【JS】2047- JS的这些新特性，你都用过么？

作者：纯爱掌门人

原文：https://juejin.cn/post/7321294927592456203

作为一门不断演进的语言，JavaScript每年都会引入新特性。这些特性的加入，能够帮助我们编写更加简洁、高效、易于维护的代码。然而，并非所有新特性都会立即广泛应用。它们的采用往往取决于社区的接受度以及浏览器的支持情况。比如我，会根据项目需求、团队习惯以及特性的成熟度来选择是否使用这些新特

性。以下是一些我认为特别有用的新JavaScript特性，以及一些补充的实例代码，大家一起共勉。

可选链

可选链让我们能够以安全的方式访问嵌套对象的属性，避免因为中间某个属性不存在而抛出错误。

```
const user = {
  profile: {
    name: 'John Doe',
    contact: {
      email: 'john@example.com',
    },
  },
};

const userEmail = user.profile?.contact?.email;
console.log(userEmail); // "john@example.com"

const userPhone = user.profile?.contact?.phone;
console.log(userPhone); // 输出undefined, 而不是抛出错误
```

空值合并运算符

空值合并运算符允许我们为可能是 `null` 或 `undefined` 的值提供一个默认值。

```
const defaultAge = 25;
const userAge = null;

const age = userAge ?? defaultAge;
console.log(age); // 25
```

Promise.allSettled()

这个方法让我们可以在所有的Promise都得到解决之后，无论是fulfilled还是rejected，都可以获得每一个Promise的结果。

```
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))); // "fulfille
d", "rejected"
```

动态导入

动态导入让我们可以在代码运行时按需加载模块。

```
button.addEventListener('click', async () => {
  try {
    const module = await import('./module1.js');
    module.load();
  } catch (error) {
    console.error('Module loading failed:', error);
  }
});
```

BigInt

BigInt是一种新的数值类型，让我们可以安全地操作大整数。

```
const largeNumber = BigInt('9007199254740991');
const anotherLargeNumber = 9007199254740992n; // 末尾的n表示BigInt
```

```
console.log(largeNumber + anotherLargeNumber); // 18014398509481983n
```

top-level await

`top-level await` 允许我们在模块顶层使用 `await` 关键字，简化了异步导入和操作的代码逻辑。

```
const data = await fetchData('api/data');
console.log(data);
```

逻辑赋值运算符

这些运算符结合了逻辑运算符和赋值运算符。

```
let a = null;
let b = 'default';
a ||= b;
console.log(a); // "default"

let x = 1;
let y = 3;
x &&= y;
console.log(x); // 1
```

String.prototype.replaceAll()

`replaceAll` 方法让我们可以更容易地替换字符串中的所有匹配项，而不仅仅是第一个匹配项。

```
let str = 'The quick brown fox jumps over the lazy dog. If the dog reacted, was it really la
zy?';
```

```
let newStr = str.replaceAll('dog', 'monkey');
console.log(newStr);
// "The quick brown fox jumps over the lazy monkey. If the monkey reacted, was it really laz
y?"
```

WeakRefs 和 FinalizationRegistry

这些新特性有助于管理内存。 `WeakRef` 对象允许您保持对另一个对象的弱引用，而不会阻止该对象被垃圾回收。 `FinalizationRegistry` 对象允许您在对象被垃圾回收时执行一些清理工作。

```
let obj = {};
const weakRef = new WeakRef(obj);

const registry = new FinalizationRegistry((value) => {
  console.log(`The object ${value} will be garbage collected.`);
});

registry.register(obj, 'myObj');

// 某个时间后，如果obj不再被其他引用，垃圾回收器会清除obj
obj = null;
```

总结

新特性的引入总是带来了新的编程范式和工具，我们可以利用它们来提升我们的开发效率和代码质量。随着新特性逐步融入我们的日常工作，我们需要保持学习的态度，不断适应和采纳这些新工具。

尽管如此，我们也要考虑到项目的现实需求，如浏览器兼容性和团队的熟悉度，以确保新特性的引入不会对项目造成负面影响。

最终，合理利用现代JavaScript特性，我们能够编写出更加现代化、高效且可维护的前端代码