What is a non-capturing group? What does (?:) do?

Ask Question

1521

466

How `?:` is used and what it's good for?

regex capturing-group regex-group

edited Jan 2 '18 at 1:06

buræquete

5,557 4 21 51

asked Aug 18 '10 at 13:17

never_had_a_name

33.5k 87 246 366

34 This question has been added to the Stack Overflow Regular Expression FAQ, under "Groups". – aliteralmind Apr 10 '14 at 0:25

15 Answers

Let me try to explain this with an example.

Consider the following text:

```
http://stackoverflow.com/
https://stackoverflow.com/questions/tagged/regex
```

Now, if I apply the regex below over it...

```
(https?|ftp)://([^/\r\n]+)(/[^\r\n]*)?
```

... I would get the following result:

```
Match "http://stackoverflow.com/"
     Group 1: "http"
     Group 2: "stackoverflow.com"
     Group 3: "/"

Match "https://stackoverflow.com/questions/tagged/regex"
     Group 1: "https"
     Group 2: "stackoverflow.com"
     Group 3: "/questions/tagged/regex"
```

But I don't care about the protocol -- I just want the host and path of the URL. So, I change the regex to include the non-capturing group `(?:)`.

```
(?:https?|ftp)://([^/\r\n]+)(/[^\r\n]*)?
```

Now, my result looks like this:

```
Match "http://stackoverflow.com/"
     Group 1: "stackoverflow.com"
     Group 2: "/"

Match "https://stackoverflow.com/questions/tagged/regex"
     Group 1: "stackoverflow.com"
     Group 2: "/questions/tagged/regex"
```

See? The first group has not been captured. The parser uses it to match the text, but ignores it later, in the final

EDIT:

As requested, let me try to explain groups too.

Well, groups serve many purposes. They can help you to extract exact information from a bigger match (which can also be named), they let you rematch a previous matched group, and can be used for substitutions. Let's try some examples, shall we?

Ok, imagine you have some kind of XML or HTML (be aware that regex may not be the best tool for the job, but it is nice as an example). You want to parse the tags, so you could do something like this (I have added spaces to make it easier to understand):

```
   \<(?<TAG>.+?)\> [^<]*? \</\k<TAG>\>
or
   \<(.+?)\> [^<]*? \</\1\>
```

The first regex has a named group (TAG), while the second one uses a common group. Both regexes do the same thing: they use the value from the first group (the name of the tag) to match the closing tag. The difference is that the first one uses the name to match the value, and the second one uses the group index (which starts at 1).

Let's try some substitutions now. Consider the following text:

```
Lorem ipsum dolor sit amet consectetuer feugiat fames mal
```

Now, let's use this dumb regex over it:

```
\b(\S)(\S)(\S)(\S*)\b
```

This regex matches words with at least 3 characters, and uses groups to separate the first three letters. The result is this:

```
Match "Lorem"
     Group 1: "L"
     Group 2: "o"
     Group 3: "r"
     Group 4: "em"
Match "ipsum"
     Group 1: "i"
     Group 2: "p"
     Group 3: "s"
     Group 4: "um"
...

Match "consectetuer"
     Group 1: "c"
     Group 2: "o"
     Group 3: "n"
     Group 4: "sectetuer"
...
```

So, if we apply the substitution string:

```
$1_$3$2_$4
```

... over it, we are trying to use the first group, add an underscore, use the third group, then the second group, add another underscore, and then the fourth group. The resulting string would be like the one below.

```
L_ro_em i_sp_um d_lo_or s_ti_ a_em_t c_no_sectetuer f_ue_
p_er_tium e_eg_stas.
```

You can use named groups for substitutions too, using `${name}` .

To play around with regexes, I recommend

details on how the regex works; it also offers a few regex engines to choose from.

edited Mar 19 at 12:47

Mohit Motwani
2,606 1 7 26

answered Aug 18 '10 at 15:39

Ricardo Nolde
22.9k 2 24 33

1 @ajsie: Traditional (capturing) groups are most useful if you're performing a replacement operation on the results. Here's an example where I'm grabbing comma-separated last & first names and then reversing their order (thanks to named groups)... regexhero.net/tester/?id=16892996-64d4-4f10-860a-24f28dad7e30 – Steve Wortham Aug 19 '10 at 15:43

1 No, it's not the same. – Ricardo Nolde Apr 19 '13 at 13:45

4 Might also point out that non-capturing groups are uniquely useful when using regex as split delimiters: "Alice and Bob"-split"\s+(?:and|or)\s+" – Yevgeniy May 7 '14 at 18:06

5 It would be interesting to have the difference between non-capturing groups (?:), and lookahead and lookbehind assertions (?=, ?!) explained. I just started learning about regular expressions, but from what I understand, non-capturing groups are used for matching and "return" what they match, but that "return value" is not "stored" for back-referencing. Lookahead and lookbehind assertions on the other hand are not only not "stored", they are also not part of a match, they just assert that something would match, but their "match" value is ignored, if I'm not mistaken... (Am I roughly right?) – Christian May 11 '14 at 20:40

4 [] is a set; [123] matches any char inside the set once; [^123] matches anything NOT inside the set once; [^/\r\n]+ matches one or more chars that are different from /, \r, \n. – Ricardo Nolde Jun 5 '14 at 20:18

152

You can use capturing groups to organize and parse an expression. A non-capturing group has the first benefit, but doesn't have the overhead of the second. You can still say a non-capturing group is optional, for example.

Say you want to match numeric text, but some numbers could be written as 1st, 2nd, 3rd, 4th,... If you want to capture the numeric part, but not the (optional) suffix you can use a non-capturing group.

```
([0-9]+)(?:st|nd|rd|th)?
```

That will match numbers in the form 1, 2, 3... or in the form 1st, 2nd, 3rd,... but it will only capture the numeric part.

edited Aug 18 '10 at 13:30

answered Aug 18 '10 at 13:24

Bill the Lizard

299k 159 501 793

`?:` is used when you want to group an expression, but you do not want to save it as a matched/captured portion

An example would be something to match an IP address:

```
/(?:\d{1,3}\.){3}\d{1,3}/
```

Note that I don't care about saving the first 3 octets, but the `(?:...)` grouping allows me to shorten the regex without incurring the overhead of capturing and storing a match.

edited Aug 18 '10 at 13:27

answered Aug 18 '10 at 13:22

RC.
22.9k 7 64 89

32

It makes the group non-capturing, which means that the substring matched by that group will not be included in the list of captures. An example in ruby to illustrate the difference:

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

answered Aug 18 '10 at 13:23

sepp2k
302k 39 603 619

19

parenthesis. Consider the expressions (a|b)c and a|bc, due to priority of concatenation over |, these expressions represent two different languages ({ac, bc} and {a, bc} respectively). However, the parenthesis are also used as a matching group (as explained by the other answers...).

When you want to have parenthesis but not capture the subexpression you use NON-CAPTURING GROUPS. In the example, (?:a|b)c

answered Feb 4 '16 at 8:07

user2369060
271 3 6

3 I was wondering why. As I think the "why" is vital to memorizing this information. – J.M.I. MADISON Aug 4 '18 at 23:59

14

Groups that **capture** you can use later on in the regex to match **OR** you can use them in the replacement part of the regex. Making a **non-capturing** group simply exempts that group from being used for either of these reasons.

Non-capturing groups are great if you are trying to capture many different things and there are some groups you don't want to capture.

Thats pretty much the reason they exist. While you are learning about groups, learn about Atomic Groups, they do a lot! There is also lookaround groups but they are a little more complex and not used so much.

Example of using later on in the regex (backreference):

`<([A-Z][A-Z0-9]*)\b[^>]*>.*?</\1>` [Finds an xml tag

`([A-Z][A-Z0-9]*)` is a capturing group (in this case it is the tagname)

Later on in the regex is `\1` which means it will only match the same text that was in the first group (the `([A-Z][A-Z0-9]*)` group) (in this case it is matching the end tag).

edited Aug 18 '10 at 13:33

answered Aug 18 '10 at 13:22

Bob Fincheimer
14.9k 1 21 52

could you give a simple example of how it will be used later to match OR? – never_had_a_name Aug 18 '10 at 13:27

i mean you can use to to match later or you can use it in the replacement. The or in that sentence was just to show you there are two uses for a capturing group – Bob Fincheimer Aug 18 '10 at 13:33

11

Let me try this with an example :-

Regex Code :- `(?:animal)(?:=)(\w+)(,)\1\2`

Search String :-

Line 1 - `animal=cat,dog,cat,tiger,dog`

Line 2 - `animal=cat,cat,dog,dog,tiger`

Line 3 - `animal=dog,dog,cat,cat,tiger`

`(?:animal)` --> Non-Captured Group 1

`(\w+)` --> Captured Group 1

`(,)` --> Captured Group 2

`\1` --> result of captured group 1 i.e In Line 1 is cat,In Line 2 is cat,In Line 3 is dog.

`\2` -->result of captured group 2 i.e comma(,)

So in this code by giving \1 and \2 we recall or repeat the result of captured group 1 and 2 respectively later in the code.

As per the order of code (?:animal) should be group 1 and (?:=) should be group 2 and continues..

but by giving the ?: we make the match-group non captured(which do not count off in matched group, so the grouping number starts from the first captured group and not the non captured), so that the repetition of the result of match-group (?:animal) can't be called later in code.

Hope this explains the use of non capturing group.

enter image description here

edited Sep 14 '18 at 6:53

AkshayNevrekar
6,573 10 22 43

answered Jan 19 '17 at 11:36

shekhar gehlot
111 1 3

great and simple explanation! – Teena George Oct 11 '18 at 3:15

7

arise where you wish to use a large number of groups some of which are there for repetition matching and some of which are there to provide back references. By default the text matching each group is loaded into the backreference array. Where we have lots of groups and only need to be able to reference some of them from the backreference array we can override this default behaviour to tell the regular expression that certain groups are there only for repetition handling and do not need to be captured and stored in the backreference array.

answered Mar 8 '14 at 17:33

Jack Peng
160 1 12

7

Well I am a JavaScript developer and will try to explain its significance pertaining to JavaScript.

Consider a scenario where you want to match `cat is animal` when you would like match cat and animal and both should have a `is` in between them.

```
 // this will ignore "is" as that's is what we want
"cat is animal".match(/(cat)(?: is )(animal)/) ;
result ["cat is animal", "cat", "animal"]

 // using lookahead pattern it will match only "cat" we ca
 // use lookahead but the problem is we can not give anyth
 // at the back of lookahead pattern
"cat is animal".match(/cat(?= is animal)/) ;
result ["cat"]

 //so I gave another grouping parenthesis for animal
 // in lookahead pattern to match animal as well
"cat is animal".match(/(cat)(?= is (animal))/) ;
result ["cat", "cat", "animal"]

 // we got extra cat in above example so removing another
```

edited Mar 1 '16 at 10:00

Alan Moore

62k 9 80 135

answered Mar 1 '16 at 9:43

Gaurav

543 7 7

6

tl;dr non-capturing groups, as the name suggests are the parts of the regex that you do not want to be included in the match and `?:` is a way to define a group as being non-capturing.

Let's say you have an email address `example@example.com` . The following regex will create two **groups**, the id part and @example.com part. `(\p{Alpha}*[a-z])(@example.com)` . For simplicity's sake, we are extracting the whole domain name including the `@` character.

Now let's say, you only need the id part of the address. What you want to do is to grab the first group of the match result, surrounded by `()` in the regex and the way to do this is to use the non-capturing group syntax, i.e. `?:` . So the regex `(\p{Alpha}*[a-z])(?:@example.com)` will return just the id part of the email.

answered May 11 '18 at 5:27

6pack kid

1,989 2 22 25

One interesting thing that I came across is the fact that you

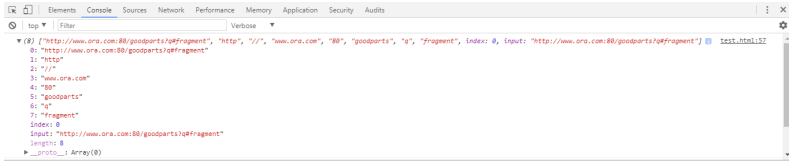
```
var parse_url_regex = /^(?:([A-Za-z]+):)(\/{0,3})([0-9.\-A
#]*))?(?:\?([^#]*))?(?:#(.*))?$/;
```

Input url string:

```
var url = "http://www.ora.com:80/goodparts?q#fragment";
```

The first group in my regex `(?:([A-Za-z]+):)` is a non-capturing group which matches the protocol scheme and colon `:` character i.e. `http:` but when I was running below code, I was seeing the 1st index of the returned array was containing the string `http` when I was thinking that `http` and colon `:` both will not get reported as they are inside a non-capturing group.

```
console.debug(parse_url_regex.exec(url));
```



I thought if the first group `(?:([A-Za-z]+):)` is a non-capturing group then why it is returning `http` string in the output array.

So if you notice that there is a nested group `([A-Za-z]+)` inside the non-capturing group. That nested group `([A-Za-z]+)` is a capturing group (not having `?:` at the beginning) in itself inside a non-capturing group `(?:([A-Za-z]+):)`. That's why the text `http` still gets captured but the colon `:` character which is inside the non-capturing group but outside the capturing group doesn't get reported in the output array.

answered Jul 15 '17 at 3:13

RBT
9,630 7 79 107

5

I cannot comment on the top answers to say this: I would like to add an explicit point which is only implied in the top answers:

The non-capturing group `(?...)` does **not remove** any characters from the original full match, **it only** reorganises the regex visually to the programmer.

To access a specific part of the regex without defined extraneous characters you would always need to use `.group(<index>)`

answered Jan 2 '18 at 1:04

Scott Anderson
165 1 12

2 You have provided the most important hint that was missing in the rest of the answers. I tried all the examples in them and using the choicest of expletives, as I did not get the desired result. Only your posting showed me where I went wrong. – Seshadri R Jul 12 '18 at 7:04

Glad to hear it! – Scott Anderson Jul 12 '18 at 12:04

I think I would give you the answer, Don't use capture variables without checking that the match succeeded.

```
#!/usr/bin/perl
use warnings;
use strict;
$_ = "bronto saurus burger";
if (/(?:bronto)? saurus (steak|burger)/)
{
    print "Fred wants a  $1";
}
else
{
    print "Fred dont wants a $1 $2";
}
```

In the above example, To avoid capturing bronto in $1, (?:) is used. If the pattern is matched , then $1 is captured as next grouped pattern. So, the output will be as below:

```
Fred wants a burger
```

It is Useful if you don't want the matches to be saved .

answered May 23 '17 at 13:40

Harini

471 4 13

1

Open your Google Chrome devTools and then Console tab: and type this:

```
"Peace".match(/(\w)(\w)(\w)/)
```

Run it and you will see:

```
["Pea", "P", "e", "a", index: 0, input: "Peace", groups: u
```

The `JavaScript` RegExp engine capture three groups, the items with indexes 1,2,3. Now use non-capturing mark to see the result.

```
"Peace".match(/(?:\w)(\w)(\w)/)
```

The result is:

```
["Pea", "e", "a", index: 0, input: "Peace", groups: undefi
```

This is obvious what is non capturing group.

answered May 7 '18 at 3:50

AmerllicA

1

1

Its extremely simple, We can understand with simple date example, suppose if the date is mentioned as 1st January 2019 or 2nd May 2019 or any other date and we simply want to convert it to **dd/mm/yyyy** format we would not need the month's name which is January or February for that matter, so in order to capture the numeric part, but not the (optional) suffix you can use a non-capturing group.

so the regular expression would be,

```
([0-9]+)(?:January|February)?
```

Its as simple as that.

edited Jan 7 at 8:08

answered Jan 7 at 8:02

Naved Ahmad
61 4

protected by eyllanesc Sep 7 '18 at 6:17

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?