RegEx match open tags except XHTML self-contained tags

Asked 15 years, 1 month ago Modified 17 days ago Viewed 3.9m times

2298

Locked. Comments on this question have been disabled, but it is still accepting new answers and other interactions. Learn more.

I need to match all of these opening tags:

```
<p>
<a href="foo">
```

But not self-closing tags:

```
<br />
<hr class="foo" />
```

I came up with this and wanted to make sure I've got it right. I am only capturing the `a-z`.

```
<([a-z]+) *[^/]*?>
```

I believe it says:

- Find a less-than, then
- Find (and capture) a-z one or more times, then
- Find zero or more spaces, then
- Find any character zero or more times, greedy, except `/`, then
- Find a greater-than

Do I have that right? And more importantly, what do you think?

html regex xhtml

Share

Improve this question

edited Jan 22 at 9:15

community wiki
12 revs, 8 users 58%
Jeff

Follow

Comments disabled on deleted / locked posts / reviews

36 Answers

Sorted by: Highest score (default)

1 2 Next

4397

Locked. There are disputes about this answer’s content being resolved at this time. It is not currently accepting new interactions.

You can't parse [X]HTML with regex. Because HTML can't be parsed by regex. Regex is not a tool that can be used to correctly parse HTML. As I have answered in HTML-and-regex questions here so many times before, the use of regex will not allow you to consume HTML. Regular expressions are a tool that is insufficiently sophisticated to understand the constructs employed by HTML. HTML is not a regular language and hence cannot be parsed by regular expressions. Regex queries are not equipped to break down HTML into its meaningful parts. so many times but it is not getting to me. Even enhanced irregular regular expressions as used by Perl are not up to the task of parsing HTML. You will never make me crack. HTML is a language of sufficient complexity that it cannot be parsed by regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Parsing HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The <center> cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like so much watery putty. If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane, he comes. HTML-plus-regexp will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Rege̿̔x-based HTML parsers are the cancer that is killing StackOverflow *it is too late it is too late we cannot be saved* the transgression of a chi͡ld ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) *dear lord help us how can anyone survive this scourge* using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes *using regex* as a tool to process HTML establishes a brea*ch between this world* and the dread realm of c͒ͪo͛ͫrrupt entities (like SGML entities, but *more corrupt) a mere glimp*se of the world of rege**x parsers for HTML will ins**tantly transport a p*rogrammer's consciousness i*nto a wor*ld* of ceaseless screaming, he comes~~, the pestilent sl~~ithy regex-infection wil**l devour your HT**ML parser, application and existence for all time like Visual Basic only worse *he comes he comes do not fi*ght h**e com̡e̶s, ̕h̵i**s un̨ho͞ly radiańcé destro҉ying all enli̍̈́̂̈́ghtenment, HTML tags **lea͠ki̧n͘g fr̶ǫm ̡yo͟ur eye͢s̸ ̛l̕ik͏e liq**uid pain, the song of re̸gular expre~~ssion parsing~~ will ext*inguish the voices of mor**tal man from the sp**here I can see it can you see ̲͚̖͔̙î̩́t̲͎̩̱͔́̋̀ it is beautiful t*he f `inal snuf` fing *of the lie**s of Man ALL IS LOŚ͖̩͇̗̪̏̈́T ALL I*S L**O**ST th*e pon̷y he comes he c̶̮omes ~~he co~~**mes t*he ich**or permeat*es al*l MY FAC*E MY FACE ᵒh god n**o NO NOO̼**O**O NΘ stop t*he an*̶͑̾̾̅ͫ͏̙̤g͇̫͛͆̾ͫ̑͆l͖͉̗̩̳̟̍ͫͥͨe̠̅s `͎a̧͈͖r̽̾̈́͒͑e` n**ot rè̑ͧ̌aͨl̘̝̙̃ͤ͂̾̆ ZA̡͊͠͝LGΌ ISͮ̂҉̯͈͕̹̘̱ TO͇̹̺ͅƝ̴ȳ̳ TH̘Ë͖́̉ ͠P̯͍̭O̚N̐Y̡ H̸̡̪̯ͨ͊̽̅̾̎Ȩ̬̩̾͛ͪ̈́̀́͘ ̶̧̨̱̹̭̯ͧ̾ͬC̷̙̲̝͖ͭ̏ͥͮ͟Oͮ͏̮̪̝͍M̲̖͊̒ͪͩͬ̚̚͜Ȇ̴̟̟͙̞ͩ͌͝S̨̥̫͎̭ͯ̿̔̀ͅ**

Have you tried using an XML parser instead?

> **Moderator's Note**
>
> This post is locked to prevent inappropriate edits to its content. The post looks exactly as it is supposed to look - there are no problems with its content. Please do not flag it for our attention.

Share Follow

edited Nov 12, 2020 at 14:21

community wiki
10 revs, 6 users 24%
bobince

179 Kobi: I think it's time for me to quit the post of Assistant Don't Parse HTML With Regex Officer. No matter how many times we say it, they won't stop coming every day... every hour even. It is a lost cause, which someone else can fight for a bit. So go on, parse HTML with regex, if you must. It's only broken code, not life and death. – bobince Nov 13, 2009 at 23:18

Comments disabled on deleted / locked posts / reviews

3584

+50

While parsing *arbitrary* HTML with only a regex is impossible, it's sometimes appropriate to use them for parsing a *limited, known* set of HTML.

If you have a small set of HTML pages that you want to scrape data from and then stuff into a database, regexes might work fine. For example, I recently wanted to get the names, parties, and districts of Australian federal Representatives, which I got off of the Parliament's web site. This was a limited, one-time job.

Regexes worked just fine for me, and were very fast to set up.

Share
Improve this answer
Follow

edited Dec 4 at 10:00

community wiki
11 revs, 11 users 35%
Kaitlin Duck Sherwood

162 Also, scraping fairly regularly formatted data from large documents is going to be WAY faster with judicious use of scan & regex than any generic parser. And if you are comfortable with coding regexes, way faster to code than coding xpaths. And almost certainly less fragile to changes in what you are scraping. So bleh. – Michael Johnston Apr 17, 2012 at 20:47

319 @MichaelJohnston "Less fragile"? Almost certainly not. Regexes care about text-formatting details than an XML parser can silently ignore. Switching between `&foo;` encodings and `CDATA` sections? Using an HTML minifier to remove all whitespace in your document that the browser doesn't render? An XML parser won't care, and neither will a well-written XPath statement. A regex-based "parser", on the other hand... – Charles Duffy Jul 11, 2012 at 16:03

81 @xiaomao indeed, if having to know all the gotchas and workarounds to get an 80% solution that fails the rest of the time "works for you", I can't stop you. Meanwhile, I'm over on my side of the fence using parsers that work on 100% of syntactically valid XML. – Charles Duffy Jul 12, 2012 at 16:07

449 I once had to pull some data off ~10k pages, all with the same HTML template. They were littered with HTML errors that caused parsers to choke, and all their styling was inline or with `<font>` etc.: no classes or IDs to help navigate the DOM. After fighting all day with the "right" approach, I finally switched to a regex solution and had it working in an hour. – Paul A Jungwirth Sep 7, 2012 at 7:14

49 @CharlesDuffy: definitely less fragile. When the third-party changes their html, they are much more likely to change the structure (breaking your xpaths) than the leaf nodes you are scraping. Scraping is not parsing. Scraping is pulling specific bits of data from a puddle of designer contaminated crap you don't care about. You DO NOT WANT to parse that puddle. You want to do only the absolute minimum amount of "parsing" that will get you your data. You don't CARE about the structure. You care about your bits of data. – Michael Johnston Nov 19, 2013 at 3:23

2357

+100

I think the flaw here is that HTML is a Chomsky Type 2 grammar (context free grammar) and a regular expression is a Chomsky Type 3 grammar (regular grammar). Since a Type 2 grammar is fundamentally more complex than a Type 3 grammar (see the Chomsky hierarchy), you can't possibly make this work.

But many will try, and some will even claim success - but until others find the fault and totally mess you up.

Share

Improve this answer

Follow

edited Aug 22, 2022 at 12:51

community wiki
10 revs, 9 users 19%
Vlad Gudim

282 The OP is asking to parse a very limited subset of XHTML: start tags. What makes (X)HTML a CFG is its potential to have elements between the start and end tags of other elements (as in a grammar rule `A -> s A e`). (X)HTML does *not* have this property *within* a start tag: a start tag cannot contain other start tags. The subset that the OP is trying to parse is not a CFG. – LarsH Mar 2, 2012 at 8:43

132 In CS theory, regular languages *are* a strict subset of context-free languages, but regular expression implementations in mainstream programming languages are more powerful. As noulakaz.net/weblog/2007/03/18/… describes, so-called "regular expressions" can check for prime numbers in unary, which is certainly something that a regular expression from CS theory can't accomplish. – Adam Mihalcin Mar 19, 2012 at 23:50

18 @eyelidlessness: the same "only if" applies to all CFGs, does it not? I.e. if the (X)HTML input is not well-formed, not even a full-blown XML parser will work reliably. Maybe if you give examples of the "(X)HTML syntax errors implemented in real world user agents" you're referring to, I'll understand what you're getting at better. – LarsH May 22, 2012 at 5:09

102 @AdamMihalcin is exactly right. Most extant regex engines are more powerful than Chomsky Type 3 grammars (eg non-greedy matching, backrefs). Some regex engines (such as Perl's) are Turing complete. It's true that even those are poor tools for parsing HTML, but this oft-cited argument is not the reason why. – dubiousjim May 31, 2012 at 13:44

7 To say the grammar of language A dictates it's parsing capabilities of another language B based on its grammar, is not valid. For example, just because HTML is Chomsky Type 2 language, doesn't mean you could write pure HTML which could parse any Chomsky Type 3 language. HTML itself is not a language with any features that give it the ability to parse other languages. Please don't say "Javascript", because javascript is not parsed by something written in HTML. – AaronLS Nov 24, 2015 at 23:54

1183

Disclaimer: use a parser if you have the option. That said...

This is the regex I use (!) to match HTML tags:

```
<(?:"[^"]*"['"]*|'[^']*'['"]*|[^'">])+>
```

It may not be perfect, but I ran this code through a *lot* of HTML. Note that it even catches strange things like `<a name="badgenerator"">`, which show up on the web.

I guess to make it not match self contained tags, you'd either want to use Kobi's negative look-behind:

```
<(?:"[^"]*"['"]*|'[^']*'['"]*|[^'">])+(?<!/\s*)>
```

or just combine if and if not.

To downvoters: This is working code from an actual product. I doubt anyone reading this page will get the impression that it is socially acceptable to use regexes on HTML.

Caveat: I should note that this regex still breaks down in the presence of CDATA blocks, comments, and script and style elements. Good news is, you can get rid of those using a regex...

Share
Improve this answer
Follow

edited May 23, 2017 at 12:34

community wiki
5 revs, 2 users 92%
itsadok

124 I would go with something that works on sane things than weep about not being universally perfect :-) – prajeesh kumar May 10, 2012 at 3:44

24 so you do not actually solve the parsing problem with regexp only but as a part of the parser this may work. PS: working product doesn't mean good code. No offence, but this is how industrial programming works and gets their money – mishmashru Apr 19, 2013 at 12:18

45 Your regex starts fail on the very shortest possible, valid HTML: `<!doctype html><title><</title>`. Simple `'<!doctype html><title><</title>'.match(/<(?:"[^"]*"['"]*|'[^']*'['"]*|[^'">])+>/g)` returns `["<!doctype html>", "<title>", "<</title>"]` while should `["<title>", "</title>"]`. – user1180790 May 1, 2014 at 16:48

3 if we're just trying to match & not match the examples given, /<.([^r>][^>]*)?>/g works :-) // javascript: '<p> <a href="foo"> <br /> <hr class="foo" />'.match(/<.([^r>][^>]*)?>/g) – imma May 22, 2014 at 16:14

2 "Is someone using CDATA inside HTML?" - yes, I do. It takes less bytes if you show HTML source code in `<pre>` tags. – cweiske Oct 14, 2015 at 10:56

603

There are people that will tell you that the Earth is round (or perhaps that the Earth is an oblate spheroid if they want to use strange words). They are lying.

There are people that will tell you that Regular Expressions shouldn't be recursive. They are limiting you. They need to subjugate you, and they do it by keeping you in ignorance.

You can live in their reality or take the red pill.

Like Lord Marshal (is he a relative of the Marshal .NET class?), I have seen the ~~Underverse~~ Stack Based Regex-Verse and returned with ~~powers~~ knowledge you can't imagine. Yes, I think there were an Old One or two protecting them, but they were watching football on the TV, so it wasn't difficult.

I think the XML case is quite simple. The RegEx (in the .NET syntax), deflated and coded in base64 to make it easier to comprehend by your feeble mind, should be something like this:

```
7L0HYBxJliUmL23Ke39K9UrX4HShCIBgEyTYkEAQ7MGIzeaS7B1pRyMpqyqBymVWZV1mFkDM7Z28
995777333nvvvfe6O51OJ/ff/z9cZmQBbPbOStrJniGAqsgfP358Hz8itn6Po9/3eIue3+Px7/3F
86enJ8+/fHn64ujx7/t7vFuUd/Dx65fHJ6dHW9/7fd/t7fy+73Ye0v+f0v+Pv//JnTvureM3b169
OP7i9Ogyr5uiWt746u+BBqc/8dXx86PP7tzU9mfQ9tWrL18d3UGnW/z7nZ9htH/y9NXrsy9fvPjq
i5/46ss3p4z+x3e8b452f9/x93a2HxIkH44PpgeFyPD6lMAEHUdbcn8ffTP9fdTrz/8rBPCe05Iv
p9WsWF788Obl9MXJl0/PXnwONLozY747+t7x9k9l2z/4vv4kqo1//993+/vf2kC5HtwNcxXH4aOf
LRw2z9/v8WEz2LTZcpaV1TL/4c3h66ex2Xv95vjF0+PnX744PbrOm59ZVhso5UHYME/dfj768H7e
Yy5uQUydDAH9+/4eR11wHbqdfPnFF6cv3ogq/V23t++4z4620A13cSzd7O1s/77rpw+ePft916c7
O/jj2bNnT7e/t/397//M9+ibA/7s6ZNnz76PP0/kT2rz/Ts/s/0NArvziYxVEZWxbm93xsrUfnlm
rASN7Hf93u/97vvf+2Lx/e89L7+/FSXiz4Bkd/hF5mVq9Yik7fcncft9350QCu+efkr/P6BfntEv
z+iX9c4eBrFz7wEwpB9P+d9n9MfuM3yzt7Nzss0/nuJfbra3e4BvZFR7z07pj3s7O7uWJM8eCkme
nuCPp88MfW6kDeH7+26PSTX8vu+ePAAiO4LVp4zIPWC1t7O/8/+pMX3rzo2KhL7+8s23T1/RhP0e
```

```
vyvm8HbsdmPXYDVhtpdnAzJ1k1jeufOtUAM8ffP06Zcnb36fl6dPXh2f/F6nRvruyHfMd9rgJp0Y
gvsRx/6/ZUzfCtX4e5hTndGzp5jQo9e/z+s3p1/czAUMlts+P3tz+uo4tISd745uJxvb3/v4ZlWs
mrjfd9SG/swGPD/6+nh+9MF4brTBRmh1Tl5+9eT52ckt5oR0xldPzp7GR8pfuXf5PWJv4nJIwvbH
W3c+GY3vPvrs9zj8Xb/147/n7/b7/+52DD2gsSH8zGDvH9+i9/fu/PftTfTXYf5hB+9H7P1BeG52
MTtu4S2cTAjDizevv3ry+vSNb8N+3+/1po2anj4/hZsGt3TY4GmjYbEKDJ62/pHB+3/LmL62wdsU
1J18+eINzTJr3dMvXr75fX7m+MXvY9XxF2e/9+nTgPu2bgwh5U0f7u/74y9Pnh6/OX4PlA2UlwTn
xenJG8L996VhbP3++PCrV68QkrjveITxr2TIt+lL+f3k22fPn/6I6f/fMqZvqXN/K4Xps6sazUGZ
GeQlar49xEvajzI35VRevDl78/sc/b7f6jkG8Va/x52N4L9lBe/kZSh1hr9fPj19+ebbR4AifyuY
12efv5CgGh9TroR6Pj2l748iYxYgN8Z7pr0HzRLg66FnRvcjUft/45i+pRP08vTV6TOe2N/9jv37
R9P0/5YxbXQDeK5E9R12XdDA/4zop+/9Ht/65PtsDVlBBUqko986WsDoWqvbPD2gH/T01DAC1NVn
3/uZ0feZ+T77fd/GVMkA4KjeMcg6RcvQLRl8HyPaWVStdv17PwHV0bOB9xUh7rfMp5Zu3icBJp25
D6f0NhayHyfI3HXHY6YYCw7Pz17fEFhQKzS6ZWChrX+kUf7fMqavHViEPPKjCf1/y5hukcyPTvjP
mHQCppRDN4nbVFPaT8+ekpV5/TP8g/79mVPo77PT1/LL7/MzL7548+XvdfritflFY00fxIsvSQPS
mvctdYZpbt7vxKRfj3018OvC/hEf/79lTBvM3debWj+b8KO0wP+3OeM2aYHumuCAGonmCrxw9cVX
X1C2d4P+uSU7eoBUMzI3/f9udjbYl/el04dI7s8fan8dWRjm6gFx+NrKeFP+WX0CxBdPT58df/X8
DaWLX53+xFdnr06f/szv++NnX7x8fnb6NAhIwsbPkPS7iSUQAFETvP2Tx8+/Og0Xt/yBvDn9vd/c
etno8S+81QKXptq/ffzKZFZ+4e/743e8zxino+8RX37/k595h5/H28+y7fPv490hQdJ349E+txB3
zPZ5J/jsR8bs/y1j2hh/2fkayOqEmYcej0cXUWMN7QrqBwjDrVZRfyQM3xjj/EgYvo4wfLTZrnVS
ebdKq0XSZJvzajKQDUv1/P3NwbEP7cN5+Odivv9/ysPfhHfkOP6b9Fl+91v7LD9aCvp/+Zi+7lLQ
j0zwNzYFP+/Y6r1NcFeDbfBIo8rug3zS3/3WPumPlN3/y8f0I2X3cz4FP+/Y6htSdr2I42fEuSPX
/ewpL4e9/n1evzn94hb+Plpw2+dnbyh79zx0CsPvbq0lb+UQ/h7xvqPq/Gc24PnR18fzVrp8I57d
mehj7ebk5VdPnp+d3GJOSP189eTsaXyk/JV7l98j4SAZgRxtf7x155PR+O6jz36Pw9/1Wz/+e/5u
v//vbsfQAxobws8M9v7xLXp/785/395ED4nO1wx5fsTeH4LnRva+eYY8rpZUBFb/j/jfm8XAvfEj
4/b/ljF1F9B/jx5PhAkp1nu/+y3n+kdZp/93jWmjJ/M11TG++VEG6puZn593PPejoOyHMQU/79jq
GwrKfpSB+tmcwZ93XPkjZffDmIKfd2z1DSm7bmCoPPmjBNT74XkrVf71I/Sf6wTU7XJA4RB+lIC6
mW1+xN5GWw1/683C5rnj/m364cmr45Pf6/SN9H4Us4LISn355vjN2ZcvtDGT6fHvapJcMISmxc0K
MAD4IyP6/5Yx/SwkP360FvD1VTH191mURr/HUY+2P3I9boPnz7Ju/pHrcWPnP3I9/r/L3sN0v52z
0fEgNrgbL8/Evfh9fw/q5Xf93u/97vvf+2Lx/e89L7+/Fe3iZ37f34P5h178kTfx/5YxfUs8vY26
7/d4/OWbb5++ogn7PX5XzOHtOP3GrsHmqobOVO/8Hh1Gk/TPl198QS6w+rLb23fcZ0fMaTfjsv29
7Zul7me2v0FgRoYVURnf9nZEkDD+H2VDf8hjeq8xff1s6GbButNLacEtefHm9VdPXp++CRTw7/v9
r6vW8b9eJ0+/PIHzs1HHdyKE/x9L4Y+s2f+PJPX/1dbsJn3wrY6wiqv85vjVm9Pnp+DgN8efM5va
j794+eb36Xz3mAf5+58+f3r68s230dRvJcxKn/l//oh3f+7H9K2O0r05PXf85s2rH83f/1vGdAvd
w+qBFqsoWvzspozD77EpXYeZ7yzdfxy0ec+l+8e/8FbR84+Wd78xbvn/qQQMz/J7L++GPB7N0MQa
2vTMBwjDrVI0PxKGb4xxfiQMX0cYPuq/Fbx2C1sU8yEF+F34iNsx1xOGa9t6l/yX70uqmxu+qBGm
AxlxWwVS11O97ULqlsFIUvUnT4/fHIuL//3f9/t9J39Y9m8W/Tuc296yUeX/b0PiHwUeP1801Y8C
j/9vz9+PAo8f+Vq35Jb/n0rAz7Kv9aPA40fC8PP8f+Rf3sC8PP08DjR1L3DXHoj6SuIz/CCghZNZb8
fb/Hf/2+37tjvuBY9vu3jmRvxNeGgQAuaAF6Pwj8/+e66M8/7rwpRNj6uVwXZRl52k0n3FVl95Q+
+fz0KSu73/dtkGDYdvZgSP5uskadrtViRKyal2IKAiQfiW+FI+tET/9/Txj9SFf8SFf8rOuKzagx
+r/vD34mUADO1P4/AQAA//8=
```

The options to set is `RegexOptions.ExplicitCapture`. The capture group you are looking for is `ELEMENTNAME`. If the capture group `ERROR` is not empty then there was a parsing error and the Regex stopped.

If you have problems reconverting it to a human-readable regex, this should help:

```
static string FromBase64(string str)
{
    byte[] byteArray = Convert.FromBase64String(str);

    using (var msIn = new MemoryStream(byteArray))
    using (var msOut = new MemoryStream()) {
        using (var ds = new DeflateStream(msIn, CompressionMode.Decompress)) {
            ds.CopyTo(msOut);
        }

        return Encoding.UTF8.GetString(msOut.ToArray());
    }
}
```

If you are unsure, no, I'm NOT kidding (but perhaps I'm lying). It WILL work. I've built tons of unit tests to test it, and I have even used (part of) the conformance tests. It's a tokenizer, not a full-blown parser, so it will only split the XML into its component tokens. It won't parse/integrate DTDs.

Oh... if you want the source code of the regex, with some auxiliary methods:

regex to tokenize an xml or the full plain regex

Share
Improve this answer
Follow

edited Nov 10, 2021 at 0:06

community wiki
13 revs, 11 users 68%
xanatos

77 not-sure-if-serious.jpg -- hopefully this is brilliant satire – Brad Mace Mar 8, 2011 at 14:53

86 Good Lord, it's massive. My biggest question is why? You realize that all modern languages have XML parsers, right? You can do all that in like 3 lines and be sure it'll work. Furthermore, do you also realize that pure regex is **provably** unable to do certain things? Unless you've created a hybrid regex/imperative code parser, but it doesn't look like you have. Can you compress random data as well? – Justin Morgan Mar 8, 2011 at 15:23

161 @Justin I don't need a reason. It could be done (and it wasn't illegal/immoral), so I have done it. There are no limitations to the mind except those we acknowledge (Napoleon Hill)... Modern languages can parse XML? Really? And I thought that THAT was illegal! :-) – xanatos Mar 8, 2011 at 15:31

105 Sir, I'm convinced. I'm going to use this code as part of the kernel for my perpetual-motion machine--can you believe those fools at the patent office keep rejecting my application? Well, I'll show them. I'll show them all! – Justin Morgan Mar 8, 2011 at 17:55

47 No, nothing is bug free: 1) All programs contain at least one bug. 2) All programs contain at least one line of unnecessary source code. 3) By #1 and #2 and using logical induction, it's a simple matter to prove that any program can be reduced to a single line of code with a bug. (from Learning Perl) – Scott Weaver Feb 16, 2012 at 0:53

322

In shell, you can parse HTML using sed:

1. Turing.sed
2. Write HTML parser (homework)
3. ???
4. Profit!

Related (why you shouldn't use regex match):

- If You Like Regular Expressions So Much, Why Don't You Marry Them?
- Regular Expressions: Now You Have Two Problems
- Hacking stackoverflow.com's HTML sanitizer

Share

Improve this answer

Follow

edited Apr 23, 2019 at 16:44

community wiki
10 revs, 7 users 43%
kenorb

4 I'm afraid you did not get the joke, @kenorb. Please, read the question and the accepted answer once more. This is not about HTML parsing tools in general, nor about HTML parsing shell tools, it's about parsing HTML via regexes. – Palec Oct 13, 2015 at 8:12

3 No, @Abdul. It is completely, provably (in the mathematical sense) impossible. – Palec Mar 24, 2017 at 13:24

6 Yes, that answer summarizes it well, @Abdul. Note that, however, regex implementations are not really *regular* expressions in the mathematical sense -- they have constructs that make them stronger, often Turing-complete (equivalent to Type 0 grammars). The argument breaks with this fact, but is still somewhat valid in the sense that regexes were never meant to be capable of doing such a job, though. – Palec Mar 24, 2017 at 14:24

2 And by the way, the joke I referred to was the content of this answer before kenorb's (radical) edits, specifically revision 4, @Abdul. – Palec Mar 24, 2017 at 14:26

15 The funny thing is that OP never asked to parse html using regex. He asked to match text (which happens to be HTML) using regex. Which is perfectly reasonable. – Paralife Mar 29, 2018 at 15:29

295

I agree that the right tool to parse XML and *especially HTML* is a parser and not a regular expression engine. However, like others have pointed out, sometimes using a regex is quicker, easier, and gets the job done if you know the data format.

Microsoft actually has a section of Best Practices for Regular Expressions in the .NET Framework and specifically talks about Consider[ing] the Input Source.

Regular Expressions do have limitations, but have you considered the following?

The .NET framework is unique when it comes to regular expressions in that it supports Balancing Group Definitions.

- See Matching Balanced Constructs with .NET Regular Expressions
- See .NET Regular Expressions: Regex and Balanced Matching
- See Microsoft's docs on Balancing Group Definitions

For this reason, I believe you CAN parse XML using regular expressions. Note however, that it **must be valid XML** (*browsers are very forgiving of HTML and allow* *bad XML syntax inside HTML*). This is possible since the "Balancing Group Definition" will allow the regular expression engine to act as a PDA.

Quote from article 1 cited above:

> **.NET Regular Expression Engine**
>
> As described above properly balanced constructs cannot be described by a regular expression. However, the .NET regular expression engine provides a few constructs that allow balanced constructs to be recognized.
>
> - `(?<group>)` - pushes the captured result on the capture stack with the name group.
> - `(?<-group>)` - pops the top most capture with the name group off the capture stack.
> - `(?(group)yes|no)` - matches the yes part if there exists a group with the name group otherwise matches no part.
>
> These constructs allow for a .NET regular expression to emulate a restricted PDA by essentially allowing simple versions of the stack operations: push, pop and empty. The simple operations are pretty much equivalent to increment, decrement and compare to zero respectively. This allows for the .NET regular expression engine to recognize a subset of the context-free languages, in particular the ones that only require a simple counter. This in turn allows for the non-traditional .NET regular expressions to recognize individual properly balanced constructs.

Consider the following regular expression:

```
(?=<ul\s+id="matchMe"\s+type="square"\s*>)
(?>
   <!-- .*? -->                  |
   <[^>]*/>                      |
   (?<opentag><(?!/)[^>]*[^/]>)  |
   (?<-opentag></[^>]*[^/]>)     |
   [^<>]*
)*
(?(opentag)(?!))
```

Use the flags:

- Singleline
- IgnorePatternWhitespace (not necessary if you collapse regex and remove all whitespace)

- IgnoreCase (not necessary)

Regular Expression Explained (inline)

```
(?=<ul\s+id="matchMe"\s+type="square"\s*>) # match start with <ul
id="matchMe"...
(?>                                        # atomic group / don't backtrack
(faster)
   <!-- .*? -->                  |         # match xml / html comment
   <[^>]*/>                      |         # self closing tag
   (?<opentag><(?!/)[^>]*[^/]>) |          # push opening xml tag
   (?<-opentag></[^>]*[^/]>)     |         # pop closing xml tag
   [^<>]*                                  # something between tags
)*                                         # match as many xml tags as possible
(?(opentag)(?!))                           # ensure no 'opentag' groups are on
stack
```

You can try this at A Better .NET Regular Expression Tester.

I used the sample source of:

```
<html>
<body>
<div>
   <br />
   <ul id="matchMe" type="square">
      <li>stuff...</li>
      <li>more stuff</li>
      <li>
          <div>
               <span>still more</span>
               <ul>
                    <li>Another &gt;ul&lt;, oh my!</li>
                    <li>...</li>
               </ul>
          </div>
      </li>
   </ul>
</div>
</body>
</html>
```

This found the match:

```
   <ul id="matchMe" type="square">
      <li>stuff...</li>
      <li>more stuff</li>
      <li>
          <div>
               <span>still more</span>
               <ul>
                    <li>Another &gt;ul&lt;, oh my!</li>
                    <li>...</li>
               </ul>
```

```
            </div>
        </li>
    </ul>
```

although it actually came out like this:

```
<ul id="matchMe" type="square">           <li>stuff...</li>           <li>more
stuff</li>          <li>              <div>                    <span>still
more</span>              <ul>                          <li>Another
&gt;ul&lt;, oh my!</li>                       <li>...</li>
</ul>               </div>          </li>       </ul>
```

Lastly, I really enjoyed Jeff Atwood's article: [Parsing Html The Cthulhu Way](). Funny enough, it cites the answer to this question that currently has over 4k votes.

Share
Improve this answer
Follow

edited Feb 23, 2020 at 4:43
Callum Watkins
2,981 ● 4 ● 32 ● 53

answered Sep 27, 2011 at 4:01
Sam
27.3k ● 12 ● 77 ● 102

19 `System.Text` is not part of C#. It's part of .NET. – John Saunders Feb 2, 2012 at 19:07

8 In the first line of your regex (`(?=<ul\s*id="matchMe"\s*type="square"\s*>) # match start with <ul id="matchMe"...`), in between "<ul" and "id" should be `\s+`, not `\s*`, unless you want it to match <ulid=... ;) – C0deH4cker Jul 6, 2012 at 2:49

@C0deH4cker You are correct, the expression should have `\s+` instead of `\s*`. – Sam Jul 6, 2012 at 22:33

4 Not that I really understand it, but I think your regex fails on `<img src="images/pic.jpg" />` – Scheintod Sep 27, 2013 at 17:05

3 @Scheintod Thank you for the comment. I updated the code. The previous expression failed for self closing tags that had a `/` somewhere inside which failed for your `<img src="images/pic.jpg" />` html. – Sam Sep 27, 2013 at 19:00

269

I suggest using [QueryPath]() for parsing XML and HTML in PHP. It's basically much the same syntax as jQuery, only it's on the server side.

Share
Improve this answer
Follow

edited May 12, 2015 at 18:54

community wiki
4 revs, 4 users 57%
John Fiala

9 @Kyle—jQuery does not parse XML, it uses the client's built–in parser (if there is one). Therefore you do not need jQuery to do it, but as little as two lines of [plain old JavaScript](). If there is no built–in parser, jQuery will not help. – RobG Oct 31, 2013 at 6:25

2 @RobG Actually jQuery uses the DOM, not the built-in parser. – Qix - MONICA WAS MISTREATED Sep 22, 2014 at 3:49

12 @Qix—you'd better tell the authors of the documentation then: "*jQuery.parseXML uses the native parsing function of the browser…*". Source: *jQuery.parseXML()* – RobG Sep 22, 2014 at 5:01

6 Having come here from the meme question (meta.stackexchange.com/questions/19478/the-many-memes-of-meta/…), I love that one of the answers is 'Use jQuery' – user8681 Apr 1, 2016 at 21:09

237

While the answers that you can't parse HTML with regexes are correct, they don't apply here. The OP just wants to parse one HTML tag with regexes, and that is something that can be done with a regular expression.

The suggested regex is wrong, though:

```
<([a-z]+) *[^/]*?>
```

If you add something to the regex, by backtracking it can be forced to match silly things like `<a >>`, `[^/]` is too permissive. Also note that `<space>*[^/]*` is redundant, because the `[^/]*` can also match spaces.

My suggestion would be

```
<([a-z]+)[^>]*(?<!/)>
```

Where `(?<! ... )` is (in Perl regexes) the negative look-behind. It reads "a <, then a word, then anything that's not a >, the last of which may not be a /, followed by >".

Note that this allows things like `<a/ >` (just like the original regex), so if you want something more restrictive, you need to build a regex to match attribute pairs separated by spaces.

Share
Improve this answer
Follow

answered Jan 27, 2010 at 12:54

community wiki
moritz

32 +1 for noting that the question is not about parsing full (X)HTML, it's about matching (X)HTML open tags. – LarsH Sep 8, 2012 at 2:26

11 Something else most of the answers seem to ignore, is that an HTML parser can very well use regular expressions in its implementation for parts of HTML, and I would be surprised if most parsers didn't do this. – Thayne Mar 26, 2015 at 19:15

2 @Thayne Exactly. When parsing individual tags, a regular expression is the right tool for the job. It is quite ridiculous that one has to scroll halfway down the page to find a reasonable answer. The accepted answer is incorrect because it mixes up lexing and parsing. – kasperd Nov 22, 2015 at 10:26

4 The answer given here will fail when an attribute value contains a '>' or '/' character. – Martin L Apr 21, 2016 at 8:14

2 The `<h1>` tag would like a word with you (easily fixed, I know, but still)... – jimbobmcgee May 7, 2020 at 18:28

201

Sun Tzu, an ancient Chinese strategist, general, and philosopher, said:

> It is said that if you know your enemies and know yourself, you can win a hundred battles without a single loss. If you only know yourself, but not your opponent, you may win or may lose. If you know neither yourself nor your enemy, you will always endanger yourself.

In this case your enemy is HTML and you are either yourself or regex. You might even be Perl with irregular regex. Know HTML. Know yourself.

I have composed a haiku describing the nature of HTML.

```
HTML has
complexity exceeding
regular language.
```

I have also composed a haiku describing the nature of regex in Perl.

```
The regex you seek
is defined within the phrase
<([a-zA-Z]+)(?:[^>]*[^/]*)?>
```

Share

Improve this answer

Follow

edited Oct 5, 2013 at 4:52

community wiki
2 revs, 2 users 98%
cytinus

How many syllables does `<([a-zA-Z]+)(?:[^>]*[^/]*)?>` have?? – LarsH Sep 25, 2023 at 20:24

Amazing, also this works fine. demo – Amine KOUIS Jan 2 at 16:46

Try:

193

```
<([^\s]+)(\s[^>]*?)?(?<!/)>
```

It is similar to yours, but the last `>` must not be after a slash, and also accepts `h1`.

Share

Improve this answer

Follow

edited Nov 25, 2009 at 21:12

community wiki
3 revs, 2 users 77%
Kobi

116 <a href="foo" title="5>3"> Oops </a> – Gareth Nov 13, 2009 at 23:11

68 `>` is valid in an attribute value. Indeed, in the 'canonical XML' serialisation you must not use `&gt;`. (Which isn't entirely relevant, except to emphasise that `>` in an attribute value is not at all an unusual thing.) – bobince Nov 14, 2009 at 0:15

5 @Kobi: what does the exlamation mark (the one you placed tpward the end) mean in a regexp? – Marco Demaio Apr 30, 2011 at 17:16

6 @bobince: are u sure? I don't understand anymore, so is this valid HTML too: `<div title="this tag is a <div></div>">hello</div>` – Marco Demaio Apr 30, 2011 at 17:31

3 @MarcoDemaio - `>` does not have to be escaped in an attribute value, but `<` does. So this is would be valid HTML: `<div title="this tag is a &lt;div>&lt;/div>">hello</div>` – Daniel Haley Oct 12, 2016 at 22:11

161

```
<?php
$selfClosing = explode(',',
'area,base,basefont,br,col,frame,hr,img,input,isindex,link,meta,param,embed');

$html = '
<p><a href="#">foo</a></p>
<hr/>
<br/>
<div>name</div>';

$dom = new DOMDocument();
$dom->loadHTML($html);
$els = $dom->getElementsByTagName('*');
foreach ( $els as $el ) {
    $nodeName = strtolower($el->nodeName);
    if ( !in_array( $nodeName, $selfClosing ) ) {
        var_dump( $nodeName );
    }
}
```

Output:

```
string(4) "html"
string(4) "body"
string(1) "p"
string(1) "a"
string(3) "div"
```

Basically just define the element node names that are self closing, load the whole html string into a DOM library, grab all elements, loop through and filter out ones which aren't self closing and operate on them.

I'm sure you already know by now that you shouldn't use regex for this purpose.

Share
Improve this answer
Follow

edited Nov 15, 2009 at 14:44

community wiki
2 revs
meder

1 If you're dealing with real XHTML then append getElementsByTagName with `NS` and specify the namespace. – meder omuraliev Nov 15, 2009 at 14:39

153

I don't know your exact need for this, but if you are also using .NET, couldn't you use Html Agility Pack?

Excerpt:

> *It is a .NET code library that allows you to parse "out of the web" HTML files. The parser is very tolerant with "real world" malformed HTML.*

Share
Improve this answer
Follow

answered Nov 16, 2009 at 23:15

community wiki
GONeale

CodePlex closed down (but this one is in the CodePlex archive). Perhaps update? – Peter Mortensen Aug 14, 2020 at 15:47

141

You want the first `>` not preceded by a `/`. Look here for details on how to do that. It's referred to as negative lookbehind.

However, a naïve implementation of that will end up matching `<bar/></foo>` in this example document

```
<foo><bar/></foo>
```

Can you provide a little more information on the problem you're trying to solve? Are you iterating through tags programatically?

Share

Improve this answer

Follow

edited Mar 19, 2012 at 15:04

community wiki
2 revs, 2 users 89%
Jherico

110

If you need this for PHP:

The PHP DOM functions won't work properly unless it is properly formatted XML. No matter how much better their use is for the rest of mankind.

simplehtmldom is good, but I found it a bit buggy, and it is is quite memory heavy [Will crash on large pages.]

I have never used querypath, so can't comment on its usefulness.

Another one to try is my DOMParser which is very light on resources and I've been using happily for a while. Simple to learn & powerful.

For Python and Java, similar links were posted.

For the downvoters - I only wrote my class when the XML parsers proved unable to withstand real use. Religious downvoting just prevents useful answers from being posted - keep things within perspective of the question, please.

Share

Improve this answer

Follow

edited Mar 19, 2012 at 15:02

community wiki
4 revs, 2 users 98%
SamGoody

100

Here's the solution:

```
<?php
// here's the pattern:
$pattern = '/<(\w+)(\s+(\w+)\s*\=\s*(\'|")(.*?)\\4\s*)*\s*(\/>|>)/';

// a string to parse:
$string = 'Hello, try clicking <a href="#paragraph">here</a>
    <br/>and check out.<hr />
    <h2>title</h2>
    <a name ="paragraph" rel= "I\'m an anchor"></a>
    Fine, <span title=\'highlight the "punch"\'>thanks<span>.
    <div class = "clear"></div>
    <br>';

// let's get the occurrences:
preg_match_all($pattern, $string, $matches, PREG_PATTERN_ORDER);

// print the result:
print_r($matches[0]);
?>
```

To test it deeply, I entered in the string auto-closing tags like:

1. <hr />
2. <br/>
3. <br>

I also entered tags with:

1. one attribute
2. more than one attribute
3. attributes which value is bound either into **single quotes** or into **double quotes**
4. attributes containing single quotes when the delimiter is a double quote and vice versa
5. "unpretty" attributes with a space before the "=" symbol, after it and both before and after it.

Should you find something which does not work in the proof of concept above, I am available in analyzing the code to improve my skills.

<EDIT> I forgot that the question from the user was to avoid the parsing of self-closing tags. In this case the pattern is simpler, turning into this:

```
$pattern = '/<(\w+)(\s+(\w+)\s*\=\s*(\'|")(.*?)\\4\s*)*\s*>/';
```

The user @ridgerunner noticed that the pattern does not allow **unquoted attributes** or **attributes with no value**. In this case a fine tuning brings us the following pattern:

```
$pattern = '/<(\w+)(\s+(\w+)(\s*\=\s*(\'|"|)(.*?)\\5\s*)?)*\s*>/';
```

</EDIT>

Understanding the pattern

If someone is interested in learning more about the pattern, I provide some line:

1. the first sub-expression (\w+) matches the tag name
2. the second sub-expression contains the pattern of an attribute. It is composed by:
3. one or more whitespaces \s+
4. the name of the attribute (\w+)
5. zero or more whitespaces \s* (it is possible or not, leaving blanks here)

6. the "=" symbol
7. again, zero or more whitespaces
8. the delimiter of the attribute value, a single or double quote ('|"). In the pattern, the single quote is escaped because it coincides with the PHP string delimiter. This sub-expression is captured with the parentheses so it can be referenced again to parse the closure of the attribute, that's why it is very important.
9. the value of the attribute, matched by *almost* anything: (.*?); in this specific syntax, using the **greedy match** (the question mark after the asterisk) the RegExp engine enables a "look-ahead"-like operator, which matches anything but what follows this sub-expression
10. here comes the fun: the \4 part is a **backreference operator**, which refers to a sub-expression defined before in the pattern, in this case, I am referring to the fourth sub-expression, which is the first attribute delimiter found
11. zero or more whitespaces \s*
12. the attribute sub-expression ends here, with the specification of zero or more possible occurrences, given by the asterisk.
13. Then, since a tag may end with a whitespace before the ">" symbol, zero or more whitespaces are matched with the \s* subpattern.
14. The tag to match may end with a simple ">" symbol, or a possible XHTML closure, which makes use of the slash before it: (/>|>). The slash is, of course, escaped since it coincides with the regular expression delimiter.

Small tip: to better analyze this code it is necessary looking at the source code generated since I did not provide any HTML special characters escaping.

Share
Improve this answer
Follow

edited Jan 31 at 11:37

community wiki
7 revs, 3 users 96%
yodabar

---

12 Does not match valid tags having attributes with no value, i.e. `<option selected>`. Also does not match valid tags with unquoted attribute values, i.e. `<p id=10>`. – ridgerunner Jul 25, 2011 at 15:01

1 @ridgerunner: Thanks very much for your comment. In that case the pattern must change a bit: $pattern = '/<(\w+)(\s+(\w+)(\s*\=\s*(\'|"|)(.*?)\\5\s*)?)*\s*>/'; I tested it and works in case of non-quoted attributes or attributes with no value. – yodabar Jul 25, 2011 at 16:41

How about a space before the tag name: `< a href="http://wtf.org" >` I'm pretty sure it is legal, but you don't match it. – Floris Oct 5, 2013 at 4:58

7 NO sorry, whitespaces before a tagname are illegal. Beyond being "pretty sure" why don't you provide some evidences of your objection? Here are mine, w3.org/TR/xml11/#sec-starttags referred to XML 1.1, and you can find the same for HTML 4, 5 and XHTML, as a W3C

validation would also warn if you make a test. As a lot of other blah-blah-poets around here, I did not still receive any intelligent argumentation, apart some hundred of minus to my answers, to demonstrate where my code fails according to the *rules of contract* specified in the question. I would only welcome them. – yodabar Oct 6, 2013 at 18:03

XML tags can contain colons, e.g. `<namespace:name>`, is that not so in HTML? – Qwertie Jul 17, 2020 at 21:54

94

Whenever I need to quickly extract something from an HTML document, I use Tidy to convert it to XML and then use XPath or XSLT to get what I need. In your case, something like this:

```
//p/a[@href='foo']
```

Share
Improve this answer
Follow

edited Apr 4, 2014 at 8:11

community wiki
2 revs, 2 users 67%
Amal Murali

91

I used a open source tool called HTMLParser before. It's designed to parse HTML in various ways and serves the purpose quite well. It can parse HTML as different treenode and you can easily use its API to get attributes out of the node. Check it out and see if this can help you.

Share
Improve this answer
Follow

answered Nov 16, 2009 at 18:34

community wiki
wen

87

I like to parse HTML with regular expressions. I don't attempt to parse idiot HTML that is deliberately broken. This code is my main parser (Perl edition):

```
$_ = join "",<STDIN>; tr/\n\r \t/ /s; s/</\n</g; s/>/>\n/g; s/\n ?\n/\n/g;
s/^ ?\n//s; s/ $//s; print
```

It's called *htmlsplit*, splits the HTML into lines, with one tag or chunk of text on each line. The lines can then be processed further with other text tools and scripts, such as grep, sed, Perl, etc. I'm not even joking :) Enjoy.

It is simple enough to rejig my slurp-everything-first Perl script into a nice streaming thing, if you wish to process enormous web pages. But it's not really necessary.

HTML Split

Some better regular expressions:

```
/(<.*?>|[^<]+)\s*/g    # Get tags and text
/(\w+)="(.*?)"/g       # Get attibutes
```

They are good for XML / XHTML.

With minor variations, it can cope with messy HTML... or convert the HTML -> XHTML first.

The best way to write regular expressions is in the Lex / Yacc style, not as opaque one-liners or commented multi-line monstrosities. I didn't do that here, yet; these ones barely need it.

Share
Improve this answer
Follow

edited Jan 31 at 11:38

community wiki
7 revs, 4 users 56%
Sam Watkins

64 "I don't attempt to parse idiot HTML that is deliberately broken." How does your code know the difference? – Kevin Panko Jul 26, 2011 at 20:38

6 (get attributes bug 1) `/(\w+)="(.*?)"/` assumes double quotes. It will miss values in single quotes. In html version 4 and earlier unquoted value is allowed, if it is a simple word. – David Andersson Sep 11, 2016 at 8:23

5 (get attributes bug 2) `/(\w+)="(.*?)"/` may falsely match text that looks like an attribute within an attribute, e.g. `<img title="Nope down='up' for aussies" src="..." />`. If applied globally, it will also match such things in ordinary text or in html comments. – David Andersson Sep 11, 2016 at 8:28

4 (get attributes bug 3) `/(\w+)="(.*?)"/` Optional whitespace should be allowed around the equal sign. – David Andersson Sep 11, 2016 at 8:42

5 (html split bug 1) `s/>/>\n/g` Since ">" is allowed in data, this may split text lines and confuse subsequent processing. – David Andersson Sep 11, 2016 at 9:09

74

About the question of the regular expression methods to parse (x)HTML, the answer to all of the ones who spoke about some limits is: you have not been trained enough to rule the force of this powerful weapon, since **nobody** here spoke about **recursion**.

A regular expression-agnostic colleague notified me this discussion, which is not certainly the first on the web about this old and hot topic.

After reading some posts, the first thing I did was looking for the "?R" string in this thread. The second was to search about "recursion".

No, holy cow, no match found. Since nobody mentioned the main mechanism a parser is built onto, I was soon aware that nobody got the point.

If an (x)HTML parser needs recursion, a regular expression parser without recursion is not enough for the purpose. It's a simple construct.

The **black art of regular expressions is hard to master**, so maybe there are further possibilities we left out while trying and testing our personal solution to capture the whole web in one hand... Well, I am sure about it :)

Here's the magic pattern:

```
$pattern = "/<([\w]+)([^>]*?)(([\s]*\/>)|(>((([^<]*?|<\!\-\-.*?\-\->)|(?R))*)
<\/\\1[\s]*>))/s";
```

Just try it. It's written as a PHP string, so the "s" modifier makes classes include newlines.

Here's a **sample note on the PHP manual** I wrote in January: Reference

(Take care. In that note I wrongly used the "m" modifier; it should be erased, notwithstanding it is discarded by the regular expression engine, since no `^` or `$` anchoring was used).

Now, we could speak about the limits of this method from a more informed point of view:

1. according to the specific implementation of the regular expression engine, recursion may have a limit in the **number of nested patterns parsed**, but it depends on the language used
2. although corrupted, (x)HTML does not drive into severe errors. It is not *sanitized*.

Anyhow, it is only a regular expression pattern, but it discloses the possibility to develop of a lot of powerful implementations.

I wrote this pattern to power the *recursive descent parser* of a template engine I built in my framework, and performances are really great, both in execution times or in memory usage (nothing to do with other template engines which use the same syntax).

Share

Improve this answer

Follow

edited Jan 31 at 11:39

community wiki
5 revs, 5 users 64%
yodabar

42 I'll put this in the "Regex which doesn't allow greater-than in attributes" bin. Check it against <input value="is 5 > 3?" /> – Gareth Jul 5, 2010 at 16:24

74 If you put something like that in production code, you would likely be shot by the maintainer. A jury would never convict him. – aehiilrs Jul 5, 2010 at 16:33

33 Regular expressions can't work because by definition they are not recursive. Adding a recursive operator to regular expressions basically makes a CFG only with poorer syntax. Why not use something designed to be recursive in the first place rather than violently insert recursion into something already overflowing with extraneous functionality? – Welbog Jul 6, 2010 at 18:38

20 My objection isn't one of functionality it is one of time invested. The problem with RegEx is that by the time you post the cutsey little one liners it appears that you did something more efficiently ("See one line of code!"). And of course no one mentions the half hour (or 3) that they spent with their cheat-sheet and (hopefully) testing every possible permutation of input. And once you get past all that when the maintainer goes to figure out or validate the code they can't just look at it and see that it is right. The have to dissect the expression and essentially retest it all over again... – Oorang Jul 10, 2010 at 15:11

17 ... to know that it is good. And that will happen even with people who are *good* with regex. And honestly I suspect that overwhelming majority of people won't know it well. So you take one of the most notorious maintenance nightmares and combine it with recursion which is the *other* maintenance nightmare and I think to myself what I really need on my project is someone a little less clever. The goal is to write code that bad programmers can maintain without breaking the code base. I know it galls to code to the least common denominator. But hiring excellent talent is hard, and you often... – Oorang Jul 10, 2010 at 15:17

71

There are some nice regexes for replacing HTML with BBCode here. For all you nay-sayers, note that he's not trying to fully parse HTML, just to sanitize it. He can probably afford to kill off tags that his simple "parser" can't understand.

For example:

```
$store =~ s/http:/http:\/\//gi;
$store =~ s/https:/https:\/\//gi;
$baseurl = $store;

if (!$query->param("ascii")) {
    $html =~ s/\s\s+/\n/gi;
    $html =~ s/<pre(.*?)>(.*?)<\/pre>/\[code]$2\[\/code]/sgmi;
}

$html =~ s/\n//gi;
$html =~ s/\r\r//gi;
$html =~ s/$baseurl//gi;
$html =~ s/<h[1-7](.*?)>(.*?)<\/h[1-7]>/\n\[b]$2\[\/b]\n/sgmi;
$html =~ s/<p>/\n\n/gi;
$html =~ s/<br(.*?)>/\n/gi;
$html =~ s/<textarea(.*?)>(.*?)<\/textarea>/\[code]$2\[\/code]/sgmi;
$html =~ s/<b>(.*?)<\/b>/\[b]$1\[\/b]/gi;
$html =~ s/<i>(.*?)<\/i>/\[i]$1\[\/i]/gi;
$html =~ s/<u>(.*?)<\/u>/\[u]$1\[\/u]/gi;
```

```
$html =~ s/<em>(.*?)<\/em>/\[i]$1\[\/i]/gi;
$html =~ s/<strong>(.*?)<\/strong>/\[b]$1\[\/b]/gi;
$html =~ s/<cite>(.*?)<\/cite>/\[i]$1\[\/i]/gi;
$html =~ s/<font color="(.*?)">(.*?)<\/font>/\[color=$1]$2\[\/color]/sgmi;
$html =~ s/<font color=(.*?)>(.*?)<\/font>/\[color=$1]$2\[\/color]/sgmi;
$html =~ s/<link(.*?)>//gi;
$html =~ s/<li(.*?)>(.*?)<\/li>/\[\*]$2/gi;
$html =~ s/<ul(.*?)>/\[list]/gi;
$html =~ s/<\/ul>/\[\/list]/gi;
$html =~ s/<div>/\n/gi;
$html =~ s/<\/div>/\n/gi;
$html =~ s/<td(.*?)>/ /gi;
$html =~ s/<tr(.*?)>/\n/gi;

$html =~ s/<img(.*?)src="(.*?)"(.*?)>/\[img]$baseurl\/$2\[\/img]/gi;
$html =~ s/<a(.*?)href="(.*?)"(.*?)>(.*?)<\/a>/\[url=$baseurl\/$2]$4\
[\/url]/gi;
$html =~ s/\[url=$baseurl\/http:\/\/(.*?)](.*?)\[\/url]/\[url=http:\/\/$1]$2\
[\/url]/gi;
$html =~ s/\[img]$baseurl\/http:\/\/(.*?)\[\/img]/\[img]http:\/\/$1\[\/img]/gi;

$html =~ s/<head>(.*?)<\/head>//sgmi;
$html =~ s/<object>(.*?)<\/object>//sgmi;
$html =~ s/<script(.*?)>(.*?)<\/script>//sgmi;
$html =~ s/<style(.*?)>(.*?)<\/style>//sgmi;
$html =~ s/<title>(.*?)<\/title>//sgmi;
$html =~ s/<!--(.*?)-->/\n/sgmi;

$html =~ s/\/\//\//gi;
$html =~ s/http:\//http:\/\//gi;
$html =~ s/https:\//https:\/\//gi;

$html =~ s/<(?:[^>'"]*|(['"]).*?\1)*>//gsi;
$html =~ s/\r\r//gi;
$html =~ s/\[img]\//\[img]/gi;
$html =~ s/\[url=\//\[url=/gi;
```

Share

Improve this answer

Follow

edited May 19, 2015 at 16:20

community wiki
4 revs, 4 users 88%
kenorb

20 Don't do this. Please. – maletor Sep 3, 2015 at 2:12

67

```
<\s*(\w+)[^/>]*>
```

The parts explained:

`<`: Starting character

`\s*`: It may have whitespaces before the tag name (ugly, but possible).

`(\w+)`: tags can contain letters and numbers (h1). Well, `\w` also matches '_', but it does not hurt I guess. If curious, use ([a-zA-Z0-9]+) instead.

`[^/>]*`: Anything except `>` and `/` until closing `>`

`>`: Closing `>`

UNRELATED

And to the fellows, who underestimate regular expressions, saying they are only as powerful as regular languages:

$a^n ba^n ba^n$ which is not regular and not even context free, can be matched with `^(a+)b\1b\1$`

Backreferencing FTW!

Share

Improve this answer

Follow

edited Oct 28, 2020 at 23:09

community wiki
7 revs, 4 users 46%
daghan

@GlitchMr, that was his point. Modern regular expressions are not technically regular, nor is there any reason for them to be. – alanaktion Feb 2, 2013 at 15:45

5 @alanaktion: The "modern" regular expressions (read: with Perl extensions) cannot match within `O(MN)` (M being regular expression length, N being text length). Backreferences are one of causes of that. The implementation in awk doesn't have backreferences and matches everything within `O(MN)` time. – null Feb 14, 2013 at 16:52

4 `<a href="foo" title="5>3"> Oops </a>` (quoting @Gareth - odd how people keep posting answers with this specific deficiency over and over. CDATA is kind of easy to overlook, but this is rather more basic) – Qwertie Jul 17, 2020 at 21:59

2 This regex will not work if html tag will contains `/` in between. For example : `<a href="example.com/test/example.html">` – Rohìt Jíndal Oct 18, 2022 at 11:07

Sorry, it doesn't match `<br />` check this demo. – Amine KOUIS Jan 2 at 18:17

63

As many people have already pointed out, HTML is not a regular language which can make it very difficult to parse. My solution to this is to turn it into a regular language using a tidy program and then to use an XML parser to consume the results. There are a lot of good options for this. My program is written using Java with the jtidy library to turn the HTML into XML and then Jaxen to xpath into the result.

Share

Improve this answer

Follow

answered Feb 4, 2010 at 16:22

community wiki
Corey Sanders

58

If you're simply trying to find those tags (without ambitions of parsing) try this regular expression:

```
/<[^/]*?>/g
```

I wrote it in 30 seconds, and tested here: https://regexr.com/

It matches the types of tags you mentioned, while ignoring the types you said you wanted to ignore.

Share

Improve this answer

Follow

edited Sep 15, 2023 at 4:50

community wiki
7 revs, 2 users 96%
Lonnie Best

2 FYI, you don't need to escape angle brackets. Of course, it does no harm to escape them anyway, but look at the confusion you could have avoided. ;) – Alan Moore May 29, 2016 at 7:47

I sometimes escape unnecessarily when I'm unsure if something is special character or not. I've edited the answer; it works the same but more concise. – Lonnie Best May 31, 2016 at 7:23

I checked it here, but it seems, doesn't match `<br />` – Amine KOUIS Jan 2 at 23:07

@AmineKOUIS You're right; it doesn't match the self-closing tags found in XHTML. However, the OP requested to only match opening tags, and HTML5 validators warn that `<br />` should simply be `<br>` instead. To match both opening tags and self-closing tags, without matching closing tags, try `/<[^/>][^>]*?>/g` . – Lonnie Best Jan 4 at 9:42

53

It seems to me you're trying to match tags without a "/" at the end. Try this:

```
<([a-zA-Z][a-zA-Z0-9]*)[^>]*(?<!/)>
```

Share

Improve this answer

Follow

answered Nov 15, 2009 at 17:13

community wiki
manixrock

11 This does not work. For the input '<x a="<b>"/><y>' the matches are x and y, although x is terminated. – ceving May 4, 2011 at 16:33

@ceving you're right demo – Amine KOUIS Jan 2 at 23:11

53

It's true that when programming it's usually best to use dedicated parsers and APIs instead of regular expressions when dealing with HTML, especially if accuracy is paramount (e.g., if your processing might have security implications). However, I don’t ascribe to a dogmatic view that XML-style markup should never be processed with regular expressions. There are cases when regular expressions are a great tool for the job, such as when making one-time edits in a text editor, fixing broken XML files, or dealing with file formats that look like but aren’t quite XML. There are some issues to be aware of, but they're not insurmountable or even necessarily relevant.

A simple regex like `<([^>"']|"[^"]*"|'[^']*')*>` is usually good enough, in cases such as those I just mentioned. It's a naive solution, all things considered, but it does correctly allow unencoded `>` symbols in attribute values. If you're looking for, e.g., a `table` tag, you could adapt it as `</?table\b([^>"']|"[^"]*"|'[^']*')*>`.

Just to give a sense of what a more "advanced" HTML regex would look like, the following does a fairly respectable job of emulating real-world browser behavior and the HTML5 parsing algorithm:

```
</?([A-Za-z][^\s>/]*)(?:=\s*(?:"[^"]*"|'[^']*'|[^\s>]+)|[^>])*(?:>|$)
```

The following matches a fairly strict definition of XML tags (although it doesn't account for the full set of Unicode characters allowed in XML names):

```
<(?:([_:A-Z][-.:\w]*)(?:\s+[_:A-Z][-.:\w]*\s*=\s*(?:"[^"]*"|'[^']*'))*\s*/?
|/([_:A-Z][-.:\w]*)\s*)>
```

Granted, these don't account for surrounding context and a few edge cases, but even such things could be dealt with if you really wanted to (e.g., by searching between the matches of another regex).

At the end of the day, use the most appropriate tool for the job, even in the cases when that tool happens to be a regex.

Share

Improve this answer

Follow

edited May 29, 2012 at 0:01

community wiki
2 revs
slevithan

Although it's not suitable and effective to use regular expressions for that purpose sometimes regular expressions provide quick solutions for simple match problems

51

and in my view it's not that horrbile to use regular expressions for trivial works.

There is a definitive blog post about matching innermost HTML elements written by Steven Levithan.

Share
Improve this answer
Follow

answered Feb 9, 2010 at 3:59

community wiki
Emre Yazici

43

If you only want the tag names, it should be possible to do this via a regular expression.

```
<([a-zA-Z]+)(?:[^>]*[^/] *)?>
```

should do what you need. But I think the solution of "moritz" is already fine. I didn't see it in the beginning.

For all downvoters: In some cases it just makes sense to use a regular expression, because it can be the easiest and quickest solution. I agree that in general you should not parse HTML with regular expressions.

But regular expressions can be a very powerful tool when you have a subset of HTML where you know the format and you just want to extract some values. I did that hundreds of times and almost always achieved what I wanted.

Share
Improve this answer
Follow

edited Aug 14, 2020 at 16:00

community wiki
5 revs, 2 users 60%
morja

please it doesn't match 'br' tag, here is a demo – Amine KOUIS Jan 3 at 13:33

@AmineKOUIS thanks, but the OP specifically asked not to match <br /> – morja Jan 6 at 15:36

39

+50

The OP doesn't seem to say what he needs to do with the tags. For example, does he need to extract inner text, or just examine the tags?

I'm firmly in the camp that says a regular expression is not the be-all, end-all text parser. I've written a large amount of text-parsing code including this code to parse HTML tags.

While it's true I'm not all that great with regular expressions, I consider regular expressions just too rigid and hard to maintain for this sort of parsing.

Share

Improve this answer

Follow

edited Aug 14, 2020 at 15:59

community wiki
2 revs, 2 users 62%
Jonathan Wood

31

This may do:

```
<.*?[^/]>
```

Or without the ending tags:

```
<[^/].*?[^/]>
```

What's with the flame wars on HTML parsers? HTML parsers must parse (and rebuild!) the entire document before it can categorize your search. Regular expressions may be a faster / elegant in certain circumstances. My 2 cents...

Share

Improve this answer

Follow

edited Jul 5, 2010 at 16:27

community wiki
3 revs, 3 users 69%
Paul

7 `<a href="foo" title="5>3"> Oops </a>` (quoting @Gareth) – Qwertie Jul 17, 2020 at 22:06

@Qwertie If you escape > it works, you can check this demo – Amine KOUIS Jan 3 at 14:15

1 2 Next

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.