Parsing attributes with regex in Perl

Asked 16 years, 4 months ago Modified 15 years, 5 months ago Viewed 4k times

2

Here's a problem I ran into recently. I have attributes strings of the form

```
"x=1 and y=abc and z=c4g and ..."
```

Some attributes have numeric values, some have alpha values, some have mixed, some have dates, etc.

Every string is *supposed* to have "`x=someval and y=anotherval`" at the beginning, but some don't. I have three things I need to do.

1. Validate the strings to be certain that they have `x` and `y`.
2. Actually parse the values for `x` and `y`.
3. Get the rest of the string.

Given the example at the top, this would result in the following variables:

```
$x = 1;
$y = "abc";
$remainder = "z=c4g and ..."
```

My question is: Is there a (reasonably) simple way to parse these *and* validate with a single regular expression? i.e.:

```
if ($str =~ /someexpression/)
{
    $x = $1;
    $y = $2;
    $remainder = $3;
}
```

Note that the string may consist of *only* `x` and `y` attributes. This is a valid string.

I'll post my solution as an answer, but it doesn't meet my single-regex preference.

regex perl

Share Improve this question Follow

asked Aug 14, 2008 at 0:40

Derek Park
46.8k ● 16 ● 59 ● 76

5 Answers

Sorted by: Highest score (default)

3

Assuming you also want to do something with the other name=value pairs this is how I would do it (using Perl version 5.10):

```
use 5.10.0;
use strict;
use warnings;

my %hash;
while(
    $string =~ m{
       (?: ^ | \G )    # start of string or previous match
       \s*

       (?<key>   \w+ ) # word characters
       =
       (?<value> \S+ ) # non spaces

       \s*             # get to the start of the next match
       (?: and )?
    }xgi
){
    $hash{$+{key}} = $+{value};
}

# to make sure that x & y exist
die unless exists $hash{x} and exists $hash{y};
```

On older Perls (at least Perl 5.6);

```
use strict;
use warnings;

my %hash;
while(
    $string =~ m{
       (?: ^ | \G )   # start of string or previous match
       \s*

       ( \w+ ) = ( \S+ )

       \s*            # get to the start of the next match
       (?: and )?
    }xgi
){
    $hash{$1} = $2;
}

# to make sure that x & y exist
die unless exists $hash{x} and exists $hash{y};
```

These have the added benefit of continuing to work if you need to work with more data.

Share

Improve this answer

Follow

edited Jul 15, 2009 at 4:37

answered Aug 21, 2008 at 20:02

Brad Gilbert

34.1k ● 11 ● 79 ● 130

`\G` already matches the start of the string, so you can replace `(?:^|\G)` with `\G`. But a better way is to put `\G` in factor at the beginning and to move the `and` at the beginning: `\G (?: ^ | \s+ and \s+) (\w+) = (\S+)` – Casimir et Hippolyte Apr 11, 2016 at 13:53

1

I'm not the best at regular expressions, but this seems pretty close to what you're looking for:

```
/x=(.+) and y=([^ ]+)( and (.*))?/
```

Except you use $1, $2, and $4. In use:

```
my @strs = ("x=1 and y=abc and z=c4g and w=v4l",
            "x=yes and y=no",
            "z=nox and w=noy");

foreach (@strs) {
    if ($_ =~ /x=(.+) and y=([^ ]+)( and (.*))?/) {
        $x = $1;
        $y = $2;
        $remainder = $4;
        print "x: $x; y: $y; remainder: $remainder\n";
    } else {
        print "Failed.\n";
    }
}
```

Output:

```
x: 1; y: abc; remainder: z=c4g and w=v4l
x: yes; y: no; remainder:
Failed.
```

This of course leaves out plenty of error checking, and I don't know everything about your inputs, but this seems to work.

Share Improve this answer Follow

answered Aug 14, 2008 at 1:56

Rudd Zwolinski
27.5k ● 17 ● 59 ● 60

1

As a fairly simple modification to Rudd's version,

```
/^x=(.+) and y=([^ ]+)(?: and (.*))?/
```

will allow you to use $1, $2 and $3 (the ?: makes it a noncapturing group), and will ensure that the string starts with "x=" rather than allowing a "not_x=" to match

If you have better knowledge of what the x and y values will be, this should be used to tighten the regex further:

```
my @strs = ("x=1 and y=abc and z=c4g and w=v4l",
        "x=yes and y=no",
        "z=nox and w=noy",
        "not-x=nox and y=present",
        "x=yes and w='there is no and y=something arg here'");

foreach (@strs) {
    if ($_ =~ /^x=(.+) and y=([^ ]+)(?: and (.*))?/) {
        $x = $1;
        $y = $2;
        $remainder = $3;
        print "x: {$x}; y: {$y}; remainder: {$remainder}\n";
    } else {
        print "$_ Failed.\n";
    }
}
```

Output:

```
x: {1}; y: {abc}; remainder: {z=c4g and w=v4l}
x: {yes}; y: {no}; remainder: {}
z=nox and w=noy Failed.
not-x=nox and y=present Failed.
x: {yes and w='there is no}; y: {something}; remainder: {}
```

Note that the missing part of the last test is due to the current version of the y test requiring no spaces, if the x test had the same restriction that string would have failed.

Share Improve this answer Follow

answered Aug 17, 2008 at 15:39

Cebjyre

6,622 ● 3 ● 34 ● 58

1

Rudd and Cebjyre have gotten you most of the way there but they both have certain problems:

Rudd suggested:

> /x=(.+) and y=([^]+)(and (.*))?/

Cebjyre modified it to:

> /^x=(.+) and y=([^]+)(?: and (.*))?/

The second version is better because it will not confuse "not_x=foo" with "x=foo" but will accept things such as "x=foo z=bar y=baz" and set $1 = "foo z=bar" which is undesirable.

This is probably what you are looking for:

> /^x=(\w+) and y=(\w+)(?: and (.*))?/

This disallows anything between the x= and y= options, places and allows and optional " and..." which will be in $3

Share Improve this answer Follow

answered Sep 15, 2008 at 15:20
Frosty
6,443 ● 3 ● 26 ● 20

0

Here's basically what I did to solve this:

```
($x_str, $y_str, $remainder) = split(/ and /, $str, 3);

if ($x_str !~ /x=(.*)/)
{
    # error
}

$x = $1;

if ($y_str !~ /y=(.*)/)
{
    # error
}

$y = $1;
```

I've omitted some additional validation and error handling. This technique works, but it's not as concise or pretty as I would have liked. I'm hoping someone will have a better suggestion for me.

Share Improve this answer Follow

answered Aug 14, 2008 at 0:46
Derek Park
46.8k ● 16 ● 59 ● 76

This looks to me simpler and more maintainable than any of the "one regexp to rule them all" solutions. I would maybe just add a ^ at the beginning of theregexps to match x= and y= to avoid the case not_x=... or similar. Why do you want a single regexp? – mirod Jul 15, 2009 at 9:01