**数据结构map**

字典：是用来存储不重复key的hash结构。不同于集合set的是，字典使用的是[键, 值]的形式来储存数据的。

先思考：我们已经有了Object这种数据结构，为什么还要map这种数据结构呢？

javascript的对象（object: {}）只能用字符串当作键。这给它的使用带来了很大的限制。

如：猜猜最终的obj是啥样

let obj1 = {a: 1},

obj2 = {b: 2},

obj = {};

obj[obj1] = 1;

obj[obj2] = 2;

console.log(obj); //

这是因为obj1和obj2在做为key值时，其实是调用了它们的toString方法。都转成了[object Object]这个字符串，所以它的key就一样了，后面的覆盖前面的，所以就只有后面一项添加上。

为了解决这个问题，ES6提供了Map数据结构，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，各种类型的值（包括对象）都可以当做健，也就是说，Object结构提供了‘字符串-值’的对应，map结构提供了‘值-值’的对应，是一种更完善的hash结构实现，如果你需要键值对的数据结构，map比object更合适。

思考：为什么会有map这种数据结构，是因为object它只能以字符串作为属性，所以要创建一种数据结构，可以以引用类型作为属性。Map就是这样的数据结构。

创建：

let map = new Map();

console.log(map);

let map = new Map([

['a', 1],

['b', 2]

]);

console.log(map); // Map { a → 1, b → 2 }

map类的属性：

size：个数

let map = new Map([

['a', 1],

['b', 2]

]);

console.log(map.size); // 2

map类的方法

1. set(key, value) 设置键名key对应的键值为value，然后返回整个map结构。如果key已经有值，则键值会被更新，否则就生成新键值。

由于该方法返回的是map结构，所以可以用链式的方式操作。

let map = new Map([

['a', 1],

['b', 2]

]);

map.set('name', '彭作洪').set('age', 3);

console.log(map); // { a → 1, b → 2, name → "彭作洪", age → 3 }

2、get(key) get方法读取key对应的键值，如果找不到key，返回undefined

let map = new Map([

['a', 1],

['b', 2]

]);

map.set('name', '彭作洪').set('age', 3);

console.log(map); // { a → 1, b → 2, name → "彭作洪", age → 3 }

console.log(map.get('name')); // 彭作洪

console.log(map.get('x')); // undefined

3、delete(key) 删除某个键，如果删除成功，返回true，失败返回false

let map = new Map([

['a', 1],

['b', 2]

]);

console.log(map.delete('a')); // true

console.log(map.delete('x')); // false

console.log(map); // Map { b → 2 }

4、has(key); 方法返回一个布尔值，表示某个键是否在当前的map对象之中

let map = new Map([

['a', 1],

['b', 2]

]);

console.log(map.has('a')); // true

console.log(map.has('x')); // false

5、clear(); 清除所有数据，没有返回值

let map = new Map([

['a', 1],

['b', 2]

]);

map.clear();

console.log(map); // Map { }

6、keys(); 返回键名的遍历器

let map = new Map([

['a', 1],

['b', 2]

]);

console.log(map.keys()); // MapIterator {"a", "b"}

7、values() 返回键值的遍历器

let map = new Map([

['a', 1],

['b', 2]

]);

console.log(map.values()); // MapIterator {1, 2}

8、entries() 返回键值对的遍历器

let map = new Map([

['a', 1],

['b', 2]

]);

console.log(map.entries()); // MapIterator {"a" => 1, "b" => 2}

9、forEach(); 使用回调函数遍历每个成员

let map = new Map([

['a', 1],

['b', 2]

]);

map.forEach(function(value, key, map) {

console.log(value);

console.log(key);

console.log(map);

})

注意事项：

1、map认为NaN是同一个键

let map = new Map([

['a', 1],

['b', 2]

]);

map.set(NaN, 10).set(NaN, 100);

console.log(map); // Map { a → 1, b → 2, NaN → 100 }

2、对象认为是不同的键名

let map = new Map([

['a', 1],

['b', 2]

]);

map.set({}, 10).set({}, 100);

console.log(map); // Map { a → 1, b → 2, {} → 10, {} → 100 }

通过这个例子，我们看出，map的键实际上是个内存地址，只要内存地址不一样，就视为两个键

这样以后扩展别人的库时，我们只要使用对象作为键名，就不会同别人的属性同名了。

3、map里面key的排列顺序是按照添加顺序进行排列的。