Ruby master - Feature #18395

Introduce Array#subtract! for performance

12/07/2021 09:21 PM - schneems (Richard Schneeman)

<table>
<thead>
<tr>
<th>Status:</th>
<th>Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority:</td>
<td>Normal</td>
</tr>
<tr>
<td>Assignee:</td>
<td></td>
</tr>
<tr>
<td>Target version:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

PR: [https://github.com/ruby/ruby/pull/5110#issuecomment-984345309](https://github.com/ruby/ruby/pull/5110#issuecomment-984345309)

It is common to use `-=` to modify an array to remove elements. Also, there is Array#difference which accepts multiple arguments. Both of these methods allocate an intermediate array when the original array could be re-used.

I am proposing we add an API onto Array that allows the programmer to perform a subtraction/difference operation and mutate the original array.

I am proposing Array#subtract!. Reasons why I did not choose Array#difference! are discussed in the comments of the PR. I'm also happy to discuss alternative names.

```ruby
ary = [0, 1, 1, 2, 1, 1, 3, 1, 1]
ary.subtract!(1) #=> [0, 2, 3]
ary #=> [0, 2, 3]
```

```ruby
ary = [0, 1, 2, 3]
ary.subtract!(3, 0, 1, 3) #=> [2]
ary #=> [2]
```

**History**

#1 - 12/08/2021 05:16 AM - sawa (Tsuyoshi Sawada)

I am strongly against it. It is a practice in Ruby for methods that destructively remove something from the receiver to return the removed value rather than the modified receiver. I also do not think it is worth adding a new method just for 1.22x or so performance improvement.

We have Array#delete already, the only difference from your proposal (other than the receiver) being that it does not accept multiple arguments. I think a better proposal would be to let Array#delete accept multiple arguments.

```ruby
a = [1, 2, 3, 2, 1]
b = [4, 2, 3]
a.delete(*b) #=> 2
a #=> [1, 1]
```

#2 - 12/08/2021 02:55 PM - schneems (Richard Schneeman)

We have Array#delete already, the only difference from your proposal (other than the receiver) being that it does not accept multiple arguments. I think a better proposal would be to let Array#delete accept multiple arguments.

In order to preserve the original behavior of returning the deleted elements, you would need to track them which would necessitate an additional array allocation so it would be no faster than Array#-=

We could return self or nil depending on whether or not an element is removed similar to Array#reject! I believe that would still be faster.

My other concern with allowing multiple values in Array#delete would be a possible performance degradation with existing code that only passes in 1 value. It's my experience through this PR that methods accepting multiple values are slower (however this may be due to a mistake on my part). I posted my code and comparison in this comment [https://github.com/ruby/ruby/pull/5110#issuecomment-967276896](https://github.com/ruby/ruby/pull/5110#issuecomment-967276896).

It is a practice in Ruby for methods that destructively remove something from the receiver to return the removed value rather than the modified receiver.

That is very interesting. I never knew there was a pattern to the return results of bang methods. The differences have always been confusing to me, but I had never considered that they might behave differently if they are destructive. If this is the convention is it documented somewhere, could we...
If the return value is the primary concern then we could follow in the pattern of Array#reject! to return either self or nil.

I personally dislike the behavior as it means I cannot substitute bang methods that are chained because they may return different results. That means to write faster Ruby code, I must write worse ruby code. I do respect the precedent and I'm happy to change the return value.

I also do not think it is worth adding a new method just for 1.22x or so performance improvement.

That's much harder to workaround. Even the 1.22x number might be high.

I am new to Ruby c internals, so maybe it is possible that there are more optimizations that can be made with my code if it no longer has to allocate the extra array.

I appreciate your time and attention. We can close the ticket and pull request unless someone disagrees.

#3 - 12/08/2021 03:26 PM - mame (Yusuke Endoh)
- Description updated

It would be very helpful if you could write a code example in the description to help us understand your proposal at a glance. I copied (and a bit tweaked) it from the rdoc you write in PR. Feel free to change the example if you don't like it.

#4 - 12/08/2021 04:38 PM - sawa (Tsuyoshi Sawada)

schneems (Richard Schneeman) wrote in #note-2:

We have Array#delete already, the only difference from your proposal (other than the receiver) being that it does not accept multiple arguments. I think a better proposal would be to let Array#delete accept multiple arguments.

In order to preserve the original behavior of returning the deleted elements, you would need to track them which would necessitate an additional array allocation so it would be no faster than Array#-=.

The original behavior of Array#delete is to return the element that last matched (= was deleted). This behavior should be retained (Please see my code example in the previous comment). There is no need to return all deleted elements, and there is no need to allocate an additional array.

It is a practice in Ruby for methods that destructively remove something from the receiver to return the removed value rather than the modified receiver.

That is very interesting. I never knew there was a pattern to the return results of bang methods. The differences have always been confusing to me, but I had never considered that they might behave differently if they are destructive. If this is the convention is it documented somewhere, could we document it somewhere? Where would be a good place to document this convention?

There is a common misunderstanding in assimilating destructive methods with bang methods. As Matz has repeatedly said, there is no connection between them. My comment is concerned with destructive methods, not bang methods (but note that some destructive methods are bang methods). I never mentioned bang methods in the previous comment. For the practice I mentioned, please see the documents of, for example, Array#shift, Array#pop, Array#delete, and String#gsub!.

I appreciate your time and attention. We can close the ticket and pull request unless someone disagrees.

I am thankful for your detailed response. Although I am against the idea of introducing a new method, I am not against having a way to destructively remove multiple elements from an array.

#5 - 12/08/2021 04:42 PM - byroot (Jean Boussier)

Have you benched array.reject! { |e| array_or_set.include?(e) }?