6th September 2019

Sticky flag "y", searching at position

The flag `y` allows to perform the search at the given position in the source string.

To grasp the use case of `y` flag, and see how great it is, let's explore a practical use case.

One of common tasks for regexps is "lexical analysis": we get a text, e.g. in a programming language, and analyze it for structural elements.

For instance, HTML has tags and attributes, JavaScript code has functions, variables, and so on.

Writing lexical analyzers is a special area, with its own tools and algorithms, so we don't go deep in there, but there's a common task: to read something at the given position.

E.g. we have a code string `let varName = "value"`, and we need to read the variable name from it, that starts at position `4`.

We'll look for variable name using regexp `\w+`. Actually, JavaScript variable names need a bit more complex regexp for accurate matching, but here it doesn't matter.

A call to `str.match(/\w+/)` will find only the first word in the line. Or all words with the flag `g`. But we need only one word at position `4`.

To search from the given position, we can use method `regexp.exec(str)`.

If the `regexp` doesn't have flags `g` or `y`, then this method looks for the first match in the string `str`, exactly like `str.match(regexp)`. Such simple no-flags case doesn't interest us here.

If there's flag `g`, then it performs the search in the string `str`, starting from position stored in its `regexp.lastIndex` property. And, if it finds a match, then sets `regexp.lastIndex` to the index immediately after the match.

When a regexp is created, its `lastIndex` is `0`.

So, successive calls to `regexp.exec(str)` return matches one after another.

An example (with flag `g`):

```
let str = 'let varName';

let regexp = /\w+/g;
alert(regexp.lastIndex); // 0 (initially lastIndex=0)

let word1 = regexp.exec(str);
alert(word1[0]); // let (1st word)
alert(regexp.lastIndex); // 3 (position after the match)

let word2 = regexp.exec(str);
alert(word2[0]); // varName (2nd word)
alert(regexp.lastIndex); // 11 (position after the match)

let word3 = regexp.exec(str);
alert(word3); // null (no more matches)
alert(regexp.lastIndex); // 0 (resets at search end)
```

Every match is returned as an array with groups and additional properties.

We can get all matches in the loop:

```
let str = 'let varName';
let regexp = /\w+/g;

let result;

while (result = regexp.exec(str)) {
  alert( `Found ${result[0]} at position ${result.index}` );
  // Found let at position 0, then
  // Found varName at position 4
}
```

Such use of `regexp.exec` is an alternative to method `str.matchAll`.

Unlike other methods, we can set our own `lastIndex`, to start the search from the given position.

For instance, let's find a word, starting from position `4`:

```
let str = 'let varName = "value"';

let regexp = /\w+/g; // without flag "g", property lastIndex is ignored

regexp.lastIndex = 4;

let word = regexp.exec(str);
alert(word); // varName
```

We performed a search of `\w+`, starting from position `regexp.lastIndex = 4`.

Please note: the search starts at position `lastIndex` and then goes further. If there's no word at position `lastIndex`, but it's somewhere after it, then it will be found:

```
let str = 'let varName = "value"';

let regexp = /\w+/g;

regexp.lastIndex = 3;

let word = regexp.exec(str);
alert(word[0]); // varName
alert(word.index); // 4
```

…So, with flag `g` property `lastIndex` sets the starting position for the search.

Flag `y` makes `regexp.exec` to look exactly at position `lastIndex`, not before, not after it.

Here's the same search with flag `y`:

```
let str = 'let varName = "value"';

let regexp = /\w+/y;

regexp.lastIndex = 3;
alert( regexp.exec(str) ); // null (there's a space at position 3, not a word

regexp.lastIndex = 4;
alert( regexp.exec(str) ); // varName (word at position 4)
```

As we can see, regexp `/\w+/y` doesn't match at position `3` (unlike the flag `g`), but matches at position `4`.

Imagine, we have a long text, and there are no matches in it, at all. Then searching with flag `g` will go till the end of the text, and this will take significantly more time than the search with flag `y`.

In such tasks like lexical analysis, there are usually many searches at an exact position. Using flag `y` is the key for a good performance.

Previous lesson Next lesson

Share Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)