链滴

最基础的 JS

作者：Rabbitzzc

原文链接：https://ld246.com/article/1596771523452

来源网站：链滴

不知道这些JavaScript方法有没有用，不过了解总比不知道好。送给需要的人

数组转换为对象

在JavaScript中，数组其实也是对象

```
const arr = ['hello', 'world'];
arr instanceof Object; // true
```

可以使用 Object.keys()和 Object.entries()来获取数组的所有键

```
Object.keys(arr); // ['0', '1']
Object.entries(arr); // [ [ '0', 'hello' ], [ '1', 'world' ] ]
```

使用 Object.assign()可以最简单的将数组转化为基础对象(**Plain Old JavaScript Object**)

```
const obj = Object.assign({}, arr);
obj instanceof Object; // true
Array.isArray(obj); // false
obj; // { '0': 'hello', '1': 'world' }
```

复制数组

有几种简单的方法可以在JavaScript中克隆数组。可以使用数组 slice()方法或扩展操作符

```
const arr = ['hello', 'world'];

// Clone using `slice()`
const arr2 = arr.slice();
arr2; // ['hello', 'world']
arr2 === arr; // false

// Clone using spread operator `...`
const arr3 = [...arr];
arr2; // ['hello', 'world']
arr2 === arr; // false
```

另外两种常见的方法是通过 concat()将数组合并空数组和使用 map()方法

```
// Clone using `concat()`
const arr4 = [].concat(arr);
arr4; // ['hello', 'world']
arr4 === arr; // false

// Clone using `map()`
const arr5 = arr.map(v => v);
arr5; // ['hello', 'world']
arr5 === arr; // false
```

这4种复制数组的方法实际上是相同的，基本上可以随便选择。当然，最显著的区别是 slice()有更好浏览器支持——一直追溯到Internet Explorer 4

Deep Copy vs Shallow Copy（深拷贝与浅拷贝）

上述4种方法都创建了数组的浅克隆。换句话说，它们克隆数组本身，而不是数组元素

数组内元素为(numbers, strings, null, undefined)，不需要关心，但是如果数组内部是对象，这时候需要斟酌了

```
const arr = [{ answer: 42 }];
const arr2 = arr.slice();
arr2[0].answer = 0;
arr[0].answer; // 0
```

JavaScript没有用于深度克隆数组的内置方法，但是项目中可以使用具有 cloneDeep()函数的库，如 lodash

```
const arr = [{ answer: 42 }];
const arr2 = require('lodash').cloneDeep(arr);
arr2[0].answer = 0;
// `42`, because Lodash did a deep clone.
arr[0].answer;
```

迭代数组

在JavaScript中有几种遍历数组的方法，一般来说，有4种常见的模式

- for (let i = 0; i < arr.length; ++i)
- forEach():arr.forEach((v, i) => { /* ... */ })
- for (const v of arr)
- for (const i in arr)

Async/Await

具有诸如 forEach之类的功能方法的最大难题是，因为将单独的函数传递给 forEach，所以很难将 async / await与 forEach一起使用

比如下面想要打印0-9，最后打印了9-0

```
async function print(n) {
  await new Promise(resolve => setTimeout(() => resolve(), 1000 - n * 100));
  console.log(n);
}

async function test() {
  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9].forEach(print);
}
test(); // 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
```

但是使用 for就比较理想了

```
async function print(n) {
  await new Promise(resolve => setTimeout(() => resolve(), 1000 - n * 100));
  console.log(n);
}
```

```
async function test() {
  for (const num of [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) {
    await print(num);
  }
}
test(); // 0 ~ 9
```

非数值属性

JavaScript数组是对象 typeof [] ==='object'。 这意味着数组可以具有非数字属性。避免使用 for / in 主要原因是 for / in遍历非数字属性，而 for， forEach 和for / of`会跳过非数字属性

```
const arr = ['a', 'b', 'c'];
arr['bad'] = 'alpha';

for (let key in arr) {
  console.log(arr[key]); // a, b, c, alpha
}

for (const val of arr) {
  console.log(val); //  a, b, c
}

arr.forEach(val => console.log(val)); //a, b, c
```

获取query值

window.location.search属性包含原始查询字符串。 例如，如果打开 http://localhost:5000/?answe =42，则 window.location.search = (?answer = 42)。

可以使用 URLSearchParams类解析查询字符串

```
const querystring = '?answer=42';
const params = new URLSearchParams(querystring);

params.get('answer'); // '42'
```

URLSearchParams的实例类似于JavaScript映射。为了获得搜索字符串中的所有键，可以使用 keys() 数或 entries()函数

这些函数返回JavaScript迭代器，而不是数组，因此需要使用 Array.from()对它们进行转换

```
const querystring = '?answer=42&question=unknown';
const params = new URLSearchParams(querystring);

Array.from(params.keys()); // ['answer', 'question']
Array.from(params.entries()); // [['answer', '42'], ['question', 'unknown']]
```

没有 URLSearchParams

URLSearchParams享有合理的浏览器支持，并在Node.js中工作。但是，URLSearchParams在 Inter et Explorer中不受支持

这时候，可以利用正则表达式来处理

```
function parse(qs) {
  return qs.
    replace(/^\?/, '').
    split('&').
    map(str => str.split('=').map(v => decodeURIComponent(v)));
}

parse('?answer=42&question=unknown'); // [['answer', '42'], ['question', 'unknown']]
```

字符串的第一个字母要大写

如果结合 toUpperCase()方法和 slice()方法，JavaScript字符串的第一个字母就很容易大写

```
const str = 'captain Picard';

const caps = str.charAt(0).toUpperCase() + str.slice(1);
caps; // 'Captain Picard'
```

如果希望将字符串中每个单词的第一个字母大写，可以使用 split()将字符串拆分为多个单词，然后使用 oin()将字符串重新组合在一起

```
const str = 'captain picard';

function capitalize(str) {
  return str.charAt(0).toUpperCase() + str.slice(1);
}

const caps = str.split(' ').map(capitalize).join(' ');
caps; // 'Captain Picard'
```

使用 CSS

```
.capitalize {
  text-transform: capitalize;
}
```

比较日期

```
const d1 = new Date('2019-06-01');
const d2 = new Date('2018-06-01');
const d3 = new Date('2019-06-01');
```

比较两个日期时，===和 ==都不起作用

```
const d1 = new Date('2019-06-01');
const d2 = new Date('2018-06-01');
const d3 = new Date('2019-06-01');

d1 === d3; // false
d1 == d3; // false
```

要比较两个日期，可以使用 toString或 valueOf。toString方法将日期转换为ISO日期字符串，而 valeOf方法将日期转换为自纪元以来的毫秒数

```
const d1 = new Date('2019-06-01');
const d2 = new Date('2018-06-01');
const d3 = new Date('2019-06-01');

d1.toString() === d2.toString(); // false
d1.toString() === d3.toString(); // true

d1.valueOf() === d2.valueOf(); // false
d1.valueOf() === d3.valueOf(); // true
```

before & after

虽然 ==和 ===都不能比较两个日期是否相等，但令人惊讶的是， <和 >都可以很好地比较日期

```
d1 < d2; // false
d1 < d3; // false

d2 < d1; // true
```

要检查日期a是否在日期b之前，只需检查a < b

另一个巧妙的技巧是 -- 可以在JavaScript中减去日期。减去a - b可以得到两个日期之间的差值(以毫为单位)

```
const d1 = new Date('2019-06-01');
const d2 = new Date('2018-06-01');
const d3 = new Date('2019-06-01');

d1 - d3; // 0
d1 - d2; // 1 year in milliseconds, 1000 * 60 * 60 * 24 * 365
```

换句话说，可以用a - b来比较两个日期a和b，如果b在a之后，那么a - b < 0

sleep

```
const start = Date.now();
async function pauseMe() {
  await sleep(2000);
  console.log('HacPai ==> ', Date.now() - start);
}
```