Ruby master - Bug #13102

Confusing method name: Set#delete?

01/04/2017 05:18 PM - kaikuchn (Kai Kuchenbecker)

Status:	Rejected		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:	2.4.0	**Backport:**	2.2: UNKNOWN, 2.3: UNKNOWN, 2.4: UNKNOWN

Description

Greetings,

a colleague of mine who recently started to learn Ruby managed to greatly confuse me today when he used Set#delete? which he claimed would delete an item from a set.
Reading the documentation I suspect the method was meant to be named delete! as it behaves similiar to Array#uniq! and such methods.

If this is not a mistake, I'd still suggest to change the method name since I think it is very surprising for a method ending in a question mark to have a side effect.

Best regards,
Kai

History

#1 - 01/04/2017 05:18 PM - kaikuchn (Kai Kuchenbecker)

PS: This has been in the stdlib since at least 1.8.7 according to the docs.

#2 - 01/06/2017 06:08 AM - zenspider (Ryan Davis)

http://ruby-doc.org/stdlib-2.4.0/libdoc/set/rdoc/Set.html#method-i-delete-3F

Compare to regular delete, which always returns self...

#3 - 01/09/2017 09:06 AM - kaikuchn (Kai Kuchenbecker)

Ryan Davis wrote:

> http://ruby-doc.org/stdlib-2.4.0/libdoc/set/rdoc/Set.html#method-i-delete-3F
>
> Compare to regular delete, which always returns self...

Exactly, it is the duality we all know from methods such as Array's #uniq and #uniq!, #select and #select!, #flatten and #flatten! or String's #chomp and #chomp! etc.
It is the more dangerous version of Set#delete so, I'm pretty sure it should have a bang.

#4 - 01/26/2017 05:39 PM - sos4nt (Stefan Schüßler)

Kai Kuchenbecker wrote:

> I think it is very surprising for a method ending in a question mark to have a side effect.

Indeed, that also applies to Set#add?.

> It is the more dangerous version of Set#delete so, I'm pretty sure it should have a bang.

Not really, Set#delete also modifies the receiver.

The actual difference is that Set#delete always returns self, whereas Set#delete? only returns self if the element existed in the set, and nil otherwise.

> it is the duality we all know from methods such as Array's #uniq and #uniq!, [...]

Not quite. If a class has a method with and without ! (e.g. uniq / uniq!), then the bang-method usually modifies the receiver (often returning nil to indicate "no changes") and and the non-bang method returns a new object.

If we had both, Set#delete and Set#delete!, I would expect the non-bang method to return a new set and the band method to modify the receiver.

That would be consistent but it would also break backwards compatibility.

#5 - 02/17/2017 03:34 PM - kaikuchn (Kai Kuchenbecker)

Stefan Schüßler wrote:

> Not really, Set#delete also modifies the receiver.
>
> The actual difference is that Set#delete always returns self, whereas Set#delete? only returns self if the element existed in the set, and nil otherwise.
>
> > it is the duality we all know from methods such as Array's #uniq and #uniq!, [...]
>
> Not quite. If a class has a method with and without ! (e.g. uniq / uniq!), then the bang-method usually modifies the receiver (often returning nil to indicate "no changes") and and the non-bang method returns a new object.
>
> If we had both, Set#delete and Set#delete!, I would expect the non-bang method to return a new set and the band method to modify the receiver.
>
> That would be consistent but it would also break backwards compatibility.

Let me point you to this explanation on the use of bang in ruby method names: https://www.ruby-forum.com/topic/176830#773946
It is perpetually misunderstood for bang to mean that it "changes the receiver", it doesn't! You will see why I chose my wording as I did when you read the linked post written by Matz back in 2009.

However, you are right to point out that Set#delete does change the receiver, which means I wrongly stated that we have the duality we know from Array#uniq(!) and others. However, my main point still stands, it is weird to have Set#delete? delete an item from a set.

#6 - 02/17/2017 03:43 PM - sos4nt (Stefan Schüßler)

I was referring to the use of bang method names in the Ruby core library:

> [...] In ruby core library the dangerous method implies that when a method ends with a bang (!), it indicates that unlike its non-bang equivalent, permanently modifies its receiver. Almost always, ruby core library will have a non-bang counterpart (method name which does NOT end with !) of every bang method (method name which does end with !) that does not modify the receiver. This convention is typically true for ruby core library but may or may not hold true for other ruby libraries.

#7 - 08/24/2019 03:27 AM - jeremyevans0 (Jeremy Evans)

- Status changed from Open to Rejected

I don't think this is a bug, I believe it is intentional. Using ? for mutating methods is uncommon, but not wrong. You can use these like predicate methods, which is probably why they end in ?:

```
if set.delete?(object)
  # object was in the set and deleted
else
  # object was not in the set
end
```

#8 - 08/24/2019 06:45 AM - Eregon (Benoit Daloze)

This is at least inconsistent with Array#delete, which could be used for the same case as above but doesn't end with a ?.
I don't think there are many good examples for predicates with side effects, is it?

A good part of the Set API is quite inconsistent with the rest, unfortunately (e.g., Set#merge being inplace).
Maybe we could slowly deprecate these inconsistent methods and align the API with Array and Hash?

#9 - 08/24/2019 02:56 PM - jeremyevans0 (Jeremy Evans)

Eregon (Benoit Daloze) wrote:

> This is at least inconsistent with Array#delete, which could be used for the same case as above but doesn't end with a ?.

This is not exactly true, as Set#delete? returns self and not the item if deleted:

```
require 'set'
set = Set[true, false]
p(set.delete?(false) ? 1 : 0)
# => 1
p(set.delete?(false) ? 1 : 0)
# => 0

array = [true, false]
p(array.delete(false) ? 1 : 0)
# => 0
p(array.delete(false) ? 1 : 0)
# => 0
```

> I don't think there are many good examples for predicates with side effects, is it?

No, there aren't a lot of good examples. I don't think there are any examples in core, and I'm not sure about the rest of stdlib.

> A good part of the Set API is quite inconsistent with the rest, unfortunately (e.g., Set#merge being inplace).
> Maybe we could slowly deprecate these inconsistent methods and align the API with Array and Hash?

We could definitely do that, but that is a feature request, not a bug report.