Strings need a named method like `dup` that doesn't duplicate if receiver is mutable

02/21/2022 08:35 PM - danh337 (Dan H)

**Status:** Open
**Priority:** Normal
**Assignee:**
**Target version:**

**Description**
This is related to #16295, but focuses only on the .+@ part.

Currently we can use .dup in a method chain when we need to mutate a String.

However there are cases where the code's context expects the String to be mutated. In cases like this, .dup always works, but we don't want to duplicate a String that is already mutable.

Since .+@ looks more like an operator, it can be unintuitive in a method chain, so this is asking for a new named method that can be used in its place, instead of always .dup.

For example:

```ruby
def add_result_text(buffer, new_result)
  text = "#{new_result.count} #{new_result.input} #{do_fancy_calc(new_result)}\n"
  buffer.dup_if_immutable << text
  # ^^^^^^^^^^^^^^^^^ new method?
end

buffer = "" # ...maybe immutable

get_lots_of_results.each do |result|
  buffer = add_result_text(buffer, result) # In case it was dup'ed
end
```

**Related issues:**
Related to Ruby master - Feature #16295: Chainable aliases for String#-@ and ...

Closed
Description updated again. I missed the subtlety in the case that the buffer had to be dup'ed.

There are many more examples in [#16295](#16295). The line in question could be something like

```ruby
buffer.dup_if_immutable.tap(&:downcase!).append(text)
```

Or lots of other permutations of chained methods before or after the `.dup_if_immutable`.

---

**#6 - 02/22/2022 10:58 AM - Eregon (Benoit Daloze)**
- Related to Feature #16295: Chainable aliases for String#-@ and String#+@ added

**#7 - 02/22/2022 11:07 AM - Eregon (Benoit Daloze)**

+@/dup_if_immutable semantics are only safe to use if you know you "own" the receiver and in other words it's OK to mutate it. But in that case it's also useless because if you "own" the receiver you already know it's mutable.

For example think about:

```ruby
NAME = "Benoit"
#
str = add_result_text(NAME, new_result)
```

This is a bug (it mutates NAME which should never be mutated), and hence why dup is more appropriate.

If you do want a buffer argument, then the method should simply mutate it directly, i.e., `buffer << text`, no need for `+@/dup_if_immutable`. And that's why I think `dup_if_immutable` is basically useless as it's rarely if ever safe to use except when it's on a literal string, and then `str = +"foo"; str.mutate_it_some_way!` already works well enough.

---

**#8 - 02/22/2022 11:08 AM - Eregon (Benoit Daloze)**

A real-world example would be interesting. But I guess there isn't a compelling one.

---

**#9 - 02/22/2022 02:56 PM - danh337 (Dan H)**

Eregon (Benoit Daloze) wrote in [#note-8](#note-8):

A real-world example would be interesting. But I guess there isn't a compelling one.

There are good examples in the other ticket where `.+@` is used but is clunky, so the method name is desirable. I am adding to that this case, where I need optimal code, that doesn't raise exception or duplicate a String as the only two options.

I'm not sure what you mean by "real-world" example using a feature that doesn't yet exist. This is from [#16295](#16295).

This already has made some of my production code ugly, when using tap. I have to say:

```ruby
(+some_object.send(a_method)).tap { |value| value << "blah" }
or some_object.send(a_method).+@.tap { |value| value << "blah" }
```

Neither of these looks like good Ruby. I'd rather say `some_object.send(a_method).thaw.tap { |value| value << "blah" }

If I'm on a team of devs, and I have other people calling my code, and I want it to be as optimized as possible, and I don't want to raise exception if I don't absolutely have to, then it's not basically useless to have a clearly named method that ensures at most 1 duplication.

I don't believe `.+@` is useless. I mean, it's useless until you need it. But it's not Ruby-ish to be required to use that in a method chain.

---

**#10 - 02/22/2022 06:44 PM - Eregon (Benoit Daloze)**

I meant code in a gem or in some open-source repository, so we can see the context and the need.

That example seems a clear case for String interpolation:

```
"#{some_object.send(a_method)}blah"
```

and I don't want to raise exception if I don't absolutely have to

That is I believe an anti-pattern.

Concretely, a method should either take a String and will never mutate it, or it needs a mutable String/a buffer (and will or can mutate it).

What I'm saying is "might or might not mutate the argument (depending on frozen state)" is a bad design.

A method should be clear about what it expects, it's very hard to work with a method which might mutate or not what you give it.

---

**#11 - 02/22/2022 06:47 PM - Eregon (Benoit Daloze)**
and I want it to be as optimized as possible

Also if that's the goal I would also avoid .tap and just use a local variable, and then +@ works if that's really the semantics you want:

```ruby
buffer = +some_object.send(a_method)
buffer << "blah"
```

But the string interpolation seems more readable and is likely as efficient.

#12 - 02/22/2022 07:02 PM - danh337 (Dan H)

Eregon (Benoit Daloze) wrote in #note-10:

> I meant code in a gem or in some open-source repository, so we can see the context and the need.

My team has needed this in its own code. I have given a pattern for it. I don't have time to search for examples from all gems or repos in the world.

That example seems a clear case for String interpolation:

```
#{some_object.send(a_method)}|blah
```

That is not the same code. You are always making a new String. I thought .+@ only dups if the receiver is frozen. The a_method result could be mutable.

and I don't want to raise exception if I don't absolutely have to

That is I believe an anti-pattern.

Concretely, a method should either take a String and will never mutate it, or it needs a mutable String/a buffer (and will or can mutate it).

What I'm saying is "might or might not mutate the argument (depending on frozen state)" is a bad design.

A method should be clear about what it expects, it's very hard to work with a method which might mutate or not what you give it.

Thanks for sharing your opinions.

I know your job here is to weed out frivolous requests, but this experience makes me sad. Wasting time.

My team will continue to use the awkward .+@ where we need it, or I will add a name for it in our own extensions lib.

#13 - 02/22/2022 07:10 PM - danh337 (Dan H)

Fixed in our extensions lib.

```ruby
class Object
  # A chainable name mainly for `String#+@`
  #
  def thaw
    frozen? ? dup : self
  end
end
```

#14 - 02/22/2022 07:49 PM - Dan0042 (Daniel DeLorme)

```ruby
(+some_object.send(a_method)).tap { |value| value << "blah" }
or
some_object.send(a_method).+@.tap { |value| value << "blah" }
```

What is "some_object"? What is "a_method"? What is "blah"? None of these give any indication of what this code is trying to achieve, what is the context. The only thing this shows is the syntax. But we already know what syntax is being asked. The example has been simplified/abstracted to a point where it doesn't say anything. I haven't yet seen a concrete, compelling example either in this thread or the other one. I even tried searching in gems for code like (+var but found nothing.

If I look at the example in the description, it seems obvious that if you're going to append text to a buffer, it's better to first make sure the buffer is writeable like:

```ruby
buffer = +buffer
get_lots_of_results.each do |result|
```
add_result_text(buffer, result)

String#+@ was meant to be used with string literals in combination with frozen_string_literal: true, and I don't quite see what you're trying to (ab)use it for.

#15 - 02/22/2022 08:31 PM - danh337 (Dan H)

At one time there were other folks who cared about this, in #16295, but it feels like I'm the only one now. And it feels like I cannot convince anyone here with any amount of rationale. And it feels like more code examples won't be enough.

If there is already a named method that does what String#+@ does I am happy to use it. Use or abuse is all relative.

#16 - 02/26/2022 11:20 PM - danh337 (Dan H)

- File driver.rb added

This is my last attempt to show why this named method is needed. The driver.rb file shows the patterns that apply to all sorts of code.

#17 - 02/26/2022 11:56 PM - danh337 (Dan H)

- File driver.rb added

Updated timing check code to be more fair.

Files

<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Date</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>driver.rb</td>
<td>12.3 KB</td>
<td>02/26/2022</td>
<td>danh337 (Dan H)</td>
</tr>
<tr>
<td>driver.rb</td>
<td>12.7 KB</td>
<td>02/26/2022</td>
<td>danh337 (Dan H)</td>
</tr>
</tbody>
</table>