Ruby master - Feature #13016
String#gsub(hash)

12/08/2016 08:21 AM - shyouhei (Shyouhei Urabe)

Status: Rejected
Priority: Normal
Assignee: 
Target version: 

Description
Background: I wanted to drop NKF dependency of my script. By doing so I noticed that I can't purge NKF.nkf `-Z4'. It can neither be rewritten using String#tr, String#encode, nor String#unicode_normalize. It is doable using String#gsub theoretically, but that requires a hand-crafted nontrivial regular expression that exactly matches what Z4 expects to convert. This is almost impossible to do, and is definitely not something debuggable.

Proposal: extend String#gsub so that it also accepts hash as its only argument, specifying input-output mapping.

```ruby
# now
def convert str
  require 'nkf'
  NKF.nkf '-Z4xm0', str
end

# proposed
def convert str
  map = { "\u3002" => "\uFF61", "\u300C" => "\uFF62", ... }
  str.gsub map
end
```

Related issues:
Has duplicate Ruby master - Feature #14443: Omit 'pattern' parameter in '(g)s...
Closed

History

#1 - 12/08/2016 08:22 AM - shyouhei (Shyouhei Urabe)
- Tracker changed from Bug to Feature

#2 - 12/08/2016 08:29 AM - akr (Akira Tanaka)
Is str.gsub(map) a shortcut for str.gsub(Regexp.union(map.keys)) { map[$&] } ?

#3 - 12/08/2016 08:50 AM - shyouhei (Shyouhei Urabe)
Akira Tanaka wrote:

  Is str.gsub(map) a shortcut for str.gsub(Regexp.union(map.keys)) { map[$&] } ?

Kind of yes. I was thinking of str.gsub(Regexp.union(map.keys), map) -equivalent behaviour.

#4 - 12/08/2016 09:22 AM - duerst (Martin Dürst)

Shyouhei Urabe wrote:

  I noticed that I can't purge NKF.nkf `-Z4'. It can neither be rewritten using String#tr, String#encode, nor String#unicode_normalize.

Can you give (a pointer to) a detailed description of what NKF, and in particular NKF.nkf -Z4, does exactly? For example, I can't find it at http://blog.layer8.sh/ja/2012/03/31/nkf_command_option/. The following may be related: -Z X0208全国標準的ASCII空間の記号-Z1X0208\ \ASCII space \ASCII space X0208全国標準的ASCII空間の記号-Z1X0208\ ..."

  It is doable using String#gsub theoretically, but that requires a hand-crafted nontrivial regular expression that exactly matches what Z4 expects to convert. This is almost impossible to do, and is definitely not something debuggable.

Please note that String#unicode_normalize, as currently implemented, also uses some huge regular expressions (though program-generated). And also has (hopefully) successfully been debugged, although with the help of testing data from Unicode.
Martin Dürst wrote:

Shyouhei Urabe wrote:

I noticed that I can't purge NKF.nkf `-Z4'. It can neither be rewritten using String#tr, String#encode, nor String#unicode_normalize.

Can you give (a pointer to) a detailed description of what NKF, and in particular NKF.nkf -Z4, does exactly? For example, I can't find it at http://blog.layer8.sh/ja/2012/03/31/nkf_command_option/

It seems there are quite few resources describing this feature on line.

- I learned it by command line "nkf --help". The output says "4: JISX0208 Katakana to JISX0201 Katakana".
- A few minutes of googling let me realize that it has been there at least since 2009. https://osdn.net/projects/nkf/news/17482 (Japanese).
- It seems this is the particular commit which implemented the feature in nkf:
https://github.com/nurse/nkf/commit/958de30bc09aeef38f2a44b5da0d8b1bb3c79e7d3
- and then copied into our repository in this commit:
https://github.com/ruby/ruby/commit/086e5b1a63d77bf5a4ebe10396a430d544f8505

So in short it converts characters into what Unicode calls the "Halfwidth" ones.

Please note that String#unicode_normalize, as currently implemented, also uses some huge regular expressions (though program-generated). And also has (hopefully) successfully been debugged, although with the help of testing data from Unicode.

Thank you. That still sounds like a hustle to me. The proposed functionality would make it a lot easier for me to emulate NKF's Z4.

Shyouhei Urabe wrote:

It seems there are quite few resources describing this feature on line.

Thanks!

That still sounds like a hustle to me.

Can you write the last sentence in Japanese? The word 'hustle' has lots of meanings, some of them confusing.

The proposed functionality would make it a lot easier for me to emulate NKF's Z4.

I agree it would make it easier. But I'm not sure about "a lot". The main work needed to implement it is to create the hash. My understanding is that you still have to do that by hand. My suggestion would be to use literal characters, not \u escapes, in most cases because that makes it much easier to spot errors.

Compared to creating the hash, the shortening from

```
str.gsub(Regexp.union(map.keys)) { map[$&] }
```
to

```
str.gsub(map)
```

seems to be a minor simplification, and one that is easily done by defining a new method:

```
class String
  def hsub(map)
    gsub(Regexp.union(map.keys)) { map[$&] }
  end
end
```

I'm not against this feature, but I think it would be good to have some more examples of where it could be useful, and some check that we don't want to use String#gmap(Hash) with some other meaning in the future.

Shyouhei Urabe wrote:

It seems there are quite few resources describing this feature on line.

Thank you. That still sounds like a hustle to me. The proposed functionality would make it a lot easier for me to emulate NKF's Z4.

#6 - 12/08/2016 11:30 AM - duerst (Martin Dürst)

Thanks!

That still sounds like a hustle to me.

Can you write the last sentence in Japanese? The word 'hustle' has lots of meanings, some of them confusing.

The proposed functionality would make it a lot easier for me to emulate NKF's Z4.

I agree it would make it easier. But I'm not sure about "a lot". The main work needed to implement it is to create the hash. My understanding is that you still have to do that by hand. My suggestion would be to use literal characters, not \u escapes, in most cases because that makes it much easier to spot errors.

Compared to creating the hash, the shortening from

```
str.gsub(Regexp.union(map.keys)) { map[$&] }
```
to

```
str.gsub(map)
```

seems to be a minor simplification, and one that is easily done by defining a new method:

```
class String
  def hsub(map)
    gsub(Regexp.union(map.keys)) { map[$&] }
  end
end
```

I'm not against this feature, but I think it would be good to have some more examples of where it could be useful, and some check that we don't want to use String#gmap(Hash) with some other meaning in the future.

Akira Tanaka wrote:

Ruby has enough feature to implement String#hsub as Martin-sensei said.

However the performance of String#hsub is not good because it creates regexp object each time. I guess creating regexp for big table each time is not acceptable for most cases.
We looked at this issue in yesterday's developer meeting.

While I claimed the use of regular expression is an implementation detail that I don't want to care about, attendees there said it is better to expose compiled structure (be they regexp) for performance. I agree with that, so I give up this proposal.

One note however: for instance if you have a table \( h = \{ 'a' => 'x', 'ab' => 'xy' \}, \) You have to carefully avoid generating gsub(/a|ab/, h). This regexp would never match ab. You have to sort the hash key by its length before feeding to Regexp.union.

---

#10 02/05/2018 07:03 AM - Shyouhei (Shyouhei Urabe)

- Has duplicate Feature #14443: Omit 'pattern' parameter in '(g)sb(!)' when 'hash' is given added