5th November 2019

Patterns and flags

Regular expressions are patterns that provide a powerful way to search and replace in text.

In JavaScript, they are available via the RegExp object, as well as being integrated in methods of strings.

Regular Expressions

A regular expression (also “regexp”, or just “reg”) consists of a *pattern* and optional *flags*.

There are two syntaxes that can be used to create a regular expression object.

The “long” syntax:

```
1  regexp = new RegExp("pattern", "flags");
```

And the “short” one, using slashes `"/"`:

```
1  regexp = /pattern/; // no flags
2  regexp = /pattern/gmi; // with flags g,m and i (to be covered soon)
```

Slashes `/.../` tell JavaScript that we are creating a regular expression. They play the same role as quotes for strings.

In both cases `regexp` becomes an instance of the built-in `RegExp` class.

The main difference between these two syntaxes is that pattern using slashes `/.../` does not allow for expressions to be inserted (like string template literals with `${...}`). They are fully static.

Slashes are used when we know the regular expression at the code writing time – and that’s the most common situation. While `new RegExp`, is more often used when we need to create a regexp “on the fly” from a dynamically generated string. For instance:

```
1  let tag = prompt("What tag do you want to find?", "h2");
2
3  let regexp = new RegExp(`<${tag}>`); // same as /<h2>/ if answered "h2" in th
```

Flags

Regular expressions may have flags that affect the search.

There are only 6 of them in JavaScript:

`i`

With this flag the search is case-insensitive: no difference between `A` and `a` (see the example below).

`g`

With this flag the search looks for all matches, without it – only the first match is returned.

`m`

Multiline mode (covered in the chapter Multiline mode of anchors ^ $, flag "m").

`s`

Enables "dotall" mode, that allows a dot `.` to match newline character `\n` (covered in the chapter Character classes).

`u`

Enables full unicode support. The flag enables correct processing of surrogate pairs. More about that in the chapter Unicode: flag "u" and class \p{...}.

`y`

"Sticky" mode: searching at the exact position in the text (covered in the chapter Sticky flag "y", searching at position)

Colors

From here on the color scheme is:

- regexp – `red`
- string (where we search) – `blue`
- result – `green`

Searching: str.match

As mentioned previously, regular expressions are integrated with string methods.

The method `str.match(regexp)` finds all matches of `regexp` in the string `str`.

It has 3 working modes:

1. If the regular expression has flag `g`, it returns an array of all matches:

```
let str = "We will, we will rock you";

alert( str.match(/we/gi) ); // We,we (an array of 2 substrings that match)
```

Please note that both `We` and `we` are found, because flag `i` makes the regular expression case-insensitive.

2. If there's no such flag it returns only the first match in the form of an array, with the full match at index `0` and some additional details in properties:

```
let str = "We will, we will rock you";

let result = str.match(/we/i); // without flag g

alert( result[0] );     // We (1st match)
alert( result.length ); // 1

// Details:
alert( result.index );  // 0 (position of the match)
alert( result.input );  // We will, we will rock you (source string)
```

The array may have other indexes, besides `0` if a part of the regular expression is enclosed in parentheses. We'll cover that in the chapter Capturing groups.

3. And, finally, if there are no matches, `null` is returned (doesn't matter if there's flag `g` or not).

This a very important nuance. If there are no matches, we don't receive an empty array, but instead receive `null`. Forgetting about that may lead to errors, e.g.:

```
let matches = "JavaScript".match(/HTML/); // = null

if (!matches.length) { // Error: Cannot read property 'length' of null
  alert("Error in the line above");
}
```

If we'd like the result to always be an array, we can write it this way:

```
let matches = "JavaScript".match(/HTML/) || [];

if (!matches.length) {
  alert("No matches"); // now it works
}
```

Replacing: str.replace

The method `str.replace(regexp, replacement)` replaces matches found using `regexp` in string `str` with `replacement` (all matches if there's flag `g`, otherwise, only the first one).

For instance:

```
// no flag g
alert( "We will, we will".replace(/we/i, "I") ); // I will, we will

// with flag g
alert( "We will, we will".replace(/we/ig, "I") ); // I will, I will
```

The second argument is the `replacement` string. We can use special character combinations in it to insert fragments of the match:

Symbols	Action in the replacement string
`$&`	inserts the whole match
`$``	inserts a part of the string before the match
`$'`	inserts a part of the string after the match
`$n`	if `n` is a 1-2 digit number, then it inserts the contents of n-th parentheses, more about it in the chapter Capturing groups
`$<name>`	inserts the contents of the parentheses with the given `name`, more about it in the chapter Capturing groups
`$$`	inserts character `$`

An example with `$&`:

```
alert( "I love HTML".replace(/HTML/, "$& and JavaScript") ); // I love HTML a
```

Testing: regexp.test

The method `regexp.test(str)` looks for at least one match, if found, returns `true`, otherwise `false`.

```
let str = "I love JavaScript";
let regexp = /LOVE/i;

alert( regexp.test(str) ); // true
```

Later in this chapter we'll study more regular expressions, walk through more examples, and also meet other methods.

Full information about the methods is given in the article Methods of RegExp and String.

Summary

- A regular expression consists of a pattern and optional flags: `g`, `i`, `m`, `u`, `s`, `y`.
- Without flags and special symbols (that we'll study later), the search by a regexp is the same as a substring search.
- The method `str.match(regexp)` looks for matches: all of them if there's `g` flag, otherwise, only the first one.
- The method `str.replace(regexp, replacement)` replaces matches found using `regexp` with `replacement`: all of them if there's `g` flag, otherwise only the first one.
- The method `regexp.test(str)` returns `true` if there's at least one match, otherwise, it returns `false`.

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…)