Symbol type

all you need to know

Agenda

1. Предварительные сведения
2. Symbol
3. Приведение к строке
4. “Скрытые” свойства
5. Symbol в литеральном объекте
6. Symbol и цикл for…in
7. Global symbols
8. Зачем?
9. Хорошо известные символы
10. Summary
11. Полезные ссылки

1. Предварительные сведения

Примитивы:

```
let x = 1;

let str = 'string';

let bool = true;

let nothing = undefined;

let _null = null;

let _bigInt = 1n;

let symbol = Symbol();
```

Сложные типы:

```
let obj = {
    a: 1,
    str: 'ok'
}

function getMessage(message) {
    return message;
}

let arr = [1, obj, getMessage, 'str'];
```

2.Symbol

Символ (анг. Symbol) — это уникальный и неизменяемый тип данных, который может быть использован как идентификатор для свойств объектов.

```
> let sym1 = Symbol();

  let sym2 = Symbol();

  sym1 === sym2
< false
```

```
let id = Symbol('id');

let green = Symbol('this is green color');
```

```
> let obj = {
      {a: 1}: 1
  }
✕ Uncaught SyntaxError: Unexpected token '{'
> let obj1 = {
      [1,2,3]: 'arr'
  }
✕ Uncaught SyntaxError: Unexpected token ','
> let obj3 = {
      ()=>1: 'func'
  }
✕ Uncaught SyntaxError: Unexpected token '('
```

```
let myVar = new Symbol()
▶ Uncaught TypeError: Symbol is not a constructor
    at new Symbol (<anonymous>)
    at <anonymous>:1:13
```

```
= Symbol("id");
```

3.Приведение к строке

```
> let id = Symbol();
< undefined
> alert(id)
✕ ▶Uncaught TypeError: Cannot convert a Symbol value to a
string
    at <anonymous>:1:1
```

```
> let myVar = Symbol();

  alert(myVar.toString())
< undefined
```

```
> let id = Symbol('id');

  id.description
< "id"
>
```

4. “Скрытые” свойства

```
> let obj = {
      id: 4
  }

  const f1 = obj => obj.id = 55;
<· undefined
> f1(obj)
<· 55
> obj
<· ▶ {id: 55}
```

```
const obj = {
    id: 4
}

const id = Symbol('id');

const f1 = obj => obj[id] = 45;

f1(obj)
45
obj
▶ {id: 4, Symbol(id): 45}
```

5. Symbol в литеральном объекте

```
let id = Symbol();

const o = {
    a: 1,
    [id]: 25
}
undefined

o
{a: 1, Symbol(): 25}
```

```
let id = Symbol();

const o = {
    a: 1,
    id: 25
}
undefined

o
▶ {a: 1, id: 25}
```

6. Symbol и цикл for...in

```
const name = Symbol('my name');

let o = {
    a: 'str',
    id: 25,
    bool: true,
    [name]: 'Vova'
}

for (let el in o) {
    console.log(el);
}
```

```
a
id
bool
```

```
const name = Symbol('my name');

let o = {
    a: 'str',
    id: 25,
    bool: true,
    [name]: 'Vova'
}

Object.keys(o)
▶ (3) ["a", "id", "bool"]
```

```
const name = Symbol('my name');

let o = {
    a: 'str',
    id: 25,
    bool: true,
    [name]: 'Vova'
}

Object.assign({}, o);
▶ {a: "str", id: 25, bool: true, Symbol(my name): "Vova"}
```

```
const name = Symbol('my name');

let o = {
    a: 'str',
    id: 25,
    bool: true,
    [name]: 'Vova'
}

let o2 = {...o}
undefined

o2
▶ {a: "str", id: 25, bool: true, Symbol(my name): "Vova"}
```

7. Global symbols

```
const myName = Symbol.for('Zenin Vladymyr');

Symbol.for('Zenin Vladymyr');

Symbol(Zenin Vladymyr)
```

```
myName === Symbol.for('Zenin Vladymyr');
true
```

```
const myName = Symbol.for('Zenin Vladymyr');
undefined
```

```
Symbol.keyFor(myName)
"Zenin Vladymyr"
```

```
var myObj = {};
var fooSym = Symbol.for('foo');
var otherSym = Symbol.for('foo');
myObj[fooSym] = 'baz';
myObj[otherSym] = 'bing';
"bing"
```

```
fooSym === otherSym
true
```

```
myObj
▶ {Symbol(foo): "bing"}
```

8. Зачем нужен Symbol ?

8. Зачем нужен Symbol ?

```
const id = Symbol('id');

const o = {
    name: 'Bob',
    age: 25,
    [id]: 26
}
undefined
```

```
for (let key in o) {
    console.log(`${key} = ${o[key]}`)
}
```

```
name = Bob
```

```
age = 25
```

```
const o = {
    name: 'Bob',
    age: 25,
    __my_uniq_logs__: 26
}

for (let key in o) {
    console.log(`${key} = ${o[key]}`)
}
```

```
name = Bob
```

```
age = 25
```

```
__my_uniq_logs__ = 26
```

9. «Хорошо известные символы»

Symbol.hasInstance: instanceof

```
> class MyClass {
      static [Symbol.hasInstance](lho) {
          return Array.isArray(lho);
      }
  }
< undefined
> MyClass [Symbol.hasInstance] ([])
< true
> [] instanceof MyClass
< true
>
```

9. «Хорошо известные символы»

Symbol.iterator

```
class Collection {
  *[Symbol.iterator]() {
    let i = 0;
    while(this[i] !== undefined) {
      yield this[i];
      ++i;
    }
  }

}
let myCollection = new Collection();
myCollection[0] = 1;
myCollection[1] = 2;
for(let value of myCollection) {
    console.log(value);
}
```

```
1

2
```

9. «Хорошо известные символы»

Symbol.isConcatSpreadable

```
> x = [1, 2].concat([3, 4], [5, 6], 7, 8);
< ▶ (8) [1, 2, 3, 4, 5, 6, 7, 8]
```

```
class ArrayIsh extends Array {
    get [Symbol.isConcatSpreadable]() {
        return true;
    }
}
class Collection extends Array {
    get [Symbol.isConcatSpreadable]() {
        return false;
    }
}
arrayIshInstance = new ArrayIsh();
arrayIshInstance[0] = 3;
arrayIshInstance[1] = 4;
collectionInstance = new Collection();
collectionInstance[0] = 5;
collectionInstance[1] = 6;
spreadableTest = [1,2].concat(arrayIshInstance).concat(collectionInstance);
▼ (5) [1, 2, 3, 4, Collection(2)] i
    0: 1
    1: 2
    2: 3
    3: 4
  ▶ 4: Collection(2) [5, 6]
    length: 5
  ▶ [[Prototype]]: Array(0)
```

9. «Хорошо известные символы»

Symbol.match

```
class MyMatcher {
    constructor(value) {
        this.value = value;
    }
    [Symbol.match](string) {
        var index = string.indexOf(this.value);
        if (index === -1) {
            return null;
        }
        return [this.value];
    }
}
var fooMatcher = 'foobar'.match(new MyMatcher('foo'));
var barMatcher = 'foobar'.match(new MyMatcher('bar'));
undefined
fooMatcher
▶ ["foo"]
barMatcher
▶ ["bar"]
```

9. «Хорошо известные символы»

Symbol.replace

```
class MyReplacer {
    constructor(value) {
        this.value = value;
    }
    [Symbol.replace](string, replacer) {
        var index = string.indexOf(this.value);
        if (index === -1) {
            return string;
        }
        if (typeof replacer === 'function') {
            replacer = replacer.call(undefined, this.value, string);
        }
        return `${string.slice(0, index)}${replacer}${string.slice(index + this.value.length)}`;
    }
}
var fooReplaced = 'foobar'.replace(new MyReplacer('foo'), 'baz');
var barMatcher = 'foobar'.replace(new MyReplacer('bar'), function () { return 'baz' });
undefined
fooReplaced
"bazbar"
barMatcher
"foobaz"
```

9. «Хорошо известные символы»

Symbol.search

```
class MySearch {
    constructor(value) {
        this.value = value;
    }
    [Symbol.search](string) {
        return string.indexOf(this.value);
    }
}
var fooSearch = 'foobar'.search(new MySearch('foo'));
var barSearch = 'foobar'.search(new MySearch('bar'));
var bazSearch = 'foobar'.search(new MySearch('baz'));
undefined
fooSearch
0
barSearch
3
bazSearch
-1
```

9. «Хорошо известные символы»

Symbol.split

```
class MySplitter {
    constructor(value) {
        this.value = value;
    }
    [Symbol.split](string) {
        var index = string.indexOf(this.value);
        if (index === -1) {
            return string;
        }
        return [string.substr(0, index), string.substr(index + this.value.length)];
    }
}
var fooSplitter = 'foobar'.split(new MySplitter('foo'));
var barSplitter = 'foobar'.split(new MySplitter('bar'));
undefined
fooSplitter
▶ (2) ["", "bar"]
barSplitter
▶ (2) ["foo", ""]
```

9. «Хорошо известные символы»

Symbol.toPrimitive

```
class AnswerToLifeAndUniverseAndEverything {
    [Symbol.toPrimitive](hint) {
        if (hint === 'string') {
            return 'Like, 42, man';
        } else if (hint === 'number') {
            return 42;
        } else {
            // when pushed, most classes (except Date)
            // default to returning a number primitive
            return 42;
        }
    }
}

var answer = new AnswerToLifeAndUniverseAndEverything();
undefined
+answer === 42;
true
Number(answer) === 42;
true
''+answer === 'Like, 42, man';
false
String(answer) === 'Like, 42, man';
true
```

9. «Хорошо известные символы»

Symbol.toStringTag

```
> class Collection {

    get [Symbol.toStringTag]() {
      return 'Collection';
    }

  }
  var x = new Collection();
< undefined
> x.toString() === '[object Collection]'
< true
```

10. Summary

- Ключи символы никогда не будут конфликтовать с ключами строками объекта, благодаря этому они отлично подходят для расширения объектов.
- Символы нельзя прочитать с помощью for...in или Object.keys(obj). Вам нужен новый Object.getOwnPopertySymbols () для доступа к символам объекта, это делает символы отличными для хранения информации, которую вы не хотите, чтобы люди получали при нормальной работе.
- Символы не являются приватными. Не надо хранить в объекте информацию, которую по-настоящему конфиденциальная, с помощью символа - ее можно получить!
- Символы не могут быть преобразованы в примитивы. Это предотвращает случайное преобразование их в строку при установке в качестве имен свойств.

9. Ссылки

1. https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/Symbol
2. https://learn.javascript.ru/symbol
3. https://habr.com/ru/company/ruvds/blog/444340
4. https://nickbulljs.medium.com/%D0%B7%D0%B0%D1%87%D0%B5%D0%BC-%D0%BE%D0%BD%D0%B8-%D0%BD%D1%83%D0%B6%D0%BD%D1%8B-%D0%B2-javascript-symbol-iterator-generator-d5d186b4f1bd
5. https://www.keithcirkel.co.uk/metaprogramming-in-es6-symbols/


СПАСИБО
ЗА ВНИМАНИЕ:D