Ruby master - Feature #16494

Allow hash unpacking in non-lambda Proc

01/09/2020 10:13 AM - zverok (Victor Shepelev)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

First of all, I fully understand the value of separating "real" keyword arguments and disallowing implicit and unexpected conversions to/from hashes.

There is, though, one **convenient style which is now broken**:

```
# words is array of hashes:
words
  .map { |text:, paragraph_id:, **rest|
    {text: text.strip, paragraph_id: paragraph_id.to_i, **rest}
  }
  .reject { |text:, is_punctuation: false, **| text.end_with?('!') || is_punctuation }
  .chunk { |paragraph_id:, timestamp: 0, **| [paragraph_id, timestamp % 60] }
  # ...and so on
```

There is several important elements to this style, making it hard to replace:

- informative errors on unexpected data structure ("missing keyword: text")
- ability to provide default values
- clear separation of declaration "what this block expects" / "what it does with expected data", especially valuable in data processing pipelines

One may argue that in some Big Hairy Very Architectured Application you should instead wrap everything in objects/extract every processing step into method or service/extract validation as a separate concern etc... But in smaller utility scripts, or deep inside of complicated algorithmic libraries, the ability to write short and clear code with explicitly declared and controlled by language arguments is pretty valuable.

This style has *no clean alternative*, all possible alternatives are either less powerful or much less readable. Compare:

```
# Try to rewrite this:
words.map { |text:, paragraph_id:, timestamp: 0, is_punctuation: false|
  log.info "Processing #{timestamp / 60} minute"
  full_text = is_punctiation ? text : text + ' '
  "<span class='word paragraph-#{paragraph_id}' data-time=#{timestamp} data-original-text=#{text}>
#{full_text}</span>"
}

# Alternative with just hashes:
words.map { |word|
  # those two used several times
  text = word.fetch(:text)
  timestamp = word.fetch(:timestamp, 0)
  log.info "Processing #{timestamp / 60} minute"
  # Absent is_punctuation is ok, it default to false
  full_text = word[:is_punctiation] ? text : text + ' '
  "<span class='word paragraph-#{word.fetch(:paragraph_id)}' data-time=#{timestamp}
 data-original-text=#{text}>#{full_text}</span>"
}

# Alternative with pattern-matching: to unpack variables, and handle default values, it will be so
mething like...
case word
in text:, paragraph_id:, timestamp:
  # skip, just unpacked
in text:, paragraph_id: # no timestamp:
  timestamp = 0
end
# I am even not trying to handle TWO default values
```

As shown above, Hash#fetch/Hash#[] style makes it much harder to understand what block expects hash to have, and how it uses hash components — and just makes the code longer and less pleasant to write and read. Pattern-matching (at least for now) is just not powerful enough for this particular case (it also has non-informative error messages, but it obviously can be improved).

My **proposal** is to **allow implicit hash unpacking** into keyword arguments in **non-lambda procs**. It would be **consistent** with implicit array unpacking, which is an important property of non-lambda procs, useful for reasons *very similar to described above*.

History

#1 - 01/14/2020 07:21 AM - mame (Yusuke Endoh)

The background of this proposal: https://bugs.ruby-lang.org/issues/14183#note-101

My personal feeling is the same as Jeremy; I'm negative because allowing the automatic Hash conversion makes the semantics complicated. However, the argument semantics of non-lambda Proc are already a mess :-) So it might be possible, though I don't like it.

However, we need to confirm if there are so many real-world use cases that rely on the old behavior. Currently, we have only one practical case found in rubocop in the discussion above, which has been already fixed (thanks to koic (Koichi ITO)!). IMO, it is far from enough to change it.

#2 - 01/14/2020 07:45 AM - zverok (Victor Shepelev)

> I'm negative because allowing the automatic Hash conversion makes the semantics complicated. However, the argument semantics of non-lambda Proc are already a mess :-) So it might be possible, though I don't like it.

I don't think it is a "mess". I think it is the "right" kind of complication to make code more easy to write. As I've already written elsewhere, there is a non-neglectable gap between what is "consistent" for computers, and what is "consistent" for humans.

> However, we need to confirm if there are so many real-world use cases that rely on the old behavior.

My proposal, in its essence, is not "please preserve the 'old' behavior, because otherwise, it would break something".

It is "please see this behavior as a modern and consistent way to use Ruby, which should not be broken by argument separation". Not something that "well, generally, not recommended, but you can do it", but as something that is seen as a "good and pretty thing, that should be promoted and used more".

I believe I am showcasing enough of the usability of the "unpacking" approach in the ticket itself (and it is not artificially constructed, just abstracted a bit example from the real-world app). I see two ways it can be refuted:

1. Show equally clean and powerful way of doing things (which, I believe, is not possible, hence the ticket)
2. Say "it doesn't matter"/"doesn't look better"/"just rewrite it with separate methods", which will be the end of the discussion :)

#3 - 01/14/2020 10:29 AM - mame (Yusuke Endoh)

I remember. I talked about the issue with matz before 2.7 release, and he said it does not matter. He may have changed his mind, so I'll confirm him at the next meeting.

In my personal opinion, it matters if there are already many real-world use cases. Otherwise, it does not matter. I guess matz has the same feelings. My current observation is that such a code is not so often written.

#4 - 01/16/2020 04:36 AM - Dan0042 (Daniel DeLorme)

My alternative proposal to accomplish the same objective: #16511

#5 - 01/16/2020 05:31 AM - matz (Yukihiro Matsumoto)

- Status changed from Open to Rejected

I admit the recent change disables something once we theoretically could. But there's no big usage for this particular feature and it doesn't worth adding even more complexity.

Matz.

#6 - 04/12/2020 09:45 AM - inopinatus (Joshua GOODALL)

I am another person with blocks affected in a production application codebase, found sixteen uses.

I see a cluster of people reporting an issue. Are we the tip of an iceberg, or the entire iceberg? There is a school of programmer that loves list comprehensions and uses them often and expressively. It may not be as theoretical an issue as hoped. The examples at https://www.ruby-lang.org/en/news/2019/12/12/separation-of-positional-and-keyword-arguments-in-ruby-3-0/ are about methods with keyword args, there is no guidance for blocks with keyword arguments.

So I have monkey-patched Enumerable to give me explicit splatting:

```
module Enumerable
  def splat
    each { |args| yield **args }
  end
end
```

Now I can write array_of_hashes.map.splat { |this:, that:, other: other_default, **rest| ... } without warnings. Also works with each, select and so on.

All the alternatives seemed quite ugly and bad for programmer happiness.

#7 - 04/12/2020 10:26 AM - zverok (Victor Shepelev)

Honestly, after how Matz has stated his opinion, I don't expect there is any room for dialogue.

The only thing I'd like to add is I feel a huge discrepancy in this decision, in my head it happens this way:

1. On the one hand, a lot of work is done to make keyword arguments "real" (and therefore more useful, less confusing, and ultimately more widely used)
2. But at the same time, changes accidentally prohibit one of the styles that is *beneficial* for keyword arguments acceptance, short, clear and *irreplaceable*. The point that *currently* the style is known lesser than it deserves for me feels inferior to the point that *potentially* it can be one of "Ruby's keyword arguments" selling points.

In other words, this decision changes the future of some concepts arguing that they were not that popular in the past. I find it quite sad.