链滴

ECMAScript 2020 中新功能罗列

作者：Rabbitzzc
原文链接：https://ld246.com/article/1585291200486
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

驱动JavaScript的语言规范称为ECMAScript，这个帖子也是分享一些和探讨ES2020之下的最新规范对前端感兴趣的朋友可以关注关注。

由于许多人的电脑是不更新浏览器的，这也是很多前端开发头疼的地方，很多新的规范和方法并不能接使用！如果需要简化开发人员的生活，我们需要使用babel来开始使用用户无法全面使用的功能。

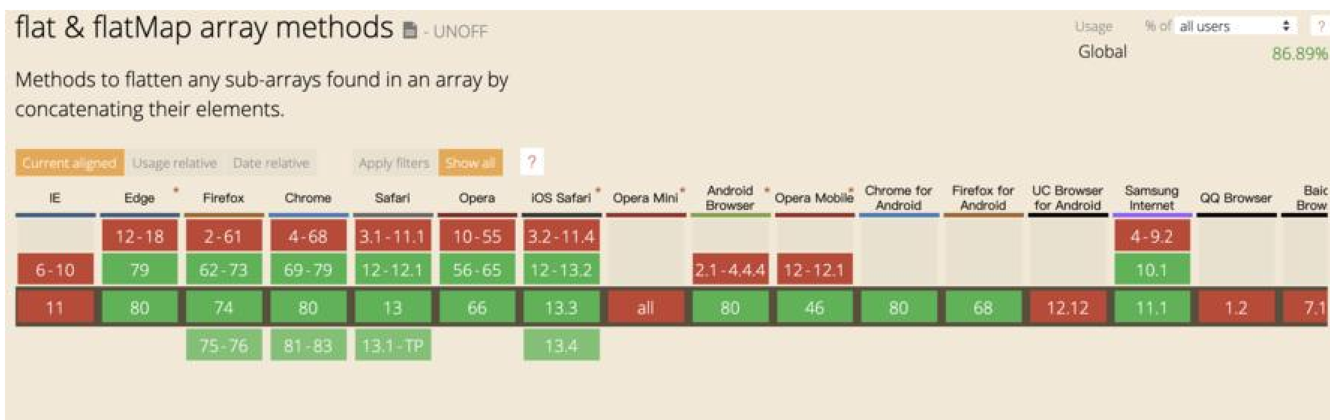
```
 yarn add parcel-bundler

"scripts": {
  "start": "parcel index.html"
},
```

.babelrc 配置

```
{
  "plugins": [
    "@babel/plugin-proposal-nullish-coalescing-operator",
    "@babel/plugin-proposal-optional-chaining",
    "@babel/plugin-proposal-class-properties",
    "@babel/plugin-proposal-private-methods",
    "@babel/plugin-syntax-bigint"
  ]
}
```

备注：下面的例子我都是通过 chrome80+ 运行的

flat & flatMap



IE	Edge	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser	Opera Mobile	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	QQ Browser	Baic Brow
	12-18	2-61	4-68	3.1-11.1	10-55	3.2-11.4							4-9.2		
6-10	79	62-73	69-79	12-12.1	56-65	12-13.2		2.1-4.4.4	12-12.1				10.1		
11	80	74	80	13	66	13.3	all	80	46	80	68	12.12	11.1	1.2	7.1
		75-76	81-83	13.1-TP		13.4									

Array.prototype.flat（）计划将数组递归展平至所需深度，并返回替换数组

语法：Array.prototype.flat（depth）
depth —默认值为1，使用时间展平所有嵌套的数组。

```
const numbers = [1, 2, [3, 4, [5, 6]]];
// Considers default depth of 1
numbers.flat();
> [1, 2, 3, 4, [5, 6]]
// With depth of 2
numbers.flat(2);
> [1, 2, 3, 4, 5, 6]
// Executes two flat operations
numbers.flat().flat();
> [1, 2, 3, 4, 5, 6]
// Flattens recursively until the array contains no nested arrays
numbers.flat(Infinity)
> [1, 2, 3, 4, 5, 6]
```

Array.prototype.flatMap（）映射使用映射执行的每个部分，并将结果展平为替换数组。它是地图作的副本，后跟一个depth = 1

语法：Array.prototype.flatMap（callback）
回调：执行该操作将产生新Array的一部分

```
const numbers = [1, 2, 3];
numbers.map(x => [x * 2]);
> [[2], [4], [6]]
numbers.flatMap(x => [x * 2]);
> [2, 4, 6]
```

私有类变量

类的主要目的之一是将我们的代码包含在可重用的模块中。一个类会在许多不同地方使用，但是又不望类中的变量都是 public，目前 JS 并不支持私有变量。

现在，通过在变量或函数前面添加一个简单的哈希符号，就可以实现私有变量了，也算是语法糖吧

```
class Message {
  #message = "Howdy"

  greet() { console.log(this.#message) }
```

原文链接：ECMAScript 2020 中新功能罗列

```
}

const greeting = new Message()

greeting.greet() // Howdy
console.log(greeting.#message) // Private name #message is not defined
```

Object.fromEntries

Object.fromEntries执行与Object.entries相反的操作。它将键值对列表转换为AN对象。

语法：Object.fromEntries（iterable）
iterable：类似于Array或Map的可迭代对象，或实现可迭代协议的对象

```
const records = [['name','Mathew'], ['age', 32]];
const obj = Object.fromEntries(records);
> { name: 'Mathew', age: 32}
Object.entries(obj);
> [['name','Mathew'], ['age', 32]];
```

Promise.allSettled()方法返回一个在所有给定的promise已被决议或被拒绝后议的promise，并带有一个对象数组，每个对象表示对应的promise结果

```
const promise1 = Promise.resolve(3);
const promise2 = new Promise((resolve, reject) => setTimeout(reject, 100, 'foo'));
const promises = [promise1, promise2];

Promise.allSettled(promises).
  then((results) => results.forEach((result) => console.log(result.status)));

// expected output:
// "fulfilled"
// "rejected"
```

大整数（BigInt）

由于JavaScript处理数字的方式，当您提高数字时，事情会变得有些古怪。如果大家在处理一些小数较多的图表计算，很容易踩 JS 处理数字的方式的坑。

原文链接：ECMAScript 2020 中新功能罗列

JavaScript可以处理的最大数字是2 ^ 53，可以通过 MAX_SAFE_INTEGER查看

```
const max = Number.MAX_SAFE_INTEGER;

console.log(max); // 9007199254740991
```

下面是一些negative_squared_cross_mark和奇怪的 JS 计算方式

```
console.log(max + 1); // 9007199254740992
console.log(max + 2); // 9007199254740992
console.log(max + 3); // 9007199254740994
console.log(Math.pow(2, 53) == Math.pow(2, 53) + 1); // true
```

这时候就可以使用新的 BigInt 数据类型解决此问题。在数字末尾加上字母'n'，我们可以开始使用并大量数字互动。

备注：无法将标准数字与BigInt数字混合使用，因此任何数学运算都需要使用BigInts完成

```
const bigNum = 100000000000000000000000000000n;

console.log(bigNum * 2n); // 200000000000000000000000000000n
```

String.prototype.trimStart() & trimEnd()

这是我最爱的方法之一

trimStart() 从字符串的开头删除空格，trimEnd() 从字符串的开头删除空格。

```
const greeting = ` Hello Javascript! `;
greeting.length;
> 19
greeting = greeting.trimStart();
> 'Hello Javascript! '
greeting.length;
> 18
greeting = 'Hello World!   ';
greeting.length;
> 15
greeting = greeting.trimEnd();
> 'Hello World!'
greeting.length;
> 12
```

Nullish Coalescing Operator

这个不好翻译，可以理解为更为严格的 || 运算符

由于JavaScript是动态类型的，因此在分配变量时，您需要牢记JavaScript对真实/错误值的处理。但是 JS 对于空字符串和 || 处理很奇怪。比如我们需要判断一个对象属性是否存在，不存在则为其他值：

```
let a = {a:0}
let b = {}
b.a = a.a || 1
> b.a = 1
```

这明显是不对的，目的是将 b.a 设置为a.a 的值，仅因为 a.a = 0，导致表达式运行结果不同，这在端开发中是很隐晦很隐晦的bug，一不小心就会犯错，比如我，经常采坑。

代替双管道 ||，我们可以使用双问号运算符将类型更严格一些(??)，仅当值为 null 或 undefined 才允许使用默认值。

```
let person = {
  profile: {
    name: "",
    age: 0
  }
};
console.log(person.profile.name ?? "Anonymous"); // ""
console.log(person.profile.age ?? 18); // 0
```

Optional Chaining Operator

这个 babel 工具如果大家关注一些前端信息的话，会听得比较多，也是受到好评的，虽然写起来的代看起来并不是那么优雅。

如果我们想要的是未定义的，我们可以返回一个值，但是如果它的路径是未定义的，该怎么办？这时就体现了 Optional Chaining Operator 的作用了

```
let person = {};

console.log(person.profile.name ?? "Anonymous"); // person.profile is undefined
console.log(person?.profile?.name ?? "Anonymous"); // 在. 之前加上问号?
console.log(person?.profile?.age ?? 18);
```

动态导入

如果你在开发中写了一个充满实用程序功能的文件(util.js)，其中某些功能可能很少使用，而导入其所依赖项可能只是浪费资源。现在，我们可以使用async / await在需要时动态导入依赖项。

这个不适用于浏览器，仅仅适用于 Node.js 环境

```
const add = (num1, num2) => num1 + num2;

export { add };

const doMath = async (num1, num2) => {
  if (num1 && num2) {
    const math = await import('./math.js');
    console.log(math.add(5, 10));
  };
};

doMath(4, 2);
```

似乎 Vue.js 动态引入组件就是这么干的~

结论

原文链接：ECMAScript 2020 中新功能罗列

现在，感兴趣的朋友可以尝试去使用这个新规范了，如果不想装 babel 插件，可以直接在 chrome 览器中直接复制代码或者自定义编码~

快去试试吧~□yum□yum□
um□yum

原文链接：ECMAScript 2020 中新功能罗列