JavaScript Regular Expression

Summary: in this tutorial, you'll learn about JavaScript regular expressions. After the tutorial, you'll know how to use regular expressions effectively to search and replace strings.

Introduction to regular expressions in JavaScript

A regular expression is a string that describes a pattern such as email addresses and phone numbers.

In JavaScript, regular expressions are objects. JavaScript provides the built-in `RegExp` type that allows you to work with regular expressions effectively.

Regular expressions are useful for searching and replacing strings that match a pattern. They have many useful applications.

For example, you can use regular expressions to extract useful information in web scraping like product prices. Additionally, you can use regular expressions to validate form fields.

Creating a regular expression

To create a regular expression in JavaScript, you enclose its pattern in forward-slash characters (`/`) like this:

```
let re = /hi/;
```

> Note that a regular expression is not surrounded by single or double quotes like a regular string.

Alternatively, you can use the `RegExp` constructor to create a regular expression:

```
let re = new RegExp('hi');
```

Both regular expressions are instances of the `RegExp` type. They match the string `'hi'` .

Testing for matching

The `RegExp` object has many useful methods. One of them is the `test()` method that allows you to test if a string contains a match of the pattern in the regular expression.

The `test()` method returns `true` if the string argument contains a match.

The following example uses the `test()` method to test whether the string `'hi John'` matches the pattern `hi` :

```
let re = /hi/;
let result = re.test('hi John');

console.log(result); // true
```

Using pattern flags

Besides a pattern, a `RegExp` object also accepts an optional flag parameter. Flags are settings that change the searching behavior. Regular expressions have many flags. We'll cover the commonly used flags in this tutorial.

1) The ignore flag (i)

The `i` flag ignores cases when searching. The I stands for ignore case. When you use the `i` flag, the regex engine will perform a *case-insensitive* search. This means it will match both lowercase and uppercase characters.

By default, the regex engine performs a *case-sensitive* search. For example `/hi/` regular expression matches the string `hi` not `Hi` .

To search for either string hi, Hi, or HI, you add the `i` flag to the regular expression `/hi/i`

```
const re = /hi/i;
const result = re.test('Hi John');

console.log(result); // true
```

In this example, the `/hi/i` will match any string `hi`, `Hi`, and `HI`. Notice that you place the flag `i` after the last forward-slash character (`/`).

The following example shows how to use the flag in the `RegExp` constructor:

```
let re = new RegExp('hi','i');
let result = re.test('HI John');

console.log(result); // true
```

2) The global flag (g)

Another commonly used flag is the global flag (`g`). When you use a regular expression without the `g` flag, the `RegExp` object checks for a match in the string but stops after finding the first one.

However, when you use the `g` flag, the `RegExp` continues to search through the entire string for all matches and returns all of them.

You can also combine flags to perform more flexible searches. For example, the `gi` flags find every match in the string regardless of the case.

The `exec()` method of the `RegExp` performs a search for a match in a string and returns an array containing detailed information about the match.

The `exec()` method returns `null` if no match is found. However, it only returns one match at a time. To get all matches in a string, you need to call the `exec()` method multiple times, typically within a loop.

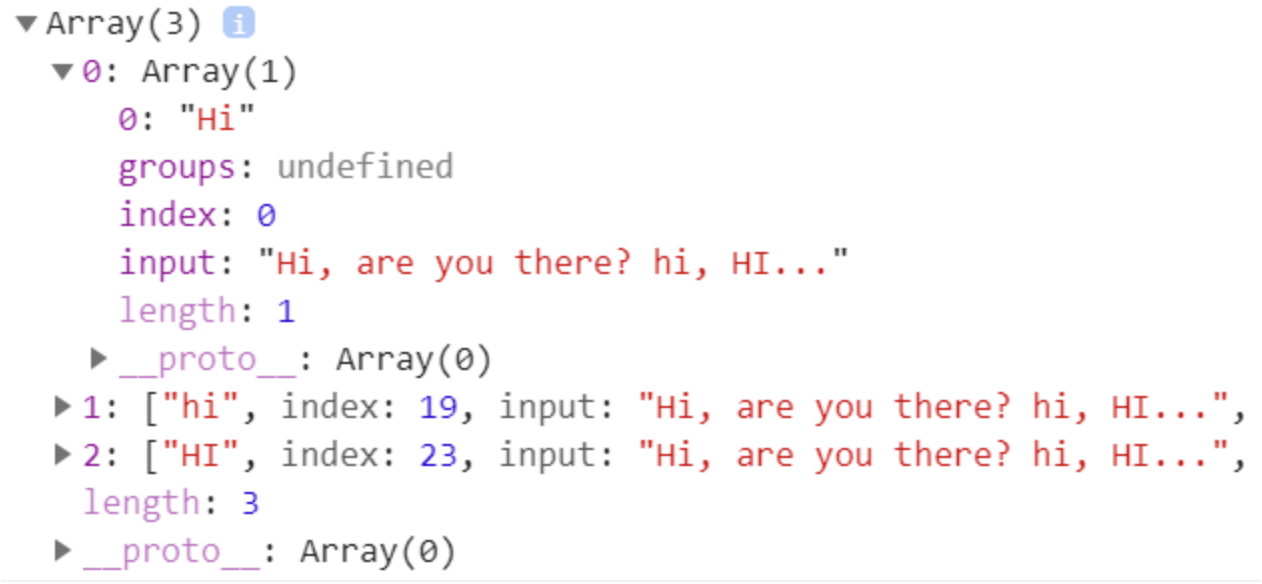
The following example uses the `exec()` method with a `do...while` loop to return all the matches:

```
let message = 'Hi, are you there? hi, HI...';
let re = /hi/gi;
```

```
let matches = [];
let match;
do {
    match = re.exec(message);
    if(match) {
      matches.push(match);
    }
} while(match != null)

console.dir(matches);
```

Output:



How it works:

- First, declare a `message` string that will be used for searching.
- Then, create a regular expression object with the pattern `/hi/gi`. The ignore flag (`i`) allows `re` object to ignore cases when executing the search and the global flag (`g`) instructs the `re` object to find all matches, not just the first one.
- Third, execute the `exec()` method until no match is found.
- Finally, show the result array in the console.

Searching strings

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

To find all matches, you use the global flag (`g`). To find the matches regardless of cases, you use the ignore flag (`i`).

The following example shows how to use the `match()` method:

```
let str = "Are you Ok? Yes, I'm OK";
let result = str.match(/OK/gi);

console.log(result);
```

Output:

```
["Ok", "OK"]
```

Replacing strings

The following example uses the `replace()` method of a string to replace the first occurrence of the string `'Ok'` in the string `str` :

```
const str = "Are you Ok? Yes, I'm OK";
const result = str.match(/OK/gi);

console.log(result);
```

Output:

```
Are you fine? Yes, I'm OK
```

To replace all occurrences of `OK` , you use a regular expression with the global flag (`g`):

```
let str = "Are you OK? Yes, I'm OK.";
let result = str.replace(/OK/g,'fine');
```

```
console.log(result);
```

Output:

```
Are you fine? Yes, I'm fine.
```

The following example uses both ignore and global flags to replace all occurrences of `OK` regardless of cases with the string `fine` :

```
let str = "Are you Ok? Yes, I'm OK.";
let result = str.replace(/OK/gi,'fine');

console.log(result);
```

Output:

```
Are you fine? Yes, I'm fine.
```

Summary

- Use `/pattern/` or `RegExp` constructor to create a regular expression.
- Use the ignore (`i`) and global (`g`) flags to modify the matching behavior.
- Use the `RegExp.test()` method to determine if a pattern is found in a string.
- Use the `RegExp.exec()` method to find the match and return an array containing the information of the match.
- Use the `String.match()` method to find all matches of a pattern in a string.
- Use the `String.replace()` method to replace text that matches a regular expression in a string.