# 第六章 集合引用类型

#### 1、与对象一样，在使用数组字面量表示法创建数组不会调用Array构造函数。

#### 2、Array.from()的应用

①把字符串转为数组

console.log(Array.from("Matt")); // ["M", "a", "t", "t"]

②对数组进行浅复制

const a1 = [1, 2, 3, 4];   
const a2 = Array.from(a1);// Array.from()对现有数组进行浅复制  
console.log(a1); // [1, 2, 3, 4]  
alert(a1 === a2); // false

③arguments 对象可以被轻松地转换为数组

function getArgsArray() {  
 return Array.from(arguments);  
 }  
console.log(getArgsArray(1, 2, 3, 4)); // [1, 2, 3, 4]

④from()也能转换带有必要属性的自定义对象，（类数组）

const arrayLikeObject = {  
 0: 1,  
 1: 2,  
 2: 3,  
 3: 4,  
 length: 4  
 };  
 console.log(Array.from(arrayLikeObject)); // [1, 2, 3, 4]

⑤增强数组的值

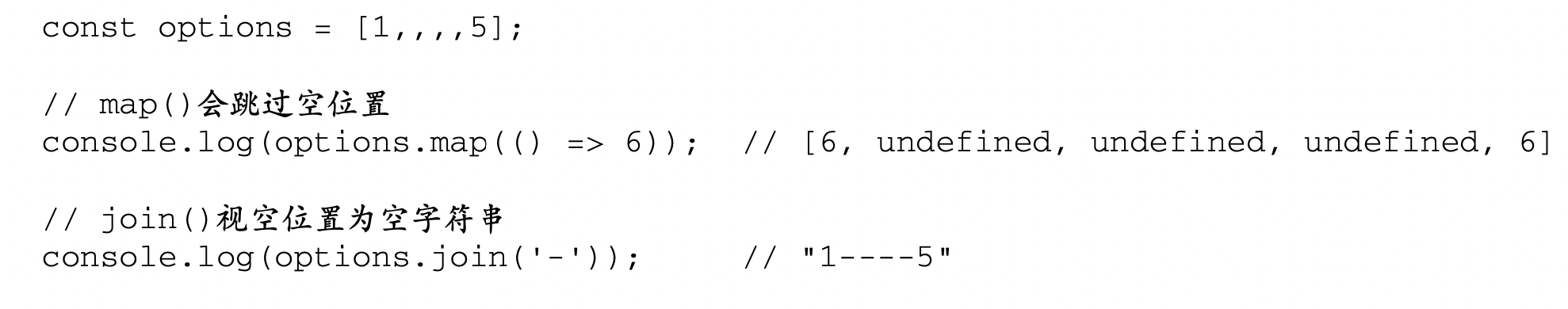
const a1 = [1, 2, 3, 4];  
 const a2 = Array.from(a1, x => x\*\*2);

⑥Array.of()可以把一组参数转换为数组。

console.log(Array.of(1, 2, 3, 4)); // [1, 2, 3, 4]  
console.log(Array.of(undefined)); // [undefined]

#### 3、对于数组空位

实践中要避免使用数组空位。如果确实需要空位，则可以显式地用 undefined 值代替。



#### 4、 Array.isArray()判断是否为一个数组

#### 5、Array的keys()、values()和 entries()方法返回迭代器

const a = ["foo", "bar", "baz", "qux"];  
// 因为这些方法都返回迭代器，所以可以将它们的内容通过Array.from()直接转换为数组实例  
const aKeys = Array.from(a.keys());  
const aValues = Array.from(a.values()); const aEntries = Array.from(a.entries());  
  
console.log(aKeys); // [0, 1, 2, 3]  
console.log(aValues); // ["foo", "bar", "baz", "qux"]   
console.log(aEntries); // [[0, "foo"], [1, "bar"], [2, "baz"], [3, "qux"]]

#### 6、Array的fill()和copyWithin()

const zeroes = [0, 0, 0, 0, 0];  
  
// 用 6 填充索引大于等于 3 的元素 zeroes.fill(6, 3);  
console.log(zeroes); // [0, 0, 0, 6, 6]  
  
// 用7填充索引大于等于1且小于3的元素   
// 第三个参数为结束索引  
zeroes.fill(7, 1, 3);   
console.log(zeroes); // [0, 7, 7, 0, 0];

#### 7、数组的valueOf返回数组本身，toString则对每个值都会调用其 toString()方法，以得到最终的字符串

[1,2,3].toString()  
// '1,2,3'

#### 8、sort()会在每一项上调用 String()转型函数，然后比较字符串来决定顺序。

#### 9、concat的补充

Symbol.isConcatSpreadable属性为false可以阻止 concat()，而为true可以强制打平类数组

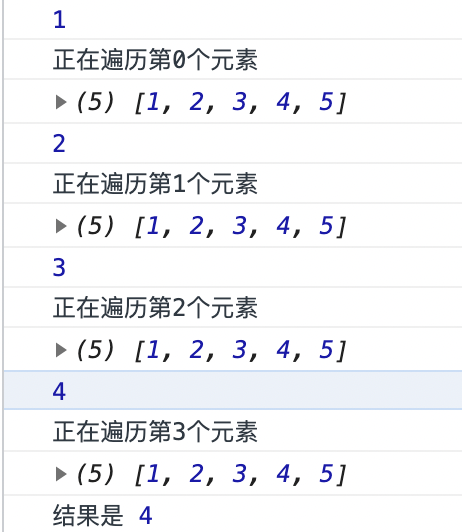
let colors = ["red", "green", "blue"];  
 let newColors = ["black", "brown"];  
 let moreNewColors = {  
 [Symbol.isConcatSpreadable]: true, //为true可以强制打平类数组  
 length: 2,  
 0: "pink",  
 1: "cyan"  
 };  
 newColors[Symbol.isConcatSpreadable] = false;  
 // 强制不打平数组 Symbol.isConcatSpreadable属性为false可以阻止 concat()  
  
  
 let colors2 = colors.concat("yellow", newColors);  
  
 // 强制打平类数组对象  
 let colors3 = colors.concat(moreNewColors);  
  
 console.log(colors); // ["red", "green", "blue"]  
 console.log(colors2); // ["red", "green", "blue", "yellow", ["black", "brown"]]   
 console.log(colors3); // ["red", "green", "blue", "pink", "cyan"]

#### 10、slice参数为负数的情况

如果slice()的参数有负值，那么就以数值长度加上这个负值的结果确定位置。比如，在包含 5 个元素的数组上调用 slice(-2,-1)，就相当于调用 slice(3,4)。如果结 束位置小于开始位置，则返回空数组。

#### 11、数组的find方法

let arr = [1,2,3,4,5];  
let result = arr.find((element,index,array)=>{  
 console.log(element); //第一个参数为遍历的元素  
 console.log(`正在遍历第${index}个元素`); //第二个参数为当前遍历元素索引  
 console.log(array); //整个数组  
 return element>=4  
})  
console.log(result) //找到后就不会往下找了,返回找到的结果



### 12、数组的5个迭代方法

every、some、filter、map、forEach，无个方法中的函数参数一致item,index,array

let numbers = [1, 2, 3, 4, 5, 4, 3, 2, 1];  
let everyResult = numbers.every((item, index, array) => item > 2);  
alert(everyResult); // false  
let someResult = numbers.some((item, index, array) => item > 2);  
alert(someResult); // true

#### 13、数组的归并方法reduce()、reduceRight()

若无传归并起点值，第二个参数，则第一次迭代将从数组的第二项开始

### 14、集合类型Map（映射）

初始化映射

const m1 = new Map([   
 ["key1", "val1"],  
 ["key2", "val2"],  
 ["key3", "val3"]  
 ]);  
 alert(m1.size); // 3

映射的方法get、set、has、clear，size来查询尺寸

const m = new Map();  
 alert(m.has("firstName")); // false  
 alert(m.get("firstName")); // undefined  
 alert(m.size); // 0  
 m.set("firstName", "Matt")  
 .set("lastName", "Frisbie"); //set方法返回映射类型  
 alert(m.has("firstName")); // true  
 alert(m.get("firstName")); // Matt  
 alert(m.size); // 2  
 m.delete("firstName"); // 只删除这一个键/值对  
 alert(m.has("firstName")); // false  
 alert(m.has("lastName")); // true  
 alert(m.size); // 1  
 m.clear(); // 清除这个映射实例中的所有键/值对  
 alert(m.has("firstName")); // false  
 alert(m.has("lastName")); // false  
 alert(m.size); // 0

#### 15、开发中选择**Object**还是**Map**（读一读）

总的来说，影响不大，对于内存和性能来说有以下差异

①内存占用

存储单个键/值对所占用的内存数量都会随键的数量线性增加。

给定固定大小的内存，Map 大约可以比 Object 多存储 50%的键/值对。

② 插入性能

向 Object 和 Map 中插入新键/值对的消耗大致相当，不过插入 Map 在所有浏览器中一般会稍微快 一点儿。对这两个类型来说，插入速度并不会随着键/值对数量而线性增加。如果代码涉及大量插入操 作，那么显然 Map 的性能更佳。

③查找速度

与插入不同，从大型 Object 和 Map 中查找键/值对的性能差异极小，但如果只包含少量键/值对， 则 Object 有时候速度更快。在把 Object 当成数组使用的情况下(比如使用连续整数作为属性)，浏 览器引擎可以进行优化，在内存中使用更高效的布局。这对 Map 来说是不可能的。对这两个类型而言， 查找速度不会随着键/值对数量增加而线性增加。如果代码涉及大量查找操作，那么某些情况下可能选 择 Object 更好一些。

④删除性能

使用 delete 删除 Object 属性的性能一直以来饱受诟病，目前在很多浏览器中仍然如此。为此， 出现了一些伪删除对象属性的操作，包括把属性值设置为undefined或null。但很多时候，这都是一 种讨厌的或不适宜的折中。而对大多数浏览器引擎来说，Map 的 delete()操作都比插入和查找更快。 如果代码涉及大量删除操作，那么毫无疑问应该选择 Map。

#### 16、“弱映射”(WeakMap)

WeakMap 是 Map 的“兄弟”类型，其 API 也是 Map 的子集。

弱映射中的键只能是 Object 或者继承自 Object 的类型，尝试使用非对象设置键会抛出 TypeError。值的类型没有限制。

弱映射的键为弱键，这些键不属于正式的引用， 不会阻止垃圾回收。即在WeakMap里的Object键，不算被引用，所以在wm.set({age:18}, "val")之后，{age:18}被当作垃圾回收。

#### 17、集合Set

Set像是增强的Map

具有has查询，add添加，size获取尺寸，delete删除和clear清空

const s = new Set();  
alert(s.has("Matt"));  
alert(s.size);  
  
s.add("Matt")  
 .add("Frisbie"); //add返回Set类型  
   
alert(s.has("Matt"));  
alert(s.size);  
s.delete("Matt");  
alert(s.has("Matt"));  
alert(s.has("Frisbie")); // true  
alert(s.size); // 1  
  
  
s.clear(); // 销毁集合实例中的所有值  
alert(s.has("Matt")); // false  
alert(s.has("Frisbie")); // false  
alert(s.size); // 0

Set 会维护值插入时的顺序，因此支持按顺序迭代。可以通过 values()， keys()，或者 Symbol.iterator 属性（引用 values()方法）来取得这个迭代器。

const s = new Set(["val1", "val2", "val3"])  
alert(s.values === s.keys) //true  
alert(s.values === s[Symbol.iterator]) //true 都是同一个方法



因为 values()是默认迭代器，所以可以直接对集合实例使用扩展操作，把集合转换为数组

const s = new Set(["val1", "val2", "val3"]);  
console.log([...s]); // ["val1", "val2", "val3"]

比较两个object是否相等

function isEqual(obj1,obj2){  
 for (let key in obj1){  
 if(typeof obj1[key] === 'object'){  
 if (!isEqual(obj1[key],obj2[key])){  
 return false  
 }  
 }  
 else if(obj2.hasOwnProperty(key)){  
 if(obj1[key]!==obj2[key]){  
 return false  
 }  
 }  
 else{  
 return false  
 }  
 }  
 return true  
}  
  
console.log(isEqual({1:1,2:{1:1}},{1:1,2:{1:1}})) //true