集合引用类型

迭代与拓展操作

ES6新增的迭代器和拓展操作符对集合引用类型特别有用
4种原生集合类型定义了默认迭代器

- Array
- 定型数组
- Map
- Set

因为支持迭代所以

- 都能使用for-of 循环
- 可以使用 拓展操作符 ...
- 都支持多种构建方法，Array.of Array.from

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

let map1 = new Map([[1, 2], [3, 4]]);
let map2 = new Map(map1);
console.log(map1); // Map {1 => 2, 3 => 4}
console.log(map2); // Map {1 => 2, 3 => 4}

let arr1 = [1, 2, 3];
let arr2 = [0, ...arr1, 4, 5];
console.log(arr2); // [0, 1, 2, 3, 4, 5]

let arr1 = [{}];
let arr2 = [...arr1];
arr1[0].foo = 'bar';
console.log(arr2[0]); // { foo: 'bar' }

let arr1 = [1, 2, 3];
// 把数组复制到定型数组
let typedArr1 = Int16Array.of(...arr1);
let typedArr2 = Int16Array.from(arr1);
console.log(typedArr1); // Int16Array [1, 2, 3]
console.log(typedArr2); // Int16Array [1, 2, 3]
// 把数组复制到映射
let map = new Map(arr1.map((x) => [x, 'val' + x]));
console.log(map); // Map {1 => 'val 1', 2 => 'val 2', 3 => 'val 3'}
// 把数组复制到集合
let set = new Set(typedArr2);
console.log(set); // Set {1, 2, 3}
// 把集合复制回数组
```

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

总结

JavaScript 中的对象是引用值，可以通过几种内置引用类型创建特定类型的对象。

- 引用类型与传统面向对象编程语言中的类相似，但实现不同
- Object 类型是一个基础类型，所有引用类型都从它继承了基本的行为。
- Array 类型表示一组有序的值，并提供了操作和转换值的能力。
- 定型数组包含一套不同的引用类型，用于管理数值在内存中的类型。
- Date 类型提供了关于日期和时间的信息，包括当前日期和时间以及计算。
- RegExp 类型是 ECMAScript 支持的正则表达式的接口，提供了大多数基本正则表达式以及一些高级正则表达式的能力。

JavaScript 比较独特的一点是，函数其实是 Function 类型的实例，这意味着函数也是对象。由于函数是对象，因此也就具有能够增强自身行为的方法。因为原始值包装类型的存在，所以 JavaScript中的原始值可以拥有类似对象的行为。有 3 种原始值包装类型：Boolean、Number 和 String。它们都具有如下特点。

- 每种包装类型都映射到同名的原始类型。
- 在以读模式访问原始值时，后台会实例化一个原始值包装对象，通过这个对象可以操作数据。
- 涉及原始值的语句只要一执行完毕，包装对象就会立即销毁。

JavaScript 还有两个在一开始执行代码时就存在的内置对象：Global 和 Math。其中，Global 对象在大多数 ECMAScript 实现中无法直接访问。不过浏览器将 Global 实现为 window 对象。所有全局变量和函数都是 Global 对象的属性。Math对象包含辅助完成复杂数学计算的属性和方法。

ECMAScript 6 新增了一批引用类型：Map、WeakMap、Set 和 WeakSet。这些类型为组织应用程序数据和简化内存管理提供了新能力。