JS 中使用扩展运算符的10种方法

> 链接：https://segmentfault.com/a/1190000038998504

复制数组

我们可以使用展开操作符复制数组，不过要注意的是这是一个**浅拷贝**。

```
const arr1 = [1,2,3];
const arr2 = [...arr1];
console.log(arr2);
// [ 1, 2, 3 ]
```

这样我们就可以复制一个基本的数组，注意，它不适用于多级数组或带有日期或函数的数组。

合并数组

假设我们有两个数组想合并为一个，早期间我们可以使用`concat`方法，但现在可以使用展开操作符：

```
const arr1 = [1,2,3];
const arr2 = [4,5,6];
const arr3 = [...arr1, ...arr2];
console.log(arr3);
// [ 1, 2, 3, 4, 5, 6 ]
```

我们还可以通过不同的排列方式来说明哪个应该先出现。

```
const arr3 = [...arr2, ...arr1];
console.log(arr3);
[4, 5, 6, 1, 2, 3];
```

此外，展开运算符号还适用多个数组的合并：

```
const output = [...arr1, ...arr2, ...arr3, ...arr4];
```

向数组中添加元素

```
let arr1 = ['this', 'is', 'an'];
arr1 = [...arr1, 'array'];
console.log(arr1);
// [ 'this', 'is', 'an', 'array' ]
```

向对象添加属性

假设你有一个`user`的对象，但它缺少一个`age`属性。

```
const user = {
  firstname: 'Chris',
  lastname: 'Bongers'
};
```

要向这个`user`对象添加`age`，我们可以再次利用展开操作符。

```
const output = {...user, age: 31};
```

使用 Math() 函数

假设我们有一个数字数组，我们想要获得这些数字中的最大值、最小值或者总和。

```
const arr1 = [1, -1, 0, 5, 3];
```

为了获得最小值，我们可以使用展开操作符和`Math.min`方法。

```
const arr1 = [1, -1, 0, 5, 3];
const min = Math.min(...arr1);
console.log(min);
// -1
```

同样，要获得最大值，可以这么做：

```
const arr1 = [1, -1, 0, 5, 3];
const max = Math.max(...arr1);
console.log(max);
// 5
```

如大家所见，最大值`5`，如果我们删除`5`，它将返回`3`。

你可能会好奇，如果我们不使用展开操作符会发生什么？

```
const arr1 = [1, -1, 0, 5, 3];
const max = Math.max(arr1);
console.log(max);
// NaN
```

这会返回**NaN**，因为JavaScript不知道数组的最大值是什么。

rest 参数

假设我们有一个函数，它有三个参数。

```
const myFunc(x1, x2, x3) => {
    console.log(x1);
    console.log(x2);
    console.log(x3);
}
```

我们可以按以下方式调用这个函数:

```
myFunc(1, 2, 3);
```

但是，如果我们要传递一个数组会发生什么。

```
const arr1 = [1, 2, 3];
```

我们可以使用展开操作符将这个数组扩展到我们的函数中。

```
myFunc(...arr1);
// 1
// 2
// 3
```

这里，我们将数组分为三个单独的参数，然后传递给函数。

```
const myFunc = (x1, x2, x3) => {
  console.log(x1);
  console.log(x2);
  console.log(x3);
};
const arr1 = [1, 2, 3];
myFunc(...arr1);
// 1
// 2
// 3
```

向函数传递无限参数

假设我们有一个函数，它接受无限个参数，如下所示:

```
const myFunc = (...args) => {
  console.log(args);
};
```

如果我们现在调用这个带有多个参数的函数，会看到下面的情况:

```
myFunc(1, 'a', new Date());
```

返回:

```
[
  1,
  'a',
  Date {
    __proto__: Date {}
  }
]
```

然后，我们就可以动态地循环遍历参数。

将 nodeList 转换为数组

假设我们使用了展开运算符来获取页面上的所有`div`：

```
const el = [...document.querySelectorAll('div')];
console.log(el);
// (3) [div, div, div]
```

在这里可以看到我们从dom中获得了3个`div`。

现在，我们可以轻松地遍历这些元素，因为它们是数组了。

```
const el = [...document.querySelectorAll('div')];
el.forEach(item => {
  console.log(item);
});
// <div></div>
// <div></div>
// <div></div>
```

解构对象

假设我们有一个对象`user`：

```
const user = {
  firstname: 'Chris',
  lastname: 'Bongers',
  age: 31
};
```

现在，我们可以使用展开运算符将其分解为单个变量。

```
const {firstname, ...rest} = user;
console.log(firstname);
console.log(rest);
// 'Chris'
// { lastname: 'Bongers', age: 31 }
```

这里，我们解构了`user`对象，并将`firstname`解构为`firstname`变量，将对象的其余部分解构为`rest`变量。

展开字符串

展开运算符的最后一个用例是将一个字符串分解成单个单词。

假设我们有以下字符串：

```
const str = 'Hello';
```

然后，如果我们对这个字符串使用展开操作符，我们将得到一个字母数组。

```
const str = 'Hello';
const arr = [...str];
console.log(arr);
// [ 'H', 'e', 'l', 'l', 'o' ]
```