Ruby trunk - Feature #12770

Hash#left_merge

09/16/2016 07:13 PM - dkniffin (Derek Kniffin)

Status:	Rejected	
Priority:	Normal	
Assignee:		
Target version:		

Description

I would like a Hash method that does the following:

```
a = {a: 1, b: nil, c: nil, d: nil}
b = {a: 1, b: 2, c: nil}
a.left_merge(b) # => {a: 1, b: 2, c: nil, d: nil}
```

So, it takes the first hash, and for any values that are nil, if there's a value for that key in the second hash, fill in the value from the second hash.

I've searched around a bit, and I haven't found this anywhere, so I'd like to propose a new one: Hash#left_merge. I've also got a first draft of the method definition:

```
class Hash
  def left_merge(new_hash)
    merge(new_hash) { |_, old_v, new_v| old_v || new_v }
  end
end
```

History

#1 - 09/16/2016 10:35 PM - marcandre (Marc-Andre Lafortune)

- Assignee set to matz (Yukihiro Matsumoto)

My feeling is that this won't be accepted (for lack of need and lack of a good name), but if you want to have a chance, you need provide a real use case.

Also your definition doesn't work for false values. You example also doesn't show which of the receiver or the argument takes precedence (I realize your definition does)

#2 - 09/17/2016 02:21 AM - dkniffin (Derek Kniffin)

Ah, yep, here's a better example:

```
a = {a: 1, b: nil, c: 3, d: nil}
b = {a: 1, b: 2, c: 4, e:nil}
a.left_merge(b) # => {a: 1, b: 2, c: 3, d: nil}
```

And yea, I admit it's not a great name. The reason I chose left_merge is because it's like a left join in SQL. I'm up for suggestions on improving it.

As for lack of need, that might be true, but I was surprised to see this wasn't a method. I mostly made this feature to propose the idea, and see how many other people would be interested in it.

#3 - 09/17/2016 07:47 AM - duerst (Martin Dürst)

- Assignee deleted (matz (Yukihiro Matsumoto))

- Status changed from Open to Feedback

Derek Kniffin wrote:

> I've also got a first draft of the method definition:
>
> ```
> def left_merge(new_hash)
>   merge(new_hash) { |_, old_v, new_v| old_v || new_v }
> end
> ```

If the definition is as easy as that, it doesn't look really worthwhile to create a new method. There are lots of different ways to merge hashes, and that's what the block can take care of!

However, when I actually use that definition, then for

```
a = {a: 1, b: nil, c: 3, d: nil}
b = {a: 1, b: 2, c: 4, e:nil}
a.left_merge(b)
```

I get { a: 1, b: 2, c: 3, d: nil, e: nil }
and not { a: 1, b: 2, c: 3, d: nil }.

So what exactly do you want?

#4 - 09/17/2016 07:58 PM - dkniffin (Derek Kniffin)

Ah, hmm. I think new example was a bit off as well. The return value should be:

{ a: 1, b: 2, c: 3, d: nil, e: nil }

#5 - 09/19/2016 02:02 AM - shevegen (Robert A. Heiler)

I think that the name appears to be a bit strange - if we have a left_merge, do we have a right_merge, an up_merge, a bottom_merge? :)

I can relate an example from molecular biology by the way, and how they "find names". :D

There was a dude from the UK called Southern, Edwin Southern specifically; the "Southern Blot" technique is named after him:

https://en.wikipedia.org/wiki/Southern_blot

In short, this technique allows you to "put" DNA onto a blot ("blotting"). This can then be used to hybridize with a probe.

Anyway, a bit later the same was done for RNA. What name was given for this? Northern blotting! But not because someone
was called "Northern", but simply because there already now was a Southern Technique. So it would make sense to call
the RNA-version Northern blotting. :D

And to complete this funny roundabout, a bit later a technique for proteins was added, immuno-blotting (via antibodies);
they called this "Western blotting". They are still trying to find an "Eastern Blotting" technique, which is not so
trivial (DNA, RNA and proteins are already covered after all), just to complete the four directions. Good that we do
not have more than four major directions... :D

Anyway to conclude this lengthy and possibly not so useful comment, names do matter!

#6 - 09/19/2016 03:00 AM - phluid61 (Matthew Kerwin)

Robert A. Heiler wrote:

> I think that the name appears to be a bit strange - if we have a left_merge, do we have a right_merge, an up_merge, a bottom_merge? :)

Given that it comes from SQL's JOIN then theoretically, yes, we could have a #right_merge. The directionality in the name implies which of the two "operands" is "more important" -- left_merge means that the rule for resolving collisions is: "keep the value from the left hash when it exists", "right_merge" would presumably be "use the value from the right hash when it exists."

i.e.

```
class Hash
  def left_merge hsh
    merge(hsh) {|key, left, right| left.nil? ? right : left }
  end

  #def right_merge hsh
  #  merge(hsh) {|key, left, right| right.nil? ? left : right }
  #end
end
```

That said, I've never wanted for this functionality to exist in the core. I don't do much of this sort of data mashing in Ruby, though.

#7 - 11/25/2016 08:08 AM - matz (Yukihiro Matsumoto)

- Status changed from Feedback to Rejected

Do you want to treat nils specially? If so, the name left_merge does not indicate the intention.
Besides that, you don't explain why you need a.left_merge(b) when we can b.merge(a).

If you have additional opinions, ideas or whatever, please reopen the issue.

Matz.