第3章 数据结构

所谓数据结构，就是计算机存储和组织数据的方式。说得通俗一点，主要就是指将数据以什么样的结构存储到计算机里面。在程序里面，最为常见的数据结构，就是数组，这种结构将多个数据有序的排列在一起，形成了一个组合。除了数组以外，集合，映射等ES6新增加的数据结构，也会在本章中向大家详细介绍。

本章中我们将学习如下的内容：

- 数组基础知识
- 数组相关属性和方法
- 集合
- 映射

3-1 数组基础

数组是大多数语言里面最常见的一种数据结构，它是一个有序的值列表。

3-1-1 创建数组

创建数组的方式大致可以分为两种：字面量创建数组和使用构造函数创建数组。示例如下：

1. 字面量创建数组

```
let arr = [];
```

2. 构造函数创建数组

```
let arr = new Array();
```

需要注意的是，无论是字面量形式创建的数组，还是构造函数创建的数组，当我们使用 `typeof` 来打印其数据类型的时候，都会返回一个 `object` ，如下：

```
let arr1 = [];
let arr2 = new Array();
console.log(typeof arr1);//object
```

```
console.log(typeof arr2);//object
```

3-1-2 数组赋值

给数组赋值的方法也非常简单，不过可以分为先声明再赋值和声明时直接赋值，如下：

1. 先声明再赋值

```
let arr = [];
arr[0] = 1;
arr[1] = 2;
arr[2] = 3;
```

注意下标是从0开始的。

2. 声明时直接赋值

```
let arr = [1,2,3,4,5];
```

需要注意的是我们可以在数组的任意位置进行赋值，数组的长度会自动改变，空的位置使用undefined来进行填充

```
let arr = [];
arr[0] = 1;
arr[4] = 10;
console.log(arr);
//[ 1, <3 empty items>, 10 ]
```

由于JS是动态语言，所以JS里面数组的数据类型可以是任意类型

```
let arr = [1,"Hello",3.14,true];
```

3-1-3 访问数组元素

通过数组的下标，我们可以轻松的访问到存储在数组里面的元素，如下：

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

> 需要注意数组里面的第一个元素是从下标0开始的。

除了这种常规的访问方式，我们还可以使用变量的方式来进行访问，如下：

```
let arr = [1,2,3,4,5];
let i = 2;
console.log(arr[i]);//3
```

3-1-4 删除元素

我们可以使用delete运算符来删除数组中的某一个元素，示例如下：

```
let arr = [1,2,3,4,5];
delete arr[2];//删除数组中的第3个元素
console.log(arr);
//[ 1, 2, <1 empty item>, 4, 5 ]
```

3-1-5 解构数组

首先我们需要了解什么是解构，所谓解构，就是将一个复杂类型的数据分解为一个普通类型数据。解构是从ES6开始新添加的功能。可以对数组和对象进行解构。这里我们先来看一下数组的解构，如下：

```
let arr = [1,2,3];
let [a,b,c] = arr;
console.log(a);//1
console.log(b);//2
console.log(c);//3
```

这里，就是将arr这个数组里面的值分解给了a,b,c

可以使用逗号来跳过不想要解构的元素，如下：

```
let arr = [1,2,3];
let [a,,b] = arr;
console.log(a);//1
console.log(b);//3
```

在解构出现之前，我们交换两个数需要使用到一个中间变量，但是现在我们可以使用解构来交换

两个数

```
let a = 1,b = 2;
[a,b] = [b,a];
console.log(a);//2
console.log(b);//1
```

关于对象的解构，我们会在后面进行介绍，详细参见对象的相关章节。

3-1-6 多维数组

首先需要说明的是，JS里面不支持传统编程语言中的多维数组。但是，由于JS的数组里面所存放的数据的数据类型可以是任意类型，所以我们可以模拟出多维数组

```
let a = ["Bill","Mary","Lucy"];
let b = [21,24,27];
let c = [a,b];//这里c就是一个多维数组
```

如果要访问多维数组里面的数据，可以使用下面的形式

```
let a = ["Bill","Mary","Lucy"];
let b = [21,24,27];
let c = [a,b];
console.log(c[0][2]);//Lucy
```

利用前面所介绍的解构，我们可以来解构一个多维数组，示例如下：

```
let arr = [[1,2,3],4,5];
let [a,b,c] = arr;
console.log(a);//[1,2,3]
console.log(b);//4
console.log(c);//5
```

3-1-7 扩展运算符

扩展运算符是ES6开始新添加的运算符，用于取出可迭代对象的每一项。这里我们可以用它来快速的展开一个数组

```
let a = ["Bill","Mary","Lucy"];
```

```
let b = [21,24,27];
let c = [...a,...b];
console.log(c);
//[ 'Bill', 'Mary', 'Lucy', 21, 24, 27 ]
```

我们再来看一个例子：

```
let a = [1,2,3];
let b = [...a,4,5,6];
console.log(b);
//[ 1, 2, 3, 4, 5, 6 ]
```

有了这个运算符以后，我们可以使用它将字符串快速转为数组

```
let str = "Hello";
let arr = [...str];
console.log(arr);
//[ 'H', 'e', 'l', 'l', 'o' ]
```

3-2 数组属性和方法

3-2-1 数组相关属性

length：返回数组元素的个数

```
let arr = [1,2,3,4,5];
console.log(arr.length);//5
```

我们利用此属性，可以快速的清空数组，这种方法比使用重新赋值的效率要高些，如下：

```
let arr = [1,2,3,4,5];
arr.length = 0;
console.log(arr);//[]
```

其实我们使用length属性就是可以随意的对数组的长度进行操控

```
let arr = [1,2,3,4,5];
arr.length = 3;
console.log(arr);//[1,2,3]
```

3-2-2 数组相关方法

1. 添加删除方法

数组可以表现得像栈一样。
栈是一种LIFO(Last-In-First-Out)数据结构，这种数据结构的特点是后进先出。
ECMAScript中专门提供了push()和pop()方法，用来实现类似栈的行为。

```
let arr = [];
let i = arr.push("red","blue");
console.log(arr);//[ 'red', 'blue' ]
console.log(i);//2

let j = arr.pop();
console.log(arr);//[ 'red' ]
console.log(j);//blue
```

注意push()方法推入元素的时候是从右往左边推入的，如下：

```
let arr = [];
arr.push("red","blue","pink");
console.log(arr);
//只有一个入口
// -----------------
//| red  blue  pink
// -----------------
```

队列方法的特点是先进先出FIFO(First-In-First-Out)，因为有两个出口。
在ECMAScript中提供了shift()方法，该方法可以去除数组中第一个元素并且返回该元素。
利用shift()和push()方法就可以实现从右往左的队列

```
//从右往左的队列
let arr = [];
arr.push("red","green","pink");
let item = arr.shift();
console.log(item);//red
console.log(arr);//[ 'green', 'pink' ]
```

如果要实现从左往右的队列，可以使用unshift()方法和pop()方法的配合。unshift()方法的作用是在数组开头推入一个元素

```
//从左往右的队列
let arr = [];
arr.unshift("red","green","pink");
let item = arr.pop();
console.log(item);//pink
console.log(arr);//[ 'red', 'green' ]
```

总结：
push()和pop()可以算是一组方法，作用分别是在数组尾部推入和弹出元素。
unshift()和shift()可以算是一组方法，作用分别是在数组头部推入和弹出元素。
推入的时候可以一次性推入多个元素，返回的是新数组的长度。
弹出的时候一次只能弹出一个元素，返回的是弹出的那个元素

2. 操作方法

操作方法里面介绍3个操作方法，分别是concat()，slice()和splice()

concat()：该方法是先创建当前数组的一个副本，然后将接收到的参数添加到副本的末尾，最后返回新构建的数组，而原本的数组不会变化。

```
let arr = [1,2,3];
let arr2 = arr.concat("red","blue");
console.log(arr);//[ 1, 2, 3 ]
console.log(arr2);//[ 1, 2, 3, 'red', 'blue' ]
```

slice()：该方法可以接收一个或者两个参数，代表返回项的起始和结束位置。
一个参数：那就代表起始位置，返回从指定的起始位置到数组末尾的所有项目
两个参数：那就代表从指定的起始位置到指定的末尾位置之间的项，但是不包括结束位置的项目。
注意：slice()方法不会影响原始数组

```
let arr = [1,2,3,4,5,6,7,8,9,10];
//一个参数
let i = arr.slice(3);
console.log(i);//[ 4, 5, 6, 7, 8, 9, 10 ]
console.log(arr);//[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
//两个参数
let j = arr.slice(2,6);
console.log(j);//[ 3, 4, 5, 6 ]
console.log(arr);//[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
```

如果传入的是负数，则用数组长度加上该数来确定相应的位置

```
let arr = [1,2,3,4,5,6,7,8,9,10];
//一个参数
let i = arr.slice(-3);//等价于slice(7)
console.log(i);//[ 8, 9, 10 ]
//两个参数
let j = arr.slice(-6,-2);//等价于slice(4,8)
console.log(j);//[ 5, 6, 7, 8 ]
//不满足条件返回空数组
let k = arr.slice(-2,-6);//等价于slice(8,4)
console.log(k);//[]
```

splice()：这个方法非常的厉害，可以实现对数组的3种类型的操作：删除，插入和替换，相当于是增删改操作都可以用这个方法来实现。
删除：可以删除任意数量的元素，只需要指定2个参数：要参数的第一项位置和要删除的项数。

```
let arr = [1,2,3,4,5,6,7,8,9,10];
//从下标为3的元素开始删除，删除5个元素
//将删除的元素返回给i
let i = arr.splice(3,5);
console.log(i);//[ 4, 5, 6, 7, 8 ]
console.log(arr);//[ 1, 2, 3, 9, 10 ]
```

插入：可以向任意位置插入任意数量的元素。只需要提供3个参数：起始位置，0(要删除的项目)，要插入的项目。

```
let arr = [1,2,3,4,5,6,7,8,9,10];
//从下标为3的元素之前开始插入
let i = arr.splice(3,0,"red","blue");
console.log(i);//[]
console.log(arr);
//[ 1, 2, 3, 'red', 'blue', 4, 5, 6, 7, 8, 9, 10 ]
```

替换：替换的原理在于插入的时候同时进行删除，这样就实现了替换功能

```
let arr = [1,2,3,4,5,6,7,8,9,10];
//从下标为3的元素之前开始插入
//插入多少，刚好就删除多少
let i = arr.splice(3,2,"red","blue");
console.log(i);//[ 4, 5 ]
console.log(arr);
//[ 1, 2, 3, 'red', 'blue', 6, 7, 8, 9, 10 ]
```

数组和字符串相互转换：join() 和 split()
join()：将数组转为字符串，可以传入分隔符作为参数

```
let arr = [1,2,3];
let str = arr.join("");
console.log(str);//123
let str2 = arr.join(",");
console.log(str2);//1,2,3
```

split()：将字符串转为数组，传入参数指明以什么作为分隔符

```
let str = "Hello";
let arr = str.split("");
console.log(arr);//[ 'H', 'e', 'l', 'l', 'o' ]
```

```
let arr2 = str.split("l");
console.log(arr2);//[ 'He', '', 'o' ]
```

3. 数组重排序方法

重排序涉及到两个方法：reverse()和sort()
reverse()：反转数组项的顺序，注意使用该方法时会改变原来数组的顺序，而不是返回一个副本

```
let arr = [1,2,3,4,5];
console.log(arr.reverse());//[ 5, 4, 3, 2, 1 ]
console.log(arr);//[ 5, 4, 3, 2, 1 ]
```

sort()：按照升序排列数组每一项

```
let arr = [0,12,3,7,-12,23];
console.log(arr.sort());
//[ -12, 0, 12, 23, 3, 7 ]
```

可以看到，我们调用sort()方法以后排序并没有正确的按照升序来进行排序。
原因在于：sort()方法排序时首先会调用每个元素的toString()转型方法，然后比较得到的字符串。即使每一项都是数值，sort()方法比较的也是字符串。

解决方法：sort()方法可以接收一个比较函数作为参数，以便我们指定哪个值位于哪个值的前面。比较函数接收两个参数，如果第一个参数应该位于第二个数的前面，返回一个负数，如果两个参数相等，返回0，如果第一个参数应该位于第二个数的后面，返回一个正数。

```
let arr = [0,12,3,7,-12,23];
console.log(arr.sort(function(a,b){
    if(a < b)
    {
        return -1;
    }
    else if(a > b)
    {
        return 1;
    }
    else{
        return 0;
    }
}));
```

如果是要进行降序排列，只需要将返回值进行修改即可

```
let arr = [0,12,3,7,-12,23];
console.log(arr.sort(function(a,b){
    if(a < b)
    {
        return 1;
    }
    else if(a > b)
    {
        return -1;
    }
    else{
        return 0;
    }
}));
```

事实上我们的比较函数还有一种更加简单的书写方法，如下：

```
let arr = [0,12,3,7,-12,23];
console.log(arr.sort(function(a,b){
    return a - b;
    //降序就返回 b - a
}));
```

最后需要注意的是，reverse()和sort()方法，返回值是经过排序之后的数组

4. 位置方法

ECMAScript还为数组提供了两个位置方法：indexOf() 和 lastIndexOf()
这两个方法都接收两个参数：要查找的项目和查找的起点位置索引。区别在于一个是从数组开头开始找，一个是从数组末尾开始找。如果没找到就返回-1

```
let arr = ["H","e","l","l","o"];
console.log(arr.indexOf("l"));//2
console.log(arr.lastIndexOf("l"));//3
console.log(arr.indexOf("z"));//-1
```

还需要注意的是，这两个方法进行查找时使用的是全等进行比较

```
let arr = ["1","2","3"];
console.log(arr.indexOf(1));//-1
```

includes()：用于查看数组里面是否包含某个元素，包含返回true，否则返回false

```
let arr = ["1","2","3"];
console.log(arr.includes(2));//flase
console.log(arr.includes("2"));//true
console.log(arr.includes(7));//false
```

3-3 集合

集合(set)是在ES6中引入的一种数据结构，用于表示唯一值的集合，所以它不能包含重复值。接下来这一小节，就让我们具体来看一下这种新的数据结构。

3-3-1 什么是集合

在ES6标准制定以前，可选的数据结构类型有限，可以说只有数组这种数据结构。而数组使用的又是数值型索引，因而经常被用于来模拟队列和栈的行为。但是如果需要使用非数值型索引，就会用非数组对象创建所需的数据结构，而这就是Set集合与后面一节要介绍的Map映射的早期实现。

Set集合是一种无重复元素的列表，这是这种数据结构的最大的一个特点。

3-3-2 创建集合

要创建一个集合，方法很简单，直接使用new就可以创建一个Set对象。如果想要集合在创建时就包含初始值，那么我们可以传入一个数组进去。

```
let s1 = new Set();
let s2 = new Set([1,2,3]);
console.log(s1);//Set {}
console.log(s2);//Set { 1, 2, 3 }
```

3-3-3 给集合添加值

使用`add()`方法可以给一个集合添加值，由于调用`add()`方法以后返回的又是一个Set对象，所以我们能连续调用`add()`方法进行值的添加，这种像链条一样的方法调用方式被称为链式调用。

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

我们还可以直接将一个数组传入`add()`方法里面

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

但是需要注意的是建立Set对象时传入数组与调用`add()`方法时传入数组是效果是不一样，区别如下：
建立Set对象时传入数组，数组每一项成为Set对象的一个元素

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

调用`add()`方法时传入数组，就是作为Set对象的一个元素

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

在Set对象中，不能够添加相同的元素，这是很重要的一个特性

```
let s1 = new Set();
s1.add(1).add(2).add(2).add(3);
console.log(s1);
//Set { 1, 2, 3 }
```

3-3-4 集合相关属性和方法

1. 用size属性获取元素个数

```
let s1 = new Set([1,2,3]);
console.log(s1.size);//3
```

2. 使用has()方法来查看一个集合中是否包含某一个值

```
let s1 = new Set([1,2,3]);
console.log(s1.has(1));//true
```

```
console.log(s1.has("1"));//false
```

3. 删除集合值

使用delete删除Set对象里面的某一个元素

```
let s1 = new Set([1,2,3]);
s1.delete(2);
console.log(s1);//Set { 1, 3 }
//没有的元素也不会报错
s1.delete("2");
console.log(s1);//Set { 1, 3 }
```

如果要一次性删除所有的元素，可以使用clear方法

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

3-3-5 遍历集合

Set对象无法像数组那样使用传统的for循环来进行遍历，但是我们可以通过for-of或者forEach来进行集合的遍历，示例如下：

```
//使用for-of进行遍历
let s = new Set([1,2,3,4,5]);
for(let value of s)
{
    console.log(value);
}
// 1
// 2
// 3
// 4
// 5
```

```
//使用forEach进行遍历
let s = new Set([1,2,3,4,5]);
s.forEach(ele => console.log(ele));
// 1
// 2
```

```
// 3
// 4
// 5
```

3-3-6 集合转数组

将集合转为数组，最快的方法就是使用前面所讲过的扩展运算符，如下：

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

除此之外，我们还可以使用Array对象所提供的from()方法来进行转换

```
let s1 = new Set([1,2,3]);
console.log(s1);//Set { 1, 2, 3 }
let arr = Array.from(s1);
console.log(arr);//[ 1, 2, 3 ]
```

前面我们有提到过，Set对象里面是不能够存放相同的元素的，利用这个特性，我们可以快速的为数组去重，如下：

```
let arr = [1,2,2,3,4,3,1,6,7,3,5,7];
let s1 = new Set(arr);
let arr2 = [...s1];
console.log(arr2);//[ 1, 2, 3, 4, 6, 7, 5 ]
```

3-3-7 弱集合

当对象添加到集合中时，只要集合存在，它们就一直存储在集合。即使对象的引用被删除了也依然如此，我们来看下面的这个例子：

```
let arr = [1,2,3];
let s = new Set(arr);
arr = null;//删除arr数组的指向
console.log(s);//Set { 1, 2, 3 } 数组依然存在于集合中
console.log(arr);//null
```

可以看到，这里我们删除了对数组的引用，但是该数组依然存在，只不过里面的值为null，这样的话垃圾回收就不会不会回收这个数组，从而可能会引起内存泄漏

什么是内存泄漏?

一个程序里面保留着已经不能在内存中访问的值时，就会发生内存泄露，也就是说占着空间却没用，造成内存的浪费。

例如：

```
let arr = [1,2,3];
arr = null;
```

断开了arr对1，2，3的引用，现在1，2，3在内存里面已经是垃圾了。内存泄露会逐渐减少全部可用内存，导致程序和系统的速度变慢甚至崩溃

那么怎样才能清空这些没用的数据呢？例如上例中的1，2，3。事实上在JavaScript中采用的是动态内存管理技术，比如垃圾回收机制，会自动从内存中删除不再被程序需要的东西。而有些编程语言，例如C++，则是需要程序员手动的管理内存，在某些东西完成任务之后，将其从内存中删除。

那么，集合的问题就在于即使失去了引用，也不会被垃圾回收，这个时候我们可以使用弱集合来避免这种状况。创建弱集合使用`new`运算符和`WeakSet()`

```
let weak = new WeakSet();
```

弱集合无法添加基本数据类型，也就是说无法像集合那样添加简单值进去。

```
let weak = new WeakSet();
weak.add(1);
//TypeError: Invalid value used in weak set
```

除了这个限制以外，弱集合和普通集合还有一些细微的区别，例如无法在创建弱集合时传入一个数组进行初始化。不过弱集合也拥有`has()`，`add()`，`delete()`等方法。

```
let arr = [1,2,3,4,5];
let weak = new WeakSet(arr);
//TypeError: Invalid value used in weak set
//无法在创建弱集合时传入一个数组进行初始化
```

还需要注意一点的是，弱集合是对对象的弱引用，所以不能访问对象里面的值列表。这使得弱集合看上去像是空的，但是并不是空的，证明如下：

```
let weak = new WeakSet();
let arr = [1,2,3];
weak.add(arr);
console.log(weak);//WeakSet {}
console.log(weak.has(arr));//true
```

3-4 映射

映射(Map)也是ES6规范中引入的一种数据结构。这是一种存储键-值对列表很方便的方法，类似于其他编程语言中的词典或者哈希表。这一小节，让我们一起来看一下映射这种数据结构。

3-4-1 什么是映射

JavaScript的对象(Object)，本质上是键值对的集合(Hash结构)，但是传统上只能用字符串当作键。这给它的使用带来了很大的限制。(我们会在第6章学习JavaScript里面的对象)

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)都可以当作键。也就是说，Object结构(对象结构)提供了"字符串—值"的对应，而Map结构提供了"值—值"的对应，是一种更完善的Hash结构的实现。

3-4-2 创建映射

使用`new`关键字与`Map()`构造函数，就可以创建一个空的map对象。如果要向Map映射中添加新的元素，可以调用`set()`方法并分别传入键名和对应值作为两个参数。如果要从集合中获取信息，可以调用`get()`方法。

```
let m = new Map();
m.set("name","xiejie");
m.set("age",18);
console.log(m);
//Map { 'name' => 'xiejie', 'age' => 18 }
console.log(m.get("name"));
//xiejie
```

在对象中，无法用对象作为对象属性的键名。但是在Map映射中，却可以这样做，可以这么说，在Map映射里面可以使用任意数据类型来作为键。

```
let m = new Map();
m.set({},"xiejie");
m.set([1,2,3],18);
m.set(3581,18);
console.log(m);
//Map { {} => 'xiejie', [ 1, 2, 3 ] => 18, 3581 => 18 }
```

传入数组来初始化Map映射

可以向Map构造函数传入一个数组来初始化Map映射，这一点同样与Set集合相似。数组中的每个元素都是一个子数组，子数组中包含一个键值对的键名与值两个元素。因此，整个Map映射中包含的全是这样的两个元素的二维数组

```
let arr = [["name","xiejie"],["age",18]];
let m = new Map(arr);
console.log(m);
//Map { 'name' => 'xiejie', 'age' => 18 }
```

3-4-3 映射相关属性和方法

在设计语言新标准时，委员会为Map映射与Set集合设计了如下3个通用的方法
has(key)：检测指定的键名在Map映射中是否已经存在
delete(key)：从Map映射中移除指定键名及其对应的值
clear()：移除Map映射中的所有键值对
Map映射同样支持size属性，其代表当前集合中包含的键值对数量

```
let arr = [["name","xiejie"],["age",18]];
let m = new Map(arr);
console.log(m);//Map { 'name' => 'xiejie', 'age' => 18 }
console.log(m.size);//2
console.log(m.has("name"));//true
console.log(m.get("name"));//xiejie
m.delete("name");
console.log(m);//Map { 'age' => 18 }
m.clear();
console.log(m);//Map {}
```

3-4-4 映射转为数组

Map结构转为数组结构，比较快速的方法还是使用前面介绍过的扩展运算符 `...` 。

```
let arr = [["name","xiejie"],["age",18]];
let m = new Map(arr);
console.log([...m.keys()]);//[ 'name', 'age' ]
console.log([...m.values()]);//[ 'xiejie', 18 ]
console.log([...m.entries()]);//[ [ 'name', 'xiejie' ], [ 'age', 18 ] ]
console.log([...m]);//[ [ 'name', 'xiejie' ], [ 'age', 18 ] ]
```

或者使用Array对象的from()方法

```
let arr = [["name","xiejie"],["age",18]];
let m = new Map(arr);
console.log(Array.from(m));
//[ [ 'name', 'xiejie' ], [ 'age', 18 ] ]
```

3-4-5 弱映射

弱映射和弱集合很类似，主要是解决存在映射里面的垃圾数据问题，创建弱映射使用`new`运算符以及`WeakMap()`构造器

```
let weakMap = new WeakMap();
```

弱映射和普通映射一样，同样也具有`has()`，`get()`，`set()`，`delete()`等方法。

总结

1. 所谓数据结构，就是计算机存储和组织数据的方式。主要就是指将数据以什么样的结构存储到计算机里面。
2. 数组是最常见的一种数据结构。在JavaScript里面创建数组可以通过字面量和构造函数两种方式来进行创建。
3. 创建数组以后我们就可以对该数组进行赋值，访问和删除等操作。
4. 解构是指将一个复杂类型的数据分解为普通类型的数据，在JavaScript中可以对数组和对象进行解构。
5. JavaScript中并不支持多维数组，但是我们可以利用动态语言的特性模拟出多维数组。
6. ES6中新添加的扩展运算符，可以快速用于取出可迭代对象的每一项。
7. 数组也拥有许多的属性和方法，掌握好这些属性和方法，可以让我们编写程序时事半功倍。
8. 集合(Set)是ES6中新引入的一种数据结构，它最大的特点就是不能包含重复值。
9. 映射(Map)也是ES6中新引入的一种数据结构，它是一种键值对结构，其最大的特点在于键可以是任意数据类型。