Ruby master - Feature #5185

Set#merge acts in place but Hash#merge does not

08/11/2011 12:16 PM - trans (Thomas Sawyer)

Status:	Rejected	
Priority:	Normal	
Assignee:	knu (Akinori MUSHA)	
Target version:	2.0.0	

Description

Waste of brain cells to have to learn and recall they are different. The expected method would be Set#merge!, Set#merge would return a new Set instance.

OTOH, why not Set#concat ? Or conversely, Array#merge ?

History

#1 - 01/12/2012 09:36 PM - tokland (Arnau Sanchez)

+1, I just got bitten by this. Hash#merge returns a new object, so one should expect Set#merge to behave the same way (Principle of Least Surprise).

Set#merge!, Set#update or Set#union! for in-places unions sound good to me.

#2 - 03/27/2012 02:34 AM - mame (Yusuke Endoh)

- Status changed from Open to Assigned

- Assignee set to knu (Akinori MUSHA)

I understand the problem, but I guess it is too late to fix it.
Anyway, I'm assigning this to knu, the maintainer of lib/set.rb.

--
Yusuke Endoh mame@tsg.ne.jp

#3 - 05/17/2012 06:23 PM - knu (Akinori MUSHA)

I feel it's hard to change this by now since #merge is a library method (for subclasses) rather than just a user method, but I could add #update as an alias for #merge and then obsolete #merge which would eventually be end-of-life'd.

#4 - 05/17/2012 11:33 PM - trans (Thomas Sawyer)

Adding #update is a good idea.

I would also add a warning to #merge stating that it's behaviour will change in a future version and to use #update instead. Then after a little while obsolete #merge altogether, but only for a bit, then bring it back with expected behaviour of producing new Set.

I advocate a slightly accelerated time scale for this transition b/c if some people are making the mistake of assuming Set#merge works like Hash's already and finding out the hard way that it's not, then I think that adds some impetus to taking the fast track.

#5 - 05/18/2012 05:19 PM - alexeymuranov (Alexey Muranov)

=begin
As the topic is surprising behavior of (({Set})) methods, i propose to deprecate (({Set#+})) as an alias of (({Set#|})), and maybe use it later for the symmetric difference. (I think symbols like (({+})) and (({|})) are too precious to alias one as another.:))

Currently (({Set#+})) is one of a few (if not the only) uses of (({#+})) for an operation which is not injective in each of the arguments: for sets (({a})), (({b})), (({c})), the equality
a + b == a + c
does not currently imply
b == c

I would have also suggested (({Set#|=})) and an alias (({Set#reverse_merge})).

I also think such methods would be natural for (({Hash})).

I can open a new issue for this if there is some interest.

P.S. I think that (({Set#+})) as the symmetric difference would look particularly good with sets of integers or symbols.

=end

#6 - 10/29/2012 02:11 AM - knu (Akinori MUSHA)

- Status changed from Assigned to Rejected

On second thought recalling my original intention, I would say #update doesn't really fit for sets.

The word "update" indicates that some data may be lost through an operation by overwriting, but Set#merge does not cause any data loss (in terms of the equality definition in Set) whereas Hash#update does. That's why I did not name it "update".

As for Set#merge, I admit it wasn't the best choice when we had Hash#merge, but you can always use the "|" operator to avoid confusion. I'd also point out that there are not many examples where #merge is not destructive. In fact, the majority works destuctively. (simple grepping in ruby's source tree and some other gems showed that)

I don't like Set#concat because the word concat[enate] usually means appending something at the bottom but Set has no sense of order.
I don't like Set#union! either because the word "union" is a noun, not a verb that a bang method is usually derived from.

#7 - 11/12/2012 11:41 PM - trans (Thomas Sawyer)

Well, maybe #merge wasn't the best choice for Hash in the first place. Who knows. I only know that polymorphism is so incredibly useful in OOP, that it's a chink in the armor for the language when inconsistencies of this nature remain.