Ruby trunk - Feature #1873

MatchData#[]: Omits All But Last Captures Corresponding to the Same Named Group

08/04/2009 07:39 AM - runpaint (Run Paint Run Run)

Status:	Rejected	
Priority:	Normal	
Assignee:	naruse (Yui NARUSE)	
Target version:	2.6	

Description

=begin
I suspect that MatchData#[:symbol] should return an Array of values when the same named group has been matched multiple times.

```
m = 'food'.match(/(?oo)(?d)/)
=> #
m[:f]
=> "d"
m.to_a
=> ["ood", "oo", "d"]
=end
```

History

#1 - 08/04/2009 12:51 PM - naruse (Yui NARUSE)

=begin
This request is in other words, enable capture history.

A-5. Disabled functions by default syntax
+ capture history
(?@...) and (?@...)
ex. /(?@a)*/.match("aaa") ==> [, ,]
see sample/listcap.c file.
http://www.geocities.jp/kosako3/oniguruma/doc/RE.txt
=end

#2 - 08/04/2009 06:40 PM - runpaint (Run Paint Run Run)

=begin

> This request is in other words, enable capture history.

I'm not sure I understand. The example shows that the history is already captured. Both the #inspect output and #to_a shows that the data is already stored inside the MatchData object; it's just not accessible with the #[Symbol] accessor.

IOW:

```
'abc'.match(/(?a)(?b)(?c)/)
=> #
```

Group 'a' is shown as having the values 'a', 'b', and 'c', but #[:a] only returns 'c':

```
'abc'.match(/(?a)(?b)(?c)/)[:a]
=> "c"
'abc'.match(/(?a)(?b)(?c)/).captures
=> ["a", "b", "c"]
```

This doesn't seem like a disabled function to me; just an assumption that each group name will only appear once.
=end

#3 - 08/05/2009 02:09 AM - naruse (Yui NARUSE)

=begin
Yes, of cource, your desiable API needs more implementation after the option enabled.

I said by the comment before, Ruby 1.9's regexp is based on Oniguruma. So if a function is already implemented in Oniguruma, the function is more easy to realize than functions which are not implemented in Oniguruma.
=end

#4 - 08/05/2009 02:33 AM - runpaint (Run Paint Run Run)

=begin
Yui,

Thank you for your help. I didn't realise it would be difficult. I'd assumed that as:

/(?a)(?b)(?c)/.named_captures
=> {"a"=>[1, 2, 3]}

And:

'abc'.match(/(?a)(?b)(?c)/)[1..3]
=> ["a", "b", "c"]

It would be simply be a matter of mapping one to the other.

How about then if we set this ticket's priority to 'Low', then add a note to the documentation of MatchData#[] that explains this quirk? :-)

(As an aside that Oniguruma document would make a great start for the RDoc of Regexp. Currently the actual syntax of the patterns doesn't seem to appear anywhere in ri.)
=end

#5 - 08/05/2009 03:33 AM - naruse (Yui NARUSE)

=begin
Oh sorry, i misunderstood. I think you wanted to access "b" with /(?\w)+/.match("abc").

Your proposal may be able to implement because those data is near by us. (you taught it by inspect data, sorry!)

Anyway however, changing return value from alwasy String to String or Array is difficult because of compatibility.
If you want access another matched string, suggest to add a new API to access them.
So what is the desirable API is the problem.
=end

#6 - 08/05/2009 03:17 PM - runpaint (Run Paint Run Run)

=begin

> Oh sorry, i misunderstood. I think you wanted to access "b" with /(?\w)+/.match("abc").

That's quite alright. :-)

> Anyway however, changing return value from alwasy String to String or Array is difficult because of compatibility.

Hmmm... That's unfortunate MatchData#[] is a lovely API. I guess one approach is to return an Array when there are multiple matches; a String otherwise. Anybody who currently relies on only the last match being returned is both in the minority and taking advantage of a bug. But I confess not to being overly fond of this solution. :-/ I'm not sure.
=end

#7 - 08/05/2009 03:59 PM - naruse (Yui NARUSE)

=begin

> one approach is to return an Array when there are multiple matches; a String otherwise

There are few APIs which returns different types except true/false or obj/nil.
This is because such API disturbs duck typing.

So new API which returns always Array seems the way.
=end

#8 - 08/05/2009 04:20 PM - runpaint (Run Paint Run Run)

=begin
I think adding another method of this form to MatchData will be confusing. How about overloading MatchData#values_at ? It currently takes one or more integer indices and returns an Array of corresponding values. It could be modified to take a list of Symbols (and Strings, if it must) and return an Array of the matches. This is backward compatible, requires no new methods, and uses the same principle as MatchData#[], which previously only accepted Integer arguments, and now accepts Symbols/Strings, too.

```
>> 'haystack'.match(/(?<h>ay).+(?<h>ack)/).values_at(:h)
['ay','ack']
>> 'haystack'.match(/(?<h>ay).+(?<h>a(?<seek>ck))/).values_at(:seek)
['ck']
>> 'haystack'.match(/(?<h>ay).+(?<h>a(?<seek>ck))/).values_at(:seek, 'h')
['ck','ay','ack']
```

=end

#9 - 08/16/2009 11:12 PM - erikh (Erik Hollensbe)

- File re_named_values_at.patch.gz added

=begin
Attached is a patch which implements the overloaded #values_at functionality. If there is a problem, let me know and I'll alter it. Test cases and docs included.

For now, any information (be it named capture, index, or unexpected type) that doesn't yield information is effectively a no-op, is ignored and no result appears in the array. This is the behavior I noticed in other areas of the MatchData class and figured it was safest to honor that.
=end

#10 - 08/19/2009 05:20 AM - runpaint (Run Paint Run Run)

=begin
Thanks, Erik. I tried the patch out and it works well. :-)
=end

#11 - 03/20/2010 03:07 AM - mame (Yusuke Endoh)

=begin
Hi,

> How about overloading MatchData#values_at ?

Why do you attempt to reuse existing method? :-/
I think it is better to introduce new method like MatchData#all_values.

> > 'haystack'.match(/(?ay).+(?a(?ck))/).values_at(:seek, 'h')
> > ['ck','ay','ack']

I expect values_at returns an array whose length is equal to the number
of arguments.

By the way, I think it is strange (or even a bug) for MatchData#values_at
to reject Symbols.

--
Yusuke ENDOH mame@tsg.ne.jp
=end

#12 - 04/02/2010 08:23 AM - znz (Kazuhiro NISHIYAMA)

- Target version set to 2.0.0

=begin

=end

#13 - 03/18/2012 02:39 PM - nahi (Hiroshi Nakamura)

- Description updated

- Assignee set to naruse (Yui NARUSE)

#14 - 03/18/2012 03:01 PM - naruse (Yui NARUSE)

- Status changed from Open to Feedback

This feature itself is acceptable, but proposed method name (API) is not acceptable.

#15 - 03/18/2012 09:31 PM - trans (Thomas Sawyer)

This is the first time I've seen regular expression groups, so it's interesting.

It occurs to me that with this addition MatchData is both a sort of Array and a sort of Hash. That being so consider md.to_h.

#16 - 04/11/2012 06:34 PM - erikh (Erik Hollensbe)

- File re_all_values.patch.gz added

I've attached a new patch -- this implements the same functionality but refers to it as all_values and reverts the old changes to values_at. This is fundamentally the same functionality as values_at with the overridden functionality described in the ticket.

Sorry for the latency on this, it's been a crazy few years. :)

Tests pass, including the new ones.

#17 - 05/26/2012 06:48 AM - Anonymous

Hi everyone. I am a newbie user of computer languages (< 1 year), and I am
providing my feedback from this position.

Summary:
I find that this feature proposal is basically an extension of Regex state
machine functionality. I am against it. I think, that the current behavior
is natural: When one uses the same capture group name again, the old value
is lost, just like when one assigns a new value to the same variable name.
In Regex machine, I value simplicity and memorizeability over abundance of
features. Moreover, as runpaint points out himself, this feature is really
not missing: "lost" captures are available via #to_a and #capture methods.

Rationale:
As a newbie, I still remember hard time that I had learning Regex. I find
the learning overhead for Ruby acceptable to make it usefull as a tool for
people, who are not programmers by profession. But I found that to actually
solve even simple domain-specific programming tasks, one has to also learn
seemingly endless list of formats, sub-standards, sub-languages, programers'
editors and other idiosyncrasies, which in total take much greater effort
that learning Ruby itself. Regex is one of these sub-languages.

The idea of a simple state machine that performs matching tasks far beyond
find & replace comes as very natural to me. I might have learned Regex in
1 hour in good old days when it only had 25 features. But today, Regex has
(lemme count) 8 anchors, 9 character classes, 8 assertions, 10 quantifiers,
9 backreferences, 10 range syntaxes, 7 pattern modifiers, 14 metacharacters,
passive/active, greedy/ungreedy concept, in total, roughly 75 symbols and
concepts to confuse one's head. Old dog programmers who have been with Regex
throughout its history might not be noticing this, but for newbies, who have
to memorize whole Regex machine in one fell swoop, is is already very hard
to learn. I think that with the number of features that Regex already has,
adding more causes disproportionate growth in learning overhead for newbies.

#18 - 07/09/2012 11:50 AM - erikh (Erik Hollensbe)

Hi folks, can I get some feedback on this patch before feature freeze? Thanks.

#19 - 07/09/2012 06:41 PM - naruse (Yui NARUSE)

erikh (Erik Hollensbe) wrote:

> I've attached a new patch -- this implements the same functionality but refers to it as all_values and reverts the old changes to values_at. This is fundamentally the same functionality as values_at with the overridden functionality described in the ticket.

I don't think all_values is a good name.
Your implementation is good for making the feature clear.

mame says

> I expect values_at returns an array whose length is equal to the number of arguments.

I agree this.
This depends usual question; what is the use case?

> Tests pass, including the new ones.

Adding tests is good contribution, but this doesn't cover the feature.
Anyway, how it should behave depends on use case.
I'm considering more simple API to get the array of strings which are captured by a group,
but it also needs the use case.

boris_stitnicky (Boris Stitnicky) wrote:

> Hi everyone. I am a newbie user of computer languages (< 1 year), and I am
> providing my feedback from this position.

Regexp is not for newbies, but for well-trained programers.

> Summary:
> I find that this feature proposal is basically an extension of Regex state
> machine functionality. I am against it. I think, that the current behavior
> is natural: When one uses the same capture group name again, the old value
> is lost, just like when one assigns a new value to the same variable name.
> In Regex machine, I value simplicity and memorizeability over abundance of
> features. Moreover, as runpaint points out himself, this feature is really
> not missing: "lost" captures are available via #to_a and #capture methods.

As you can see through the patch, Oniguruma, the regexp engine of Ruby 1.9,
doesn't lost the old value.

> Rationale:
> As a newbie, I still remember hard time that I had learning Regex. I find
> the learning overhead for Ruby acceptable to make it usefull as a tool for
> people, who are not programmers by profession. But I found that to actually
> solve even simple domain-specific programming tasks, one has to also learn
> seemingly endless list of formats, sub-standards, sub-languages, programers'
> editors and other idiosyncrasies, which in total take much greater effort
> that learning Ruby itself. Regex is one of these sub-languages.

Ruby won't reject a new feature because it is difficult for a newbie,
because Ruby belives a newbie shall be a professional.
Ruby won't barrier the growth of programmers.

> The idea of a simple state machine that performs matching tasks far beyond
> find & replace comes as very natural to me. I might have learned Regex in
> 1 hour in good old days when it only had 25 features. But today, Regex has
> (lemme count) 8 anchors, 9 character classes, 8 assertions, 10 quantifiers,
> 9 backreferences, 10 range syntaxes, 7 pattern modifiers, 14 metacharacters,
> passive/active, greedy/ungreedy concept, in total, roughly 75 symbols and
> concepts to confuse one's head. Old dog programmers who have been with Regex
> throughout its history might not be noticing this, but for newbies, who have
> to memorize whole Regex machine in one fell swoop, is is already very hard
> to learn. I think that with the number of features that Regex already has,
> adding more causes disproportionate growth in learning overhead for newbies.

You haven't see the delight of regular expression.

#20 - 10/27/2012 04:49 AM - ko1 (Koichi Sasada)

- Target version changed from 2.0.0 to 2.6

I changed the target to "next minor" because no discussion there.

#21 - 10/20/2017 01:27 AM - mame (Yusuke Endoh)

- Status changed from Feedback to Rejected

The discussion has been stalled. It seems the feature itself looks good (note that matz has not said his opinion, thoguh). Anyway, the name matters. Please reopen or create a new ticket if you come up with a good name for the method.

Files

re_named_values_at.patch.gz	1015 Bytes	08/16/2009	erikh (Erik Hollensbe)
re_all_values.patch.gz	1.39 KB	04/11/2012	erikh (Erik Hollensbe)