How can I match a quote-delimited string with a regex?

Asked 16 years ago Modified 8 years, 6 months ago Viewed 124k times

47

If I'm trying to match a quote-delimited string with a regex, which of the following is "better" (where "better" means both more efficient and less likely to do something unexpected):

```
/"[^"]+"/ # match quote, then everything that's not a quote, then a quote
```

or

```
/".+?"/   # match quote, then *anything* (non-greedy), then a quote
```

Assume for this question that empty strings (i.e. "") are not an issue. It seems to me (no regex newbie, but certainly no expert) that these will be equivalent.

Update: Upon reflection, I think changing the `+` characters to `*` will handle empty strings correctly anyway.

regex perl

Share
Improve this question
Follow

edited Dec 29, 2008 at 23:39

brian d foy
132k ● 31 ● 211 ● 604

asked Dec 17, 2008 at 16:19

Graeme Perrow
57.2k ● 24 ● 86 ● 125

9 Answers

Sorted by: Highest score (default)

46

You should use number one, because number two is bad practice. Consider that the developer who comes after you wants to match strings that are followed by an exclamation point. Should he use:

```
"[^"]*"!
```

or:

```
".*?"!
```

The difference appears when you have the subject:

```
"one" "two"!
```

The first regex matches:

```
"two"!
```

while the second regex matches:

```
"one" "two"!
```

Always be as specific as you can. Use the negated character class when you can.

Another difference is that [^"]* can span across lines, while .* doesn't unless you use single line mode. [^"\n]* excludes the line breaks too.

As for backtracking, the second regex backtracks for each and every character in every string that it matches. If the closing quote is missing, both regexes will backtrack through the entire file. Only the order in which then backtrack is different. Thus, in theory, the first regex is faster. In practice, you won't notice the difference.

Share

Improve this answer

Follow

edited Jun 24, 2016 at 9:51

Etienne Neveu

12.7k ● 9 ● 37 ● 59

answered Dec 18, 2008 at 10:45

Jan Goyvaerts

22k ● 7 ● 62 ● 72

1 This counterexample is exactly what I was looking for when I wrote the question. Thanks Jan – Graeme Perrow Dec 18, 2008 at 20:24

26

More complicated, but it handles escaped quotes and also escaped backslashes (escaped backslashes followed by a quote is not a problem)

```
/(["'])((\\{2})*|(.*?[^\\](\\{2})*))\1/
```

Examples:

"hello\"world" matches **"hello\"world"**

"hello\\"world" matches **"hello\\"**

Share Improve this answer Follow

answered Jun 21, 2013 at 9:17

nico

261 ● 3 ● 2

how would you then remove / fix the \\? or any of the backslash protected characters from the resultant string? – kdubs Feb 26, 2020 at 21:37

14

I would suggest:

```
([\"'])(?:\\\1|.)*?\1
```

But only because it handles escaped quote chars and allows both the ' and " to be the quote char. I would also suggest looking at this article that goes into this problem in depth:

http://blog.stevenlevithan.com/archives/match-quoted-string

However, unless you have a serious performance issue or cannot be sure of embedded quotes, go with the simpler and more readable:

```
/".*?"/
```

I must admit that non-greedy patterns are not the basic Unix-style 'ed' regular expression, but they are getting pretty common. I still am not used to group operators like (?:stuff).

Share

Improve this answer

Follow

edited Dec 17, 2008 at 18:37

answered Dec 17, 2008 at 17:37

Harold Bamford

1,629 ● 1 ● 15 ● 26

What's the "(:?stuff)" you mention supposed to mean? I know "(?:stuff)", but not the other. – Tomalak Dec 17, 2008 at 17:44

In Perl, these are called "Extended Patterns". Check out perldoc.perl.org/perlre.html under "Extended Patterns" (about 1/3 of the way down). In this case, it is just like (stuff) except there is no capture ($1 or \1) involved. – Harold Bamford Dec 17, 2008 at 17:51

It was. I just now fixed it. I didn't understand the original comment (I thought there was some obscure escaping bug in SO). Sorry for the confusion. – Harold Bamford Dec 17, 2008 at 18:38

No worries. ;-) Essentially, you can delete this whole dialog as it has no substance anymore. – Tomalak Dec 17, 2008 at 20:08

6

I'd say the second one is better, because it fails faster when the terminating `"` is missing. The first one will backtrack over the string, a potentially expensive operation. An alternative regexp if you are using perl 5.10 would be `/"[^"]++"/`. It conveys the same meaning as version 1 does, but is as fast as version two.

Share
Improve this answer
Follow

edited Dec 17, 2008 at 16:45

answered Dec 17, 2008 at 16:38

Leon Timmermans
30.2k ● 2 ● 64 ● 110

Why is the second one able to fail faster? – innaM Dec 17, 2008 at 16:46

I added the explanation a few seconds before you asked. It the second one doesn't backtrack. – Leon Timmermans Dec 17, 2008 at 16:48

1 If you want to get really fancy and support escaped quotes in the regexp, you could do this: /"(?:[^"]|(?<!\)(?>\\)*\")++"/. I've explained that one at stackoverflow.com/questions/56554/… – Leon Timmermans Dec 17, 2008 at 17:06

Good point regarding the backtracking. For my "long running loop" example, this would surely cause more hurt than the lazy quantifier. Also good that you mentioned possessive quantifiers. +1 – Tomalak Dec 17, 2008 at 19:37

Leon is wrong about the backtracking. .*? backtracks for each and every character in the string, when the closing " is present. When the closing " is missing, both regexes backtrack. – Jan Goyvaerts Dec 18, 2008 at 10:47

4

I'd go for number two since it's much easier to read. But I'd still like to match empty strings so I would use:

```
/".*?"/
```

Share Improve this answer Follow

answered Dec 17, 2008 at 16:27

PEZ
17k ● 7 ● 47 ● 66

@Graeme Perrow: .*? is the defacto standard of non greedy matching – slf Dec 17, 2008 at 16:35

2

From a performance perspective (extremely heavy, long-running loop over long strings), I could imagine that

```
"[^"]*"
```

is faster than

```
".*?"
```

because the latter would do an additional check for each step: peeking at the next character. The former would be able to mindlessly roll over the string.

As I said, in real-world scenarios this would hardly be noticeable. Therefore I would go with number two (if my current regex flavor supports it, that is) because it is much more readable. Otherwise with number one, of course.

Share Improve this answer Follow

answered Dec 17, 2008 at 17:27

Tomalak
338k ● 68 ● 545 ● 635

2

Using the negated character class prevents matching when the boundary character (doublequotes, in your example) is present elsewhere in the input.

Your example #1:

```
/"[^"]+"/ # match quote, then everything that's not a quote, then a quote
```

matches only the smallest pair of matched quotes -- excellent, and most of the time that's all you'll need. However, if you have nested quotes, and you're interested in the largest pair of matched quotes (or in all the matched quotes), you're in a much more complicated situation.

Luckily Damian Conway is ready with the rescue: Text::Balanced is there for you, if you find that there are multiple matched quote marks. It also has the virtue of matching other paired punctuation, e.g. parentheses.

Share
Improve this answer
Follow

edited Dec 29, 2008 at 18:29

answered Dec 29, 2008 at 18:04

Trochee
813 ● 1 ● 6 ● 8

0

I prefer the first regex, but it's certainly a matter of taste.

The first one might be more efficient?

```
Search for double-quote
add double-quote to group
for each char:
    if double-quote:
        break
```

```
    add to group
add double-quote to group
```

Vs something a bit more complicated involving back-tracking?

Share Improve this answer Follow

answered Dec 17, 2008 at 16:32

Douglas Leeder

53.3k ● 9 ● 99 ● 138

0

Considering that I didn't even know about the "*?" thing until today, and I've been using regular expressions for 20+ years, I'd vote in favour of the first. It certainly makes it clear what you're trying to do - you're trying to match a string that doesn't include quotes.

Share Improve this answer Follow

answered Dec 17, 2008 at 16:40

Paul Tomblin

183k ● 59 ● 323 ● 410

I've been using regular expressions for many years too, and I knew there was a way to do things in a non-greedy way, but didn't realize how easy it was until today. That's why I asked - I find it easier to read (now that I know what it means), so is there a reason I shouldn't use it? – Graeme Perrow Dec 17, 2008 at 16:46

The only limitation is in your regex engine. There's the remote possibility that you are facing one that does not support non-greedy quantifiers. Modern ones generally do. – Tomalak Dec 17, 2008 at 17:39

Some regex engines supports changing the default greediness (making .* be nongreedy and .*? be greedy). In PHP you can use the U regex modifier for this. I have had use of that when scraping html. – PEZ Dec 17, 2008 at 18:05

PEZ: I strongly recomment you use /.*?/ instead of /.*/U Most people will recognize .*? as a lazy quantifier, or at least as something they don't know. A /U tucked away at the end of the regex is easily missed. It's about keeping your code human-readable. – Jan Goyvaerts Dec 19, 2008 at 0:17