Published in JavaScript in Plain English

Maxwell Follow

Nov 21, 2022 · 4 min read · ✦ · Listen

10 New JavaScript Features You Must Learn

Mastering new JavaScript features to make writing our code clean and fun

JavaScript is constantly being upgraded and iterated, and there are more and more new features to make our code clean and interesting to write, in this article we introduce 10 new features.

1. String.prototype.replaceAll

`replaceAll()` returns a new string in which all matches of the pattern are replaced by replacement terms. The pattern can be a string or a regular expression, and the replacement term can be a string or a function executed for each match.

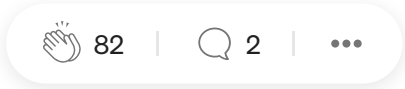
```
let str = 'I like apple ,I eat apple sometimes';
let newStr = str.replaceAll('apple', 'orange');

console.log(newStr);

/**** Output ****/
//I like orange ,I eat orange sometimes
```

2. Use “BigInt” to support large number calculation

MAX_SAFE_INTEGER” in JS will be unsafe to calculate numbers exceeding “Number.




```
let v = Math.pow(2, 55) === Math.pow(2, 55) + 1 // true
console.log(v);
let v2 = Math.pow(2, 55);
console.log(v2); //36028797018963970

let v3 = Math.pow(2, 55) + 1;
console.log(v3);   //36028797018963970
```

This problem can be completely avoided by using “BigInt”:

```
BigInt(Math.pow(2, 55)) === BigInt(Math.pow(2, 55)) + BigInt(1) // false
```

3. Use “Object.hasOwn” instead of the “in” operator

Sometimes we want to know if an attribute exists on an object, and we usually use the “in” operator, but this is flawed.

`in` operator returns true if the specified property is located in the object or its prototype chain:

```
const Person = function (age) {
  this.age = age
}
Person.prototype.name = 'Lily'

const p = new Person(24)
console.log('age' in p) // true
console.log('name' in p) // true
```

`Object.hasOwn`

```
let object = { age: 24 }
Object.hasOwn(object, 'age') // true
let object2 = Object.create({ age: 24 })
Object.hasOwn(object2, 'age') // false
let object3 = Object.create(null)
Object.hasOwn(object3, 'age') // false
```

4. numeric separator

The newly introduced value separator uses the _ (underscore) character to provide separation between groups of values, making them easier to read

```
let count = 1_000;
let number = 1_000_000;
let account = 1_000.0_001;
```

5. String.prototype.trimStart() / String.prototype.trimEnd()

`String.prototype.trim()` is used to remove spaces, newlines, etc. from the head and tail, and now the head and tail are controlled separately by `trimStart()`, `trimEnd()`. `trimLeft()`, `trimRight()` are their aliases.

```
let str = '  Hello JavaScript  ';
str.trimLeft();
//'Hello JavaScript  '
str.trimRight();
//'  Hello JavaScript'
```

6. Array.prototype.flat() / Array.prototype.flatMap()

Flattening arrays is a new feature of the Array prototype that allows you to raise the level of the lower array by passing in a level depth parameter (default is 1). You can write a larger number if you want to raise all levels, but this is not recommended. The `flatMap()` method first maps each element using the map function, and then flattens the result into a new array.

```
const a1 = [1, 2, [3, 4]].flat();
console.log(a1);  // [1, 2, 3, 4]


const a2 = [1, 2, [3, 4, [5, 6]]].flat(2);
console.log(a2); // [1, 2, 3, 4, 5, 6]

const a3 = [1, 2, 3, 4].flatMap(v => [v, v * 2]);
console.log(a3);  // [1, 2, 2, 4, 3, 6, 4, 8]
```

7. change the catch parameter to optional

During try...catch error handling, the code would report an error if no parameters were passed to catch. In the new specification, catch binding parameters and parentheses can be omitted.

```
try{
 return true;
}catch{
 return false;
}
```

8. Null merge operator (???)

When the left operand is null or undefined, it returns the right operand. Otherwise, it returns the operand on the left.

```
const str = null ?? 'default string';
console.log(str);
// expected output: "default string"

const num = 0 ?? 42;
console.log(num);
// expected output: 0

/**  Note
 * Unlike the logical or (||) operator, the logical or returns the right-hand oper
 */
```

9. Optional Chain Operators (?.)

If you access an attribute on a property that does not exist on the object, using the . operator will report an error directly, using ?. returns undefined

```
let obj = {};
console.log(obj.person.name)
// Cannot read properties of undefined (reading 'name')

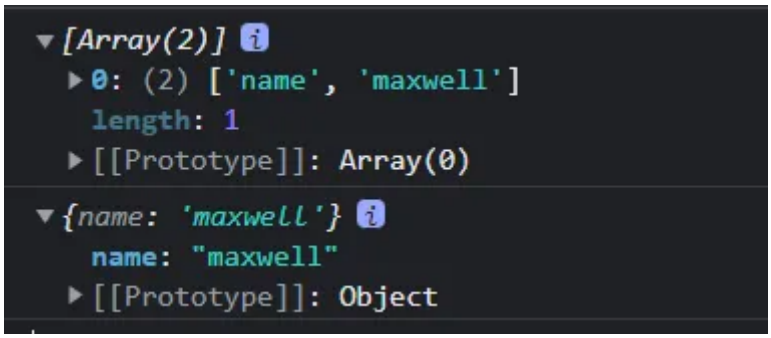
console.log(obj.?person?.name)
// expected output:undefine
```

10. Object.fromEntries()

`Object.entries` converts an object to a `[key, value]` key-value pair.

`object.fromEntries()` is used to reduce the key-value pair to an object structure.

```
    const entries = [['name', 'maxwell']];
    console.log(entries);
````const object = Object.fromEntries(entries);
    console.log(object);
```


```

8 Commonly Used JavaScript Libraries, Become a Real Master

Master these JavaScript tool librarys to make your projects look great.

levelup.gitconnected.com

JavaScript Design Patterns: Strategy Pattern

The purpose of learning design patterns is code reusability, to make code easier for others to understand, and to...

levelup.gitconnected.com

8 tools and methods often used in JavaScript

8 Javascript methods that are often used in work, remember these methods and avoid reinventing the wheel.

levelup.gitconnected.com

*More content at **PlainEnglish.io**. Sign up for our **free weekly newsletter**. Follow us on **Twitter**, **LinkedIn**, **YouTube**, and **Discord**. Interested in Growth Hacking? Check out **Circuit**.*

JavaScript Javascript Tips Programming Software Development Coding