First Look at How a Regex Engine Works Internally

Knowing how the regex engine works enables you to craft better regexes more easily. It helps you understand quickly why a particular regex does not do what you initially expected. This saves you lots of guesswork and head scratching when you need to write more complex regexes.

After introducing a new regex token, this tutorial explains step by step how the regex engine actually processes that token. This inside look may seem a bit long-winded at certain times. But understanding how the regex engine works enables you to use its full power and help you avoid common mistakes.

While there are many implementations of regular expressions that differ sometimes slightly and sometimes significantly in syntax and behavior, there are basically only two kinds of regular expression engines: text-directed engines, and regex-directed engines. Nearly all modern regex flavors are based on regex-directed engines. This is because certain very useful features, such as lazy quantifiers and backreferences, can only be implemented in regex-directed engines.

A regex-directed engine walks through the regex, attempting to match the next token in the regex to the next character. If a match is found, the engine advances through the regex and the subject string. If a token fails to match, the engine *backtracks* to a previous position in the regex and the subject string where it can try a different path through the regex. This tutorial will talk a lot more about backtracking later on. Modern regex flavors using regex-directed engines have lots of features such as atomic grouping and possessive quantifiers that allow you to control this backtracking.

A text-directed engine walks through the subject string, attempting all permutations of the regex before advancing to the next character in the string. A text-directed engine never backtracks. Thus, there isn't much to discuss about the matching process of a text-directed engine. In most cases, a text-directed engine finds the same matches as a regex-directed engine.

When this tutorial talks about regex engine internals, the discussion assumes a regex-directed engine. It only mentions text-directed engines in situations where they find different matches. And that only really happens when your regex uses alternation with two alternatives that can match at the same position.

The Regex Engine Always Returns the Leftmost Match

This is a very important point to understand: a regex engine always returns the leftmost match, even if a "better" match could be found later. When applying a regex to a string, the engine starts at the first character of the string. It tries all possible permutations of the regular expression at the first character. Only if all possibilities have been tried and found to fail, does the engine continue with the second character in the text. Again, it tries all possible permutations of the regex, in exactly the same order. The result is that the regex engine returns the *leftmost* match.

When applying `cat` to `He captured a catfish for his cat.`, the engine tries to match the first token in the regex `c` to the first character in the match `H`. This fails. There are no other possible permutations of this regex, because it merely consists of a sequence of literal characters. So the regex engine tries to match the `c` with the `e`. This fails too, as does matching the `c` with the space. Arriving at the 4th character in the string, `c` matches `c`. The engine then tries to match the second token `a` to the 5th character, `a`. This succeeds too. But then, `t` fails to match `p`. At that point, the engine knows the regex cannot be matched starting at the 4th character in the string. So it continues with the 5th: `a`. Again, `c` fails to match here and the engine carries on. At the 15th character in the string, `c` again matches `c`. The engine then proceeds to attempt to match the remainder of the regex at character 15 and finds that `a` matches `a` and `t` matches `t`.

The entire regular expression could be matched starting at character 15. The engine is "eager" to report a match. It therefore reports the first three letters of catfish as a valid match. The engine never proceeds beyond this point to

see if there are any “better” matches. The first match is considered good enough.

In this first example of the engine’s internals, our regex engine simply appears to work like a regular text search routine. However, it is important that you can follow the steps the engine takes in your mind. In following examples, the way the engine works has a profound impact on the matches it finds. Some of the results may be surprising. But they are always logical and predetermined, once you know how the engine works.

Make a Donation

Page URL: https://www.regular-expressions.info/engine.html
Page last updated: 22 November 2019
Site last updated: 08 April 2020