what is the difference between ?:, ?! and ?= in regex?

Ask Question

64

43

I searched for the meaning of these expressions but couldn't understand the exact differnce between them. This is what they say:

- `?:` Match expression but do not capture it.
- `?=` Match a suffix but exclude it from capture.
- `?!` Match if suffix is absent.

I tried using these in simple RegEx and got similar results for all. example: the following 3 expressions give very similar results.

- `[a-zA-Z0-9._-]+@[a-zA-Z0-9-]+(?!\.[a-zA-Z0-9]+)*`
- `[a-zA-Z0-9._-]+@[a-zA-Z0-9-]+(?=\.[a-zA-Z0-9]+)*`
- `[a-zA-Z0-9._-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9]+)*`

javascript regex

edited Sep 10 '15 at 16:42

Alan Moore
60.7k 9 78 132

asked May 29 '12 at 18:33
RK Poddar
615 2 9 18

Please show us your test case. They should not give the same results. – Bergi May 29 '12 at 18:41

@sepp2k, it same similar results in few case, one of them mentioned in the question. – RK Poddar May 29 '12 at 18:41

@Bergi, i tested it with random data, containing english words, phone numbers, urls, e-mail addresses, numbers, etc.. – RK Poddar May 29 '12 at 18:43

4 @RKAgarwal Ah, I see what you did there. You added a `*` after the groups, so they're simply ignored. – sepp2k May 29 '12 at 18:44

noobie note: you'd only use these at the start of parenthesis, and parenthesis form a capturing group (different parenthesis sets extract different sections of text). – Ryan Taylor Jun 7 '17 at 19:06

4 Answers

79

The difference between `?=` and `?!` is that the former requires the given expression to match and the latter requires it to **not** match. For example `a(?=b)` will match the "a" in "ab", but not the "a" in "ac". Whereas `a(?!b)` will match the "a" in "ac", but not the "a" in "ab".

The difference between `?:` and `?=` is that `?=` excludes the expression from the entire match while `?:` just doesn't create a capturing group. So for example `a(?:b)` will match the "ab" in "abc", while `a(?=b)` will only match the "a" in "abc". `a(b)` would match the "ab" in "abc" *and* create a capture containing the "b".

answered May 29 '12 at 18:43
sepp2k
293k 38 593 609

Good description with good examples. – Madushan Perera Nov 25 '15 at 3:18

54

```
?:  is for non capturi
?=  is for positive loo
?!  is for negative loo
?<= is for positive loo
?<! is for negative loo
```

Please check here: http://www.regular-expressions.info/lookaround.html for very good tutorial and examples on lookahead in regular expressions.

answered May 29 '12 at 18:38

anubhava
521k 46 316 390

12 Yet JavaScript does not know lookbehind. – Bergi May 29 '12 at 18:45

1 This one is more complete for general regex. – Yan Yang Jan 25 '18 at 0:37

/(?<=^a)b/ worked for me in javascript! There seems to be no tutorial for looking behind in Javascript on the internet. – Y. Yoshii May 16 '18 at 6:58

Only recent versions of browsers have started supporting look behind in JS – anubhava May 16 '18 at 7:10

– anubhava I don't know any alternative to /(?<=^a)b/ using the pure regular expression. Perhaps I can but I would have to rely on callback functions. – Y. Yoshii May 16 '18 at 7:17

8

To better understand let's apply the three expressions plus a capturing group and analyze each behavior.

- `()` **capturing group -** the regex inside the parenthesis must be matched and the match create a capturing group
- `(?:)` **non capturing group** - the regex inside the parenthesis must be matched but doesn't not create the capturing group
- `(?=)` **positive look ahead** - asserts that the regex must be matched
- `(?!)` **negative look ahead** - asserts that it is impossible to match the regex

Let's apply `q(u)i` to *quit*. `q` matches *q* and the capturing group `u` matches *u*. The match inside the capturing group is taken and a capturing group is created. So the engine continues with `i`. And `i` will match *i*. This last match attempt is successful. *qui* is matched and a capturing group with *u* is created.

Let's apply `q(?:u)i` to *quit*. Again, `q` matches *q* and the non-capturing group `u` matches *u*. The match from the non-capturing group is taken, but the capturing group is not created. So the engine continues with `i`. And `i` will match *i*. This last match attempt is successful. *qui* is matched

Let's apply `q(?=u)i` to *quit*. The lookahead is positive and is followed by another token. Again, `q` matches *q* and `u` matches *u*. Again, the match from the lookahead must be discarded, so the engine steps back from `i` in the string to *u*. The lookahead was successful, so the engine continues with `i`. But `i` cannot match *u*. So this match attempt fails.

Let's apply `q(?=u)u` to *quit*. The lookahead is positive and is followed by another token. Again, `q` matches *q* and `u` matches *u*. The match from the lookahead must be discarded, so the engine steps back from `u` in the string to *u*. The lookahead was successful, so the engine continues with `u`. And `u` will match *u*. So this match attempt is successful. *qu* is matched

Let's apply `q(?!i)u` to *quit*. Even in this case lookahead is positive (because `i` does not match) and is followed by another token.

Again, `q` matches *q* and `i` doesn't matches *u*. The match from the lookahead must be discarded, so the engine steps back from `u` in the string to *u*. The lookahead was successful, so the engine continues with `u`. And `u` will match *u*. So this match attempt is successful. *qu* is matched

So, in conclusion, the real difference between lookahead and non-capturing groups it is all about if you want just test the existence or test and save the match. Capturing group are expensive so use it judiciously.

edited Jun 7 '17 at 23:35

answered Aug 15 '16 at 22:20

freedev
11.2k 4 46 68

5

Try matching `foobar` against these:

```
/foo(?=b)(.*)/
/foo(?!b)(.*)/
```

The first regex will match and will return "bar" as first submatch — `(?=b)` matches the 'b', but does not consume it, leaving it for the following parentheses.

The second regex will NOT match, because it expects "foo" to be followed by something different from 'b'.

`(?:...)` has exactly the same effect as simple `(...)`, but it does not return that portion as a submatch.

edited Dec 21 '17 at 21:00

Abdullah
1,271 11 23

answered May 29 '12 at 18:42

lanzz
32k 7 65 87