Symbol

1. 概述

ES6 引入 Symbol 的原因是为对象添加独一无二的属性或方法。Symbol 是一种原始数据类型，它属于 JavaScript 语言的原生数据类型之一。

Symbol 值通过 `Symbol()` 函数生成。对象的属性名现在可以有两种类型，一种是原来就有的字符串，另一种就是新增的 Symbol 类型。凡是属性名属于 Symbol 类型，就都是独一无二的，可以保证不会与其他属性名产生冲突。

```
let s = Symbol();
typeof s; // "symbol"
```

`Symbol()` 函数前不能使用 `new` 命令，这是因为生成的 Symbol 是一个原始类型的值，不是对象，所以不能使用 `new` 命令来调用。 另外，由于 Symbol 值不是对象，所以也不能添加属性。基本上，它是一种类似于字符串的数据类型。

```
new Symbol(); // TypeError: Symbol is not a constructor
```

`Symbol()` 函数可以接受一个字符串作为参数，表示对 Symbol 实例的描述。 这主要是为了在控制台显示，或者转为字符串时，比较容易区分。

```
let s1 = Symbol('foo');
let s2 = Symbol('bar');

s1 // Symbol(foo)
s2 // Symbol(bar)

s1.toString() // "Symbol(foo)"
s2.toString() // "Symbol(bar)"
```

上例中，s1 和 s2 是两个 Symbol 值。如果不加参数，它们在控制台的输出都是 `Symbol()`，不利于区分。有了参数以后，就等于为它们加上了描述，输出的时候就能够分清，到底是哪一个值。

如果 Symbol 的参数是一个对象，就会调用该对象的 `toString()` 方法，将其转为字符串，然后才生成一个 Symbol 值。

```
const obj = {
  toString() {
    return 'abc';
  }
};
```

```
const sym = Symbol(obj);
sym // Symbol(abc)
```

`Symbol()` 函数的参数只是表示对当前 Symbol 值的描述，因此相同参数的 Symbol 函数的返回值是不相等的。

```
// 没有参数的情况
let s1 = Symbol();
let s2 = Symbol();
s1 === s2; // false

// 有参数的情况
let s1 = Symbol('foo');
let s2 = Symbol('foo');
s1 === s2; // false
```

Symbol 值不能与其他类型的值进行运算，会报错。

```
let sym = Symbol('My symbol');
"your symbol is " + sym; // TypeError: can't convert symbol to string
`your symbol is ${sym}`; // TypeError: can't convert symbol to string
```

但是，Symbol 值可以显式转为字符串。

```
let sym = Symbol('My symbol');
String(sym); // 'Symbol(My symbol)'
sym.toString(); // 'Symbol(My symbol)'
```

Symbol 值也可以转为布尔值，但是不能转为数值。

```
let sym = Symbol();
Boolean(sym) // true
!sym  // false
!!sym  // true
if (sym) {
  console.log("a"); // a
}

Number(sym) // TypeError: Cannot convert a Symbol value to a number
sym + 2 // TypeError: Cannot convert a Symbol value to a number
```

2. Symbol.prototype.description

Symbol 值的实例属性 `description` 返回 Symbol 值的描述。

```
const sym = Symbol('foo');
sym.description; // "foo"
```

3. Symbol 作为属性名

由于每一个 Symbol 值都是不相等的，这意味着只要 Symbol 值作为标识符，用于对象的属性名，就能保证不会出现同名的属性。这对于一个对象由多个模块构成的情况非常有用，能防止某一个键被不小心改写或覆盖。

```
let s = Symbol();

// 第一种写法
let a = {};
a[s] = 'Hello!';

// 第二种写法
let a = {
  [s]: 'Hello!'
};

// 第三种写法
let a = {};
Object.defineProperty(a, s, { value: 'Hello!' });

// 以上写法都得到同样结果
a[s] // "Hello!"
```

Symbol 值作为对象属性名时，不能用点运算符。

```
const mySymbol = Symbol();
const a = {};

a.mySymbol = 'Hello!'; // 此处 mySymbol 是字符串
a[mySymbol] // undefined, 此处 mySymbol 是 Symbol 类型
a['mySymbol'] // "Hello!", 获取字符串 mySymbol 属性
```

4. 属性名的遍历

Symbol 值作为属性名，遍历对象的时候，该属性不会出现在 `for...in`、`for...of` 循环中，也不会被 `Object.keys()`、`Object.getOwnPropertyNames()`、`JSON.stringify()` 返回。

但是，它也不是私有属性，有一个 `Object.getOwnPropertySymbols()` 方法，可以获取指定对象的所有 Symbol 属性名。该方法返回一个数组，成员是当前对象的所有用作属性名的 Symbol 值。

```
const obj = {};
const foo = Symbol('foo');
obj[foo] = 'bar';

for (let i in obj) {
  console.log(i); // 无输出
}
Object.keys(obj); // []
JSON.stringify(obj); // {}
Object.getOwnPropertyNames(obj) // []
Object.getOwnPropertySymbols(obj) // [Symbol(foo)]
```

上例中，需要使用 `Object.getOwnPropertySymbols()` 方法才能得到对象的所有 Symbol 属性名。

另一个新的 API，`Reflect.ownKeys()` **方法可以返回所有类型的键名，包括常规键名和 Symbol 键名。**

```
let obj = {
  [Symbol('my_key')]: 1,
  enum: 2,
  nonEnum: 3
};

Reflect.ownKeys(obj); //  ["enum", "nonEnum", Symbol(my_key)]
```

5. Symbol.for()，Symbol.keyFor()

如果希望重新使用同一个 Symbol 值，`Symbol.for()` 方法可以做到这一点。它接受一个字符串作为参数，然后搜索有没有以该参数作为名称的 Symbol 值。如果有，就返回这个 Symbol 值，否则就新建一个以该字符串为名称的 Symbol 值，并将其注册到全局。

```
let s1 = Symbol.for('foo');
let s2 = Symbol.for('foo');

typeof s1; // 'symbol'
typeof s2; // 'symbol'

s1; // Symbol(foo)
s2; // Symbol(foo)

s1 === s2 // true
```

Symbol.for()与Symbol()这两种写法，都会生成新的 Symbol。它们的区别是，前者会被登记在全局环境中供搜索，后者不会。`Symbol.for()` 不会每次调用就返回一个新的 Symbol 类型的值，而是会先检查给定的`key` 是否已经存在，如果不存在才会新建一个值。比如，如果调用 `Symbol.for("cat")` 30 次，每次都会返回同一个 Symbol 值，但是调用 `Symbol("cat")` 30 次，会返回 30 个不同的 Symbol 值。

```
Symbol.for("bar") === Symbol.for("bar"); // true
Symbol("bar") === Symbol("bar"); // false
```

上面代码中，由于Symbol()写法没有登记机制，所以每次调用都会返回一个不同的值。

`Symbol.keyFor()` **方法返回一个已登记的 Symbol 类型值的 key。**

```
let s1 = Symbol.for("foo");
Symbol.keyFor(s1); // "foo"

let s2 = Symbol("foo");
Symbol.keyFor(s2); // undefined，变量 s2 属于未登记的 Symbol 值，所以返回 undefined。
s2.description; // "foo"
```

注意，Symbol.for()为 Symbol 值登记的名字，是全局环境的，不管有没有在全局环境运行。

```
function foo() {
  return Symbol.for('bar');
}

const x = foo();
const y = Symbol.for('bar');
console.log(x === y); // true
```

`Symbol.for()` 的这个全局登记特性，可以用在不同的 iframe 或 service worker 中取到同一个值。

```
iframe = document.createElement('iframe');
iframe.src = String(window.location);
document.body.appendChild(iframe);

iframe.contentWindow.Symbol.for('foo') === Symbol.for('foo'); // true
```