointed out. I think it's safe.

It is *my preference*, I don't like empty value (foo:).
But no idea what should we put it in.

--
// SASADA Koichi at atdot dot net

#12 - 03/12/2013 01:08 PM - headius (Charles Nutter)

On arity: I already feel like the numeric arity has been stretched too far. There's not really a good way to introduce another dimension of required args into that single value. Arity should reflect how many arguments are required, so I suppose if there's required kwargs they should be included...but marcandre makes a good point too (options hash passed in can fulfill the requirement in fewer args. Myself, I'd say arity should reflect the number of actual args, treating any required kwargs as a single "options" argument requirement, and people should just use parameters for any metaprogramming.

def foo(a, b:1) => arity 1
def foo(a, b:) => arity 2
def foo(a, b:1, c:, d:2, e:) => arity 2 because a single options hash could fulfill all required kwargs

Parameters in each case would be:

[[:req, :a], [:key, :b]]
[[:req, :a], [:keyreq, :b]]
[[:req, :a], [:key, :b], [:keyreq, :c], [:key, :d], [:keyreq, :e]]

ko1 doesn't like the "empty value" foo: format, but I also do not have any alternative to suggest. Syntax decisions probably end up with matz or other folks concerned about syntax. I like def foo(bar:).

My one contribution to syntax discussions would be to point out that normally, you make a position argument optional by having "=" followed by some expression. You make it non-optional by removing the "=" and the expression. In the case of kwargs, we have def foo(bar: 1) for optional kwarg, but we can only remove the expression (removing the : would make it a positional arg). The result is def foo(bar:). Perhaps if kwargs had originally been specified as def foo(bar:=1) then we could say we're following the same syntax rules as for optional args...but I think that form is pretty ugly :-)

#13 - 03/12/2013 04:53 PM - Eregon (Benoit Daloze)

I am thinking
def meth(a, b: 0, c: required)
or so would be nice to read, and it could simply be required defined
as a method raising an exception (it would be tricky to get the
missing keyword argument name though),
but of course that might be a namespace problem.

On 11 March 2013 02:47, SASADA Koichi ko1@atdot.net wrote:

> (2013/03/09 17:12), headius (Charles Nutter) wrote:
>
> > There's also probably no code out there that relies on the behavior ko1 pointed out. I think it's safe.
>
> It is *my preference*, I don't like empty value (foo:).
> But no idea what should we put it in.
>
> --
> // SASADA Koichi at atdot dot net

#14 - 03/12/2013 09:19 PM - trans (Thomas Sawyer)

=begin
Is it really a good idea to support required keyword arguments? If it is ((*required*)) shouldn't it really be a regular argument? I worry it would encourage API designers to put extraneous labels on things that arn't necessary, creating more difficult APIs to recollect w/o any advantage.

At face, why do this:

```
def foo(a, b:)
```

when you can just do this:

```
def foo(a, b)
```

=end

#15 - 03/12/2013 09:35 PM - prijutme4ty (Ilya Vorontsov)

trans (Thomas Sawyer) wrote:

> =begin
> Is it really a good idea to support required keyword arguments? If it is ((*required*)) shouldn't it really be a regular argument? I worry it would encourage API designers to put extraneous labels on things that arn't necessary, creating more difficult APIs to recollect w/o any advantage.
>
> At face, why do this:
>
> ```
> def foo(a, b:)
> ```
>
> when you can just do this:
>
> ```
> def foo(a, b)
> ```
>
> =end

If one use several required keyword arguments like in def foo(a:, b:) the developer needn't remember order of arguments, he can use args in any order with names clearing his intentions. For example you will never be looking at some file hidden far in a thirdparty gem, trying to understand, who is sender and who is recipient, if you have a method Message.send(text, from:, to:)

#16 - 03/12/2013 10:19 PM - headius (Charles Nutter)

Eregon: In addition to not being able to get the keyword argument name for that "required" error, we also couldn't really introspect via Method#parameters to know that there's a required kwarg.

trans: prijutme4ty makes probably the best point: you want to be able to use named arguments (rather than positional) and still require certain values get passed in.

Your example doesn't really make sense to me. I can call

```
def foo(a, b:)
```

using code like

```
foo(1, b: 2)
```

Where of course I can't do the same for

```
def foo(a, b)
```

Keyword arguments are not only useful on the receiver side, they're useful on the caller side.

#17 - 03/12/2013 10:51 PM - trans (Thomas Sawyer)

=begin
@prijutme4ty I think using #send as an example is a bit deceiving b/c it's one of the most well recognized predicate-to-preposition relations in the English language. Take it out of typical "mail" context and we are no better off.

To make what I am saying more clear, consider that there is nothing preventing the developer from defining #send as:

```
def send(text, sender:, recipient:)
```

So, in ((*real code*)), you ((*will*)) be looking at some file hidden far in a thirdparty gem, trying to understand what required keyword arguments you need to use. Thus the above use of required keywords gains us nothing over:

```
def send(text, sender, recipient)
```

except more typing on the caller side.

We should be careful about adding additional verbiage to APIs. It comes at a cost. It may not always be easy to recollect the order of arguments, but it can also be just as, if not more, difficult to remember the correct keyword, not to mention how to correctly spell it, leaving code open to more "bug vectors" due to typos.
=end

#18 - 03/13/2013 12:23 AM - Eregon (Benoit Daloze)

On 12 March 2013 14:19, headius (Charles Nutter) headius@headius.com wrote:

> Issue #7701 has been updated by headius (Charles Nutter).
>
> Eregon: In addition to not being able to get the keyword argument name for that "required" error, we also couldn't really introspect via Method#parameters to know that there's a required kwarg.

Indeed, it is just an idea for readability, for the rest it does not fit.

#19 - 03/13/2013 12:34 AM - trans (Thomas Sawyer)

=begin
Wouldn't it be better if parameters returned something more "OOPL".

```
def foo(a, b: 2)
end

params = method(:foo).parameters
#=> [#<Param a>, #<Param b>]

params.first.name  #=> :a
params.first.required?  #=> true
params.first.ordered?  #=> true

params.last.name  #=> :b
params.last.required?  #=> false
params.first.keyword?  #=> true
params.last.default  #=> 2
```

Then wouldn't Eergon's idea be just fine? Param could see that the default is the special "required parameter error" and report accordingly.

```
def foo(a, b: required)
end

params = method(:foo).parameters
params.last.required?  #=> true
```

=end

#20 - 03/13/2013 12:49 AM - headius (Charles Nutter)

trans: Modifying what #parameters returns is out of scope for this discussion. It already returns arrays of arrays of symbols, and changing it now

would break backward compatibility.

WRT whether required keyword arguments gains us anything over required positional arguments...I say yes, it gains us required keyword arguments. The original justification for adding required keyword arguments was so you didn't have to put your own error handling everywhere. The only way to enforce that a given keyword is being passed right now is to do something like Eregon suggested, but that's extra code compared to just omitting the kwarg's value and still doesn't give us everything that real syntactic support does.

#21 - 03/13/2013 02:56 AM - trans (Thomas Sawyer)

> trans: Modifying what #parameters returns is out of scope for this discussion. It already returns arrays of arrays of symbols, and changing it now would break backward compatibility.

Why should be out of scope? If you design to the limitation of what you have then what you get will be likewise limited.

It doesn't really matter though. The format of Method#parameter's output doesn't have to change to take into account a required default. I suppose you want something (sadly esoteric) like:

```
def foo(a, b: required); end
method(:foo).parameters
=> [[:req, :a], [:reqkey, :b]]
```

Surely that's possible.

But besides all that, you haven't really addressed my points. To say "the original justification for adding required keyword arguments was so you didn't have to put your own error handling everywhere" just begs the question. You shouldn't be doing that either. I think the perceived advantage is rather illusory in real code and more commonly will be detrimental, not beneficial. And I've cited some reasons why I think that is so.

#22 - 03/13/2013 06:50 AM - headius (Charles Nutter)

trans (Thomas Sawyer) wrote:

> > trans: Modifying what #parameters returns is out of scope for this discussion. It already returns arrays of arrays of symbols, and changing it now would break backward compatibility.
>
> Why should be out of scope? If you design to the limitation of what you have then what you get will be likewise limited.

Because the format of #parameters has very little to do with providing a required keyword argument feature. It does not move the discussion about required keyword arguments forward when we discuss new data formats for #parameters. Please stay on topic :-)

> It doesn't really matter though. The format of Method#parameter's output doesn't have to change to take into account a required default. I suppose you want something (sadly esoteric) like:
>
> ```
> def foo(a, b: required); end
> method(:foo).parameters
> => [[:req, :a], [:reqkey, :b]]
> ```
>
> Surely that's possible.

This was already suggested by multiple folks earlier in this issue. It's probably going to be :keyreq (along with :keyrest) but otherwise I consider the #parameters question solved.

> But besides all that, you haven't really addressed my points. To say "the original justification for adding required keyword arguments was so you didn't have to put your own error handling everywhere" just begs the question. You shouldn't be doing that either. I think the perceived advantage is rather illusory in real code and more commonly will be detrimental, not beneficial. And I've cited some reasons why I think that is so.

It seems a little early to claim that API designers "shouldn't" be doing something with keyword args (like requiring they be passed in) given that Ruby's only had keyword arguments for a couple weeks. It seems your only justification for rejecting required kwargs is because you believe it will be harder to remember required keywords than required positional arguments. Is that correct?

Honestly, this argument would have to work for optional keyword arguments if you want it to work for required keyword arguments. In order to make use of an API that has optional keyword arguments, you're *also* going to need to be looking at the docs and remembering the accepted keywords, their spelling, and so on. The only thing different about required keyword arguments is that you'll get an error if you forget a required kwarg, which is arguably providing *more* information and better hints for the user to fix their code. Imagine...

```
def foo(a:, b:) .... end

foo(1) => ArgumentError("Method foo' requires arguments a', `b'")
```

Bad thing?

#23 - 03/13/2013 10:31 PM - trans (Thomas Sawyer)

> Because the format of #parameters has very little to do with providing a required keyword argument feature. It does not move the discussion about required keyword arguments forward when we discuss new data formats for #parameters. Please stay on topic :-)

Wait. You are the one who brought it up in #16. I was just responding to that.

> It seems a little early to claim that API designers "shouldn't" be doing something with keyword args (like requiring they be passed in) given that Ruby's only had keyword arguments for a couple weeks.

I agree for the opposite reason! Why the rush to add required keyword arguments when Ruby took, jeez, how many years, to get keyword arguments? Wouldn't it be prudent to give it some time first?

> It seems your only justification for rejecting required kwargs is because you believe it will be harder to remember required keywords than required positional arguments. Is that correct?

Really? That's what you got from what I wrote? Well, try reading what I wrote again. If you want a summary, I suppose the closest common description is YAGNI.

> def foo(a:, b:) .... end
> foo(1) => ArgumentError("Method foo' requires arguments a', `b'")
> Bad thing?

Yes. What Ruby *really needs* is to improve it's current argument errors.

def foo(a, b) .... end

foo(1) => ArgumentError("Method foo' requires 2 arguments: a', `b'")

#24 - 03/14/2013 01:24 AM - prijutme4ty (Ilya Vorontsov)

trans (Thomas Sawyer) wrote:

> So, in ((*real code*)), you ((*will*)) be looking at some file hidden far in a thirdparty gem, trying to understand what required keyword arguments you need to use. Thus the above use of required keywords gains us nothing over:
>
> ```
> def send(text, sender, recipient)
> ```
>
> except more typing on the caller side.

In case of keyword arguments you immediately get an exception (and code can be verified even before it was run).
You need not only to write code but also to read. And in this case you can understand intention of code without looking at documentation. Probably a bit more time for writing code but much less time to read and understand.

#25 - 03/14/2013 04:55 AM - headius (Charles Nutter)

trans (Thomas Sawyer) wrote:

> Wait. You are the one who brought it up in #16. I was just responding to that.

mame brought it up in #9.

> I agree for the opposite reason! Why the rush to add required keyword arguments when Ruby took, jeez, how many years, to get keyword arguments? Wouldn't it be prudent to give it some time first?

Because they should have been there to begin with?

> Really? That's what you got from what I wrote? Well, try reading what I wrote again. If you want a summary, I suppose the closest common description is YAGNI.

I think the +1s on this bug indicate people do want it. Whether they need it remains to be seen.

> def foo(a:, b:) .... end

foo(1) => ArgumentError("Method foo' requires arguments a', `b'")
Bad thing?

Yes.

Arguing against a feature solely based on YAGNI doesn't seem very effective. Your other arguments were better because they actually discuss concerns about the feature, rather than just saying "I don't think this will be useful, so let's not do it." Do you have other substantive concerns about the feature itself?

I think having required and optional kwargs goes nicely with required and optional positional args, and I personally think it will see good use in DSLs and more readable APIs. For the same reason optional kwargs are useful to have in Ruby 2.0, required kwargs provide the same benefits *plus* guarantees that you'll receive certain keywords. It's the naming+positional flexibility of kwargs with the strictness of required positional args.

In short, I'll summarize my rebuttal this way: Do you agree that keyword args are useful? Do you agree that being able to force certain args to be given is useful? Then why wouldn't required keyword args be a natural feature to have in place?

#26 - 04/14/2013 01:55 AM - nagachika (Tomoyuki Chikanaga)

- Status changed from Assigned to Closed

I think it was committed at r39735.

#27 - 04/14/2013 09:34 AM - nobu (Nobuyoshi Nakada)

Thanks, the BTS seems missing the commit.
I've thought this ticket had been closed already.

#28 - 04/17/2013 12:23 AM - headius (Charles Nutter)

Oh, excellent! I had not noticed this was completed until today.

Thanks, everyone!

#29 - 04/17/2013 12:24 AM - headius (Charles Nutter)

Oh, a question... target version will be 2.1, yes?

#30 - 05/02/2013 10:50 PM - rogerdpack (Roger Pack)

(Same question as Charles) this will be included with 2.1 (I didn't see it in this list, but assume it will anyway? https://bugs.ruby-lang.org/projects/ruby-trunk/roadmap#Ruby-210).
Thank you.

#31 - 12/23/2021 11:41 PM - hsbt (Hiroshi SHIBATA)

- Project changed from 14 to Ruby

Files

0001-required-keyword-arguments.patch	7.82 KB	02/16/2013	nobu (Nobuyoshi Nakada)