Ruby master - Feature #14249

Remove str[match_str]

12/27/2017 01:17 PM - ana06 (Ana Maria Martinez Gomez)

Status:	Open
Priority:	Normal
Assignee:	
Target version:	

Description

I wonder if str[match_str] makes sense.

```
"ana"['a'] => "a"
```

I would say this is not expected and it brings problems, for example when accessing nested hashes. For example:

```
params = { "user" => "Nicolas Cage" } => {"user"=>"Nicolas Cage"}

params["user"]["age"] => "age"
```

I think str[regexp] is enough and that str[match_str] can be removed.

History

#1 - 12/27/2017 02:05 PM - ana06 (Ana Maria Martinez Gomez)

- Backport deleted (2.3: UNKNOWN, 2.4: UNKNOWN, 2.5: UNKNOWN)

- ruby -v deleted (2.4)

- Tracker changed from Bug to Feature

#2 - 12/27/2017 04:12 PM - naruse (Yui NARUSE)

Use Hash#dig instead.

#3 - 05/11/2018 09:57 AM - ana06 (Ana Maria Martinez Gomez)

My example in the first comment is not really correct (something went wrong when copying). What I meant:

```
params = { "user" => { "name" => "Nicolas Cage", "age" => 27 } }

params["user"]["age"] #=> 27

params = { "user" => "Nicolas Cage" }

params["user"]["age"] #=> "age"
```

naruse (Yui NARUSE) using dig works nicely for this case, but then the question is if it makes sense to keep str[match_str]. It is a confusing method...

#4 - 05/11/2018 10:20 AM - Hanmac (Hans Mackowiak)

I can tell you that the str[match_str] is used in many codes, but i can't find any example now

and no, it is different from str[regex] !

#5 - 05/13/2018 11:32 PM - yugui (Yuki Sonoda)

IIUC, Ana is saying that str[match_str] does not make sense because it always returns match_str.
But it does make sense because it returns nil if str does not contain such a substring. i.e. it is useful to test if the string contains a substring.

You would say that str[regex] can do the same thing. And theoretically it is true. But it also means you must compile match_str into Regexp beforehand.
So str[match_str] is a convenient (and sometimes more efficient) way to test substrings.

Then, you might say that str.contains can do the same thing. And theoretically it is true. Here str[match_str] is necessary to just keep compatibility between String and Regexp.
Since most of all methods in the standard library which expect a pattern of character sequence accepts both of String and Regexp, it is natural for users to expect that the compatibility is kept in String#[] too.

#6 - 05/14/2018 10:57 AM - Eregon (Benoit Daloze)

I believe "abc".include?("ab") is much clearer than "abc"["ab"].
Are there real usages of it?
I guess the main concern here is compatibility.

#7 - 05/14/2018 01:54 PM - gotoken (Kentaro Goto)

A use in replacement may look intuitive.

```
s = "Cage"
s["age"] = "heese"
p s
#=> "Cheese"
```

#8 - 05/19/2018 06:13 PM - shevegen (Robert A. Heiler)

As Kentaro showed, there are use cases for [], as-in replacement.

It is quite concise so I think it may be useful for some ruby hackers.

I personally use the more verbose variants usually; as Benoit showed, ".include?" checks, and then often .tr() or .sub() or .gsub(). It is not as concise as Kentaro's example, but I like working with .sub() and .gsub() and .include?() so I tend to use them when possible.

I also use Regexes a lot; they are extremely useful.

Of course in the above, a regex can be used and I think there is more ruby code that uses regexes instead:

```
s[/age/] = "heese"
```

Mostly because, I think, because it will be more flexible, since you can check for more substring-replacements, and it sort of combines the .include? check as well, since the replacement is only done when there is a match to the regex. Regexes are absolutely awesome in ruby.

I don't know offhand about differences as hanmac stated, but I personally use regexes almost always, so the // syntax. But Yuki also mentioned a difference, e. g. the String having to become a regex (and I remember in the old pickaxe, that creating a regex is more costly than creating a String).

Since some time we can use regexes for .start_with?() and .end_with?() checks; before that, I think we could use only Strings.

In .include?() checks, we seem to be only able to use Strings. Would adding regex support there be good? I don't want to distract from the issue here, so I am just mentioning it.

Anyway, Ana wrote:

> the question is if it makes sense to keep str[match_str]. It is a confusing method...

I don't really think it is a confusing method - it seems very simple to me. I don't use it myself because of alternatives, but I think trying to want to make ruby "perfect" in every theoretical aspect, is very, very difficult, even without backwards compatibility.

Ruby is a great language but that does not necessarily mean that 100% of ruby is perfect (or great). Ruby hackers have to decide on what they want to write, how to write it and so forth.

What works in my case is to focus (and use) what I like in ruby and mostly ignore the rest. (I am not speaking about string[match_string] but e. g. @@class_variables as example; I realized one day that I do not need it, so I stopped using them and everything is fine).

Perhaps that also works for your use cases when writing ruby code Ana.