Ruby master - Feature #12745

String#(g)sub(!) should pass a MatchData to the block, not a String

09/09/2016 06:34 PM - herwin (Herwin W)

Status:	Feedback	
Priority:	Normal	
Assignee:	matz (Yukihiro Matsumoto)	
Target version:		

Description

A simplified (and stupid) example: replace some placeholders in a string with function calls

```
def placeholder(val)
  raise 'Incorrect value' unless val == 'three'
  '3'
end

str = '1.2.[three].4'
str.gsub!(/\[(\w+)\]/) { |m| placeholder(m) }
```

This raises the 'incorrect value' because we don't pass the match 'three', but the full string '[three]'. It looks like we have 3 options to fix that:

1. Match [three] instead of three in the placeholder replacement method
2. Pass m[1..-2] instead of m to the method (or strip it in placeholder)
3. Use $1 in the method call, ignore the value that's passed to the block

Options 1 and 2 look kind of code duplication to me (and they're possible in the simplified example, but might get tricky in real situations). I don't like option 3 because you completely ignore the value that's been passed to the block in favor of global variables, you can't use named captures, and writing code this way makes it incompatible with Rubinius.

I think it would be more logical to pass a MatchData (like what you'd get with String#match) instead of a String to the block. The #to_s returns the whole string, so in 90% of the use cases the code could remain unaltered, but the remaining 10% makes it a change that shouldn't be backported to 2.3.

Attached is a very naive patch to pass a matchdata to the block called by String#sub. The additional change in rbinstall.rb was required to run make install, which actually shows an incompatiblity (which I hadn't anticipated)

Related issues:

Related to Ruby master - Feature #6802: String#scan should have equivalent yi...	**Assigned**	
Related to Ruby master - Feature #5749: new method String#match_all needed	**Assigned**	
Related to Ruby master - Feature #5606: String#each_match(regexp)	**Feedback**	
Related to Ruby master - Feature #546: String#gsub □ Strnig#scan □□□□□□□□□□□□□□	**Rejected**	**09/06/2008**

History

#1 - 10/11/2016 08:54 AM - akr (Akira Tanaka)

matz's old idea: add String#sg (or String#gs).
ruby-dev:33553

#2 - 10/11/2016 11:29 AM - shyouhei (Shyouhei Urabe)

We looked at this issue in developer meeting today.

While matz's suggestion of new method was excavated from the past, myself and others argued possibility of extending the gsub method that exists today. Proposals include:

1. what about just passing a MatchData instead of a String. That hurts no one because almost every time when we need a block to pass to gsub etc, what is needed is $1.
2. what about passing a MatchData in addition to a String like gsub(){|string, md|}. It is 100% safe even when the block parameter is used.
3. what about extending the block parameter string with a module, like open-uri does, to define additional method.
4. there are of course attendees who think a separate method is the cleanest solution.

My idea was #3 but in reality #2 can be a perfect and easy solution. What do you think matz?

#3 - 10/11/2016 03:24 PM - akr (Akira Tanaka)

Shyouhei Urabe wrote:

> 1. what about passing a MatchData in addition to a String like gsub(){|string, md|}. It is 100% safe even when the block parameter is used.

Surprisingly, it may be possible.

```
% ruby -e '
p "abc".gsub(/b/) {|str| "(#{str})" }
String.prepend Module.new {
  def gsub(*args)
    super(*args) {|str| yield str, $~ }
  end
}
p "abc".gsub(/b/) {|str| "(#{str})" }
p "abc".gsub(/b/) {|str, md| "(#{md.inspect})" }
'
"a(b)c"
"a(b)c"
"a(#<MatchData \"b\">)c"
```

It is important that String#gsub yields two arguments
instead of an two-elements array which causes incompatiblity.

```
% ruby -e '
p "abc".gsub(/b/) {|str| "(#{str})" }
String.prepend Module.new {
  def gsub(*args)
    super(*args) {|str| yield [str, $~] }
  end
}
p "abc".gsub(/b/) {|str| "(#{str})" }
p "abc".gsub(/b/) {|str, md| "(#{md.inspect})" }
'
"a(b)c"
"a([\"b\", #<MatchData \"b\">])c"
"a(#<MatchData \"b\">)c"
```

#4 - 10/11/2016 07:21 PM - herwin (Herwin W)

As reference: the e-mail Akira Tanaka mentioned can be found at http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-dev/33553, or with a pretty terrible translation to English at
https://translate.google.com/translate?sl=ja&tl=en&u=http%3A%2F%2Fblade.nagaokaut.ac.jp%2Fcgi-bin%2Fscat.rb%2Fruby%2Fruby-dev%2F33553

#5 - 10/11/2016 07:31 PM - herwin (Herwin W)

About the suggestions by Shyouhei Urabe: even though option 2 is very pragmatic (it solves the problem, keeps backwards compatibility and is a small change), the idea of the following code looks a bit off to me:

```
str.gsub!(/\[(\w+)\]/) { |_, m| placeholder(m) }
```

I guess you would use either a String or a MatchObject in the block, but never both (if you'd need them both, you could get the String from the MatchObject of course). Always having to ignore the first argument to the block feels a bit like a hack. Not that I have a better suggestion though.

A separate method that passes a MatchData object would be another solution that feels a bit hacky: String#gsub and String#gs would be almost identical, and which method would be preferred if you don't add a block, but use a second parameter as the replacement string? And without looking at the documentation, I would have no idea what String#gs/String#sg would do, the names are very undescriptive.

#6 - 11/25/2016 07:18 AM - matz (Yukihiro Matsumoto)

- Assignee set to matz (Yukihiro Matsumoto)

- Status changed from Open to Feedback

Out of Shyouhei's 4 options,

1. not acceptable for compatibility's sake
2. not excited, may cause compatibility problem. besides that, String#scan cannot be enhanced by this option
3. looks nice but I have performance concern
4. the best solution, if we have a good name candidate.

Matz.

#7 - 11/26/2016 08:45 PM - herwin (Herwin W)

What about sub_md as a name?

#8 - 12/04/2016 05:40 PM - herwin (Herwin W)

I posted this link to IRC (#ruby on freenode), just to see if anyone had a good name suggestion. The suggestions #matchsub and #msub were offered. I don't really like them, #matchsub sounds like it tries to subtitute a match (exactly what #sub does, #msub reminds me of the /m modifier or regex (multiline, just as #gsub is named after the /g modifier (global)) instead of a MatchData object. (Although I smiled at the suggestion #gsubthewayitshouldhavebeenallalong, the name reminds me to much of mysql_real_escape_string to take serious)

Another option that was suggested was adding a keyword argument to toggle the behaviour. I actually liked that proposal: it keeps backwards compatibility and doesn't result in an explosion of the number of methods

#9 - 12/15/2016 10:34 AM - Eregon (Benoit Daloze)

Maybe this functionality should just be an extra keyword argument to these methods?

Like
str.gsub!(/[(\w+)]/, md: true)

Not as concise, but much better on compatibility and clarity of the intent.

#10 - 12/16/2016 01:00 AM - shyouhei (Shyouhei Urabe)

Benoit Daloze wrote:

> Maybe this functionality should just be an extra keyword argument to these methods?
>
> Like
> str.gsub!(/[(\w+)]/, md: true)

We already use that feature for another purpose.

```
p "emdash".gsub(/md/, "md" => true) #=> "etrueash"
```

#11 - 01/19/2017 09:22 AM - akr (Akira Tanaka)

How about String#gsb and String#sb ?

- gsb and sb are shorter and gsub and sub.
- gsb and sb is readable as g-substitute and substitute.

#12 - 01/19/2017 07:43 PM - herwin (Herwin W)

Regarding String#gsb and String#sb: It is far from clear what the difference between #sub and #sb is just by the method names. And personally I don't care that much about the length of the method names, I prefer clearness over shortness.

#13 - 01/20/2017 07:50 AM - duerst (Martin Dürst)

Herwin W wrote:

> Regarding String#gsb and String#sb: It is far from clear what the difference between #sub and #sb is just by the method names. And personally I don't care that much about the length of the method names, I prefer clearness over shortness.

Agreed. I also think that the new methods should have longer names (maybe shorter than gsub_with_match_object, but still longer than just gsub), because most of the time, the old methods will do just fine (they did so for more than 20 years of Ruby history).

#14 - 03/15/2018 08:09 AM - akr (Akira Tanaka)

How about String#subm and String#gsubm ?

It is bit longer but not so long.

#15 - 03/15/2018 08:10 AM - matz (Yukihiro Matsumoto)

subm andgsubm are acceptable.

Matz.

#16 - 03/15/2018 10:49 AM - Eregon (Benoit Daloze)

shyouhei (Shyouhei Urabe) wrote:

> Benoit Daloze wrote:
>
> > Maybe this functionality should just be an extra keyword argument to these methods?
> >
> > Like
> > str.gsub!(/[(\w+)]/, md: true)
>
> We already use that feature for another purpose.
>
> ```
> p "emdash".gsub(/md/, "md" => true) #=> "etrueash"
> ```

Not with a Symbol though.
So I think gsub(/regexp/, md: true) would not conflict with existing usages.

matz (Yukihiro Matsumoto) wrote:

> subm andgsubm are acceptable.

Those names sound very cryptic to me.

#17 - 11/08/2018 07:31 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #6802: String#scan should have equivalent yielding MatchData added

#18 - 11/08/2018 07:31 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #5749: new method String#match_all needed added

#19 - 11/08/2018 07:31 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #5606: String#each_match(regexp) added

#20 - 11/08/2018 07:32 AM - shyouhei (Shyouhei Urabe)

- Related to Feature #546: String#gsub □ Strnig#scan □□□□□□□□□□□□□□□ added

#21 - 02/21/2019 07:17 PM - sawa (Tsuyoshi Sawada)

What about having Enumerator#with_m, such that

```
gsub(pattern).with_m{|match, match_data| block} → new_str
sub(pattern).with_m{|match, match_data| block} → new_str
```

And if we want to expand that to scan, we can introduce

```
scan → enumerator
```

And then do

```
scan.with_m(pattern){|(match, ...), match_data| block} → str
```

#22 - 02/21/2019 08:02 PM - jeremyevans0 (Jeremy Evans)

sawa (Tsuyoshi Sawada) wrote:

> What about having Enumerator#with_m, such that
>
> ```
> gsub(pattern).with_m{|match, match_data| block} → new_str
> sub(pattern).with_m{|match, match_data| block} → new_str
> ```

This doesn't make sense to add to Enumerator in my opinion, since most Enumerators will not be created from gsub/sub:

```
[1].each.with_m{}
```

Possibly gsub and sub could return an instance of an Enumerator subclass that supports the with_m, but that seems like overkill.

Reading through the previously discussed alternative approaches:

- Passing a second argument to the block breaks if the block is a lambda.
- A keyword argument via :md is currently allowed but ignored. I don't think that breaks compatibility, but with the discussion of keyword arguments in #14183, introducing an :md keyword argument may make it harder to transition such code to Ruby 3.

- subm and gsubm are both fairly cryptic in terms of method names, but compared to other approaches, new method names seem to be the best way to handle this if we want to add support for it.

We could leave things as they are and use $1/$2/$~ to access the captures instead the block.

#23 - 10/18/2019 07:20 AM - shugo (Shugo Maeda)

jeremyevans0 (Jeremy Evans) wrote:

> - Passing a second argument to the block breaks if the block is a lambda.
> - A keyword argument via :md is currently allowed but ignored. I don't think that breaks compatibility, but with the discussion of keyword arguments in #14183, introducing an :md keyword argument may make it harder to transition such code to Ruby 3.
> - subm and gsubm are both fairly cryptic in terms of method names, but compared to other approaches, new method names seem to be the best way to handle this if we want to add support for it.

I prefer to the last approach, but don't like names subm and gsubm.

How about repl and repl_all?
JavaScripts's replace is similar to Ruby's sub, but Ruby has already replace and it's destructive.
repl is an abbreviation for replace. grepl is confusing with grep, so I propose repl_all instead.

And I prefer to match_all for the MatchData version of scan (#5749).

Files

ruby_string_sub_matchdata.diff	952 Bytes	09/09/2016	herwin (Herwin W)