

JavaScript进阶（ ES6 ）

---ES6新增数据类型和数据结构



河北师范大学软件学院
Software College of Hebei Normal University

内容提纲

- 新增数据类型 (Symbol)
- 新增数据结构 (Set)
- 新增数据结构 (Map)



新增的数据类型Symbol

• 属性名的冲突问题，以及Symbol的提出

- ES5的对象属性名都是字符串，这容易造成属性名的冲突（参见Demo11中属性名冲突案例）
- ES6引入了一种新的原始数据类型Symbol，表示独一无二的值，通过Symbol函数生成
- Symbol变量属于基本数据类型（不是对象），Symbol前不能使用new命令
- 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，主要用于区分变量

```
let s = Symbol();  
typeof s; // "symbol"  
var s1 = Symbol('foo');  
var s2 = Symbol('bar');  
console.log(s1); // Symbol(foo)  
console.log(s2); // Symbol(bar)
```

新增的数据类型Symbol

• Symbol的特点

- Symbol函数的参数只是表示Symbol值的描述，相同参数的Symbol函数的返回值是不相等的
- Symbol变量不能与其他值进行运算，但可转换成字符串类型

```
var s1 = Symbol("foo");    var s1 = Symbol();  
var s2 = Symbol("foo");    var s2 = Symbol();  
s1 === s2 // false        s1 === s2 // false
```

```
var sym = Symbol('My symbol');  
// "your symbol is " + sym; // 报错  
var sym = Symbol('My symbol');  
String(sym); // 'Symbol(My symbol)'  
sym.toString(); // 'Symbol(My symbol)'
```

新增的数据类型Symbol

• 作为属性名的Symbol

- 由于每一个Symbol值都是不相等的，这意味着Symbol值可以作为标识符，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，作为对象属性的具体形式如下

```
var mySymbol = Symbol();
var a = {};
a[mySymbol] = 'Hello!';// 第一种写法
var a = {
  [mySymbol]: 'Hello!'// 第二种写法
};
var a = {};
Object.defineProperty(a, mySymbol, { value: 'Hello!' });// 第三种写法
// 以上写法都得到同样结果
a[mySymbol] // "Hello!"
```


新增的数据类型Symbol

- 作为属性名的Symbol（注意访问属性的方法）

- 区分使用点操作符和中括号操作符时，访问对象属性的不同，Symbol需使用[]，而不是点

```
var mySymbol = Symbol();  
var a = {};  
a.mySymbol = 'Hello!';  
console.log(a[mySymbol]); // undefined  
console.log(a['mySymbol']); // "Hello!"
```

- 使用Symbol值定义属性时，Symbol值须放在方括号之中

```
let s = Symbol();  
let obj = {  
  [s]: function (arg) {console.log(arg);} //若不放中括号内会怎样?  
};  
obj[s](123);
```

新增的数据类型Symbol

• 作为属性名的Symbol的遍历特性

- Symbol作为属性名，该属性不会出现在for...in、for...of循环中
- 也不会被Object.keys()、Object.getOwnPropertyNames()返回，但它也不是私有属性
- 使用Object.getOwnPropertySymbols方法，可以获取指定对象的所有Symbol属性名

```
var obj = {}; var foo = Symbol("foo");
Object.defineProperty(obj, foo, {
  value: "foo bar",
});
for (var i in obj) {
  console.log(i); // 无输出
}
console.log(Object.getOwnPropertyNames(obj)); // []
console.log(Object.getOwnPropertySymbols(obj)); // [Symbol(foo)]
```

新增的数据类型Symbol

•与Symbol变量复用相关的静态方法

- Symbol.for()接受一个字符串作为参数，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，否则就新建并返回一个以该字符串为名称的Symbol值
- Symbol.keyFor()方法返回一个已登记的Symbol类型值的key，字符串类型

```
var s1 = Symbol.for('foo');  
var s2 = Symbol.for('foo');  
console.log(s1 === s2); // true
```

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


内容提纲

- 新增数据类型 (Symbol)
- 新增数据结构 (Set)
- 新增数据结构 (Map)



新增数据结构 (Set)

- ES6提供了新的数据结构Set

- 它类似于数组，但是成员的值都是唯一的，没有重复的值
- 用Set构造函数来生成Set对象，用法类似实例化数组对象，通过new实例化Set对象
- 通过add方法向Set结构加入成员，Set结构不会添加重复的值

```
let s1 = new Set([1,2,3,4,5,5,6,2,2]);  
console.log(s1); //Set(6) {1, 2, 3, 4, 5...}
```

```
var s2 = new Set();  
[2, 3, 5, 4, 5, 2, 2].map(x => s2.add(x));  
for (let i of s2) {  
  console.log(i);  
} // 2 3 5 4
```

新增数据结构 (Set)

• Set的原型属性和方法

- Set.prototype.constructor、Set.prototype.size
- Set.prototype.add(value)、Set.prototype.delete(value)
- Set.prototype.has(value)、Set.prototype.clear()
- Set.prototype.keys() (注意返回的类型)、Set.prototype.values()、Set.prototype.entries()

```
var properties = new Set();
properties.add('width');
properties.add('height');
console.log(properties.size);
if (properties.has('width') && properties.has('height')) {
    console.log("do something!");
}
```

• WeakSet (成员只能是对象且都是弱引用，参阅回收机制)

内容提纲

- 新增数据类型 (Symbol)
- 新增数据结构 (Set)
- 新增数据结构 (Map)



新增数据结构 (Map)

- ES6提供了新的数据结构Map

- 它类似于对象，也是键值对的集合，但是“键”的范围不限于字符串
- Object结构提供了“字符串-值”的对应，Map结构提供了“值-值”的对应
- 创建Map时也可以使用一个数组作为构造参数，此数组的每个元素是键值对的数组

```
var m = new Map();  
var o = {p: 'Hello World'};  
m.set(o, 'content')  
m.get(o); // "content"  
m.has(o); // true  
m.delete(o); // true  
m.has(o); // false
```

```
var o = {}  
var map = new Map([  
  ['name', '张三'],  
  [o, 'Author']  
]);  
map.size // 2  
map.has('name'); // true  
map.get('name'); // "张三"  
map.has(o); // true  
map.get(o); // "Author"
```


新增数据结构 (Map)

• Map的原型属性和方法

- Map.prototype.size、Map.prototype.set(key、value)、Map.prototype.get(key)
- Map.prototype.has(key)、Map.prototype.delete(key)、Map.prototype.clear()
- Map.prototype.keys() (注意返回类型)、Map.prototype.values()、Map.prototype.entries()

```
let map = new Map()
    .set(1, 'a')
    .set(2, 'b');
// get方法读取key对应的键值，如果找不到key，返回undefined
var m = new Map();
var hello = function() {console.log("hello");}
m.set(hello, "Hello ES6!"); // 键是函数
m.get(hello); // Hello ES6!
```

• WeakMap (只接受对象作为键名，弱引用)



Thank You !



河北师范大学软件学院
Software College of Hebei Normal University

ES6 新增的遍历语法for...of

- 你是如何遍历数组中的元素的？
- 当 JavaScript 刚被发布的时候，可能如下这么写

```
for (var index = 0; index < myArray.length; index++) {  
    console.log(myArray[index]);  
}
```

- 从 ES5 开始，你可能使用内置的 forEach 方法

```
myArray.forEach(function (value) {  
    console.log(value);  
});
```

- 缺点：不能通过使用 break 语句退出循环或者使用 return 语句从封闭函数中返回

ES6

- 当使用for...in会如何

```
for (var index in myArray) { // 不要用for...in遍历数组，for...in可用来遍历对象  
    console.log(myArray[index]);  
}
```

- 分配给索引的值为字符串 "0" , "1" 等等，不是真正的数字类型的数字。这样使用是不方便，不符合原意的，比如想的到第2个元素的下一个元素时，你可能不希望得到字符串运算 ("2" + 1 == "21") 的结果

-