## 数组

### 初探数组

我们已经学习了 JavaScript 中的对象的概念。假如我们想表示教室里一排学生的座次，我们或许可以用对象字面量这样写：

let seating = {  
 "1": "Mason Mills",  
 "2": "Rachel Griffiths",  
 "3": "Melissa Duncan",  
 "4": "Marley Hughes",  
 "5": "Charlie Henderson",  
 "6": "Brandon Sharp",  
 "7": "Morgan Woods",  
 "8": "Maddox Andrews",  
 "9": "Lexie Jefferson",  
 "10": "Estelle Bolton"   
};  
  
alert(seating["5"]); // "Charlie Henderson"  
alert(seating["9"]); // "Estelle Bolton"

然后，我们可以使用中括号访问法访问某个具体的序号上的学生姓名。但是这样显然非常不方便：

* 每增加一个学生，就要手动安排一个序号；
* 如果我们要将一个学生移到别的位置，将会非常糟糕：我们要把从这里到那里所有的学生重新排一遍序号。
* 我们不能方便地获取到已经加入到这个列表中的学生数量。

所幸，JavaScript 为我们提供了另外一种结构可以避免上面说的问题，它就是**数组**。

我们只需在数组中写下学生姓名，而无需被序号和获取学生数量等问题所困扰，数组为我们搞定了这一切。你可以这样构造一个数组：

let seating = ["Mason Mills", "Rachel Griffiths", "Melissa Duncan", "Marley Hughes", "Charlie Henderson", "Brandon Sharp", "Morgan Woods", "Maddox Andrews", "Lexie Jefferson", "Estelle Bolton"]；

你猜对了！这种构造形式被称为*数组字面量*。一对中括号 [] 标志着数组字面量的开始与结束，中括号之间的每一项都是数组的一个*成员*，或者叫*元素*。这两种称呼是等价的，本书中会交替出现这两个名词。

我们依然可以通过中括号访问法访问数组元素，不同的是，这里不用写成字符串的形式，我们只需要使用数字来访问进行访问，这种访问方式称为*下标访问法*，每个元素的唯一序号称为*下标*。下标是从 0 开始的，换句话说，第一个元素下标为 0，第二个为 1 ，第三个为 2……以此类推，第 n 个元素下标为 n-1 。如果数组中找不到指定下标的元素，就会得到一个 undefined。

// 你不需要写成这样;  
let student = seating["3"];  
alert(student); // "Melissa Duncan"  
  
// 只需要这样：  
let student = seating[2];  
alert(student); // "Melissa Duncan"  
  
// 找不到指定下标的元素的例子：  
alert(seating[100]); // undefined

如果要移动某个元素的位置，你只需要在构造时将它写在你需要的位置即可。你还可以使用 length 属性来获取数组的长度（即所包含元素的数量）。由于第 n 个元素的下标是 n-1，因此数组中最后一个元素的下标是 length 值减一。

// const seating = ["Mason Mills", "Rachel Griffiths", "Melissa Duncan", "Marley Hughes", "Charlie Henderson", "Brandon Sharp", "Morgan Woods", "Maddox Andrews", "Lexie Jefferson", "Estelle Bolton"]；  
// Morgan Woods 上课会跟 Maddox Andrews 说话，把他挪到第二排吧！  
let seating = ["Mason Mills", "Morgan Woods", "Rachel Griffiths", "Melissa Duncan", "Marley Hughes", "Charlie Henderson", "Brandon Sharp", "Maddox Andrews", "Lexie Jefferson", "Estelle Bolton"]；  
  
alert(seating.length); // 10  
alert(seating[0]); // "Mason Mills"  
alert(seating[1]); // "Morgan Woods"  
alert(seating[seating.length - 1]); // "Estelle Bolton"  
alert(seating[seating.length]); // undefined （超出了数组边界）

就像我们更改一个对象的属性一样，我们也可以用类似的方式更改数组的元素值，只要知道它的下标即可：

alert(seating[0]); // Mason Mills 被点名了  
seating[0] = "Anchimolios"; // 这个位置坐了一个古希腊人！  
alert(seating[0]); // 现在报上名字的是 Anchimolios

可怜的 Mason Mills！他的位置被一个我们都不认识的古希腊人给占了。不过我们不需要为他感到难过，我们会在下文学到如何在不剥夺任何一个学生座位的情况下安排新的学生进入这里。

现在，我们知道这一列有十个学生，但是今天很巧，坐在第四、第五位的 "Melissa Duncan、Marley Hughes 都没来上课，我们可以给他们留两个空位：

let seating = ["Anchimolios", "Morgan Woods", "Rachel Griffiths", , , "Charlie Henderson", "Brandon Sharp", "Maddox Andrews", "Lexie Jefferson", "Estelle Bolton"]；

老师开始点人数了！坐在第一个的是 Anchimolios……咦，是谁坐在第四、第五个来着？老师没看到人，就想不起来他们的名字，只能用 undefined 来表示自己的无奈。

alert(seating[2]); // "Rachel Griffiths"  
alert(seating[3]); // undefined  
alert(seating[4]); // undefined

几分钟后，Melissa Duncan 和 Marley Hughes 气喘吁吁地赶回了他们的座位上，老师终于想起了他们的名字。

seating[3] = "Marley Hughes";  
seating[4] = "Melissa Duncan";

等等……好像有点不对！这两人的位置是不是坐反了？我们记得 Melissa Duncan 明明坐在 Marley Hughes 的前面！老师发现了这个问题，现在要求他们互换座位。于是 Marley Hughes 先腾出了座位，等 Melissa Duncan 从原本属于他的位置上起来，好坐回去。

let temp = seating[3];  
seating[3] = seating[4];  
seating[4] = temp;

Marley Hughes 站起来，站到了一个临时（temp）位置，于是 Melissa Duncan 一步到位，坐到了第四桌，Marley Hughes 终于也回到了他的位置上。但是，这两个小伙子不仅迟到，还搞出了这么些名堂，于是，**他们被老师记住了！**（默哀一秒钟）

现在终于开始正式上课了，同学们开始聚精会神地听讲，只见老师在黑板上写下了 “Array 构造器“几个大字，同学们似乎想起了什么……

note：还记得在上一节中提到的 Object.values Object.keys 和 Object.entries 吗？它们都会得到数组。如果不记得的话，可以回去看一看。

### Array 构造器

在 JavaScript 中，每种类型的值都具有一个*构造器*，数组也不例外。它是一类称为 array 的值。我们已经见到了 number 类型（数字）的构造器 Number，string 类型（字符串）的构造器 String，还有 object 类型（对象）的构造器 Object。那么数组——这类称为 array 类型的值，它的构造器自然叫 Array。虽然我们总是用字面量方式创建这些值，但是了解它们的构造器也是很有必要的。

我们可以创建一个构造器的**实例**，在前文中这种形式就叫做创建实例，但是现在我们要冠以一个更准确的术语：实例化。例如，当我们写下 const o = new Object(); 的时候，我们实例化了 Object 构造器，得到了一个对象。我们也可以如法炮制，实例化 Array 构造器：

let a = new Array();

它创建了一个*空数组*，与这种字面量形式是等价的：

let a = [];

如果我们知道数组应该有多少元素（例如：我们知道有多少学生坐在同一列），可以在实例化 Array 构造器时附加一个数据，以表示创建一个包含这么多元素的数组：

let a = new Array(10);  
alert(a.length); // 10  
  
// 相当于  
let a = [, , , , , , , , , ]; // 嘿，怎么搞的，今天一个人都没回来？

趁着老师发出怒吼之前，我们赶紧往下看！

我们也可以把需要包含的元素写在 Array 构造器的括号里，括号内的东西有个正式名称，叫*实参*。

let a = new Array("Anchimolios", "Morgan Woods", "Rachel Griffiths", "Charlie Henderson");  
alert(a.length); // 4  
  
// 相当于  
let a = ["Anchimolios", "Morgan Woods", "Rachel Griffiths", "Charlie Henderson" ]; // 好吧，至少回来了四个

这时有同学提问：

假如我们要用这种方式创建一个只包含一个数字成员的数组，不就会与创建“包含这个数量的数组”混淆了吗？

这是 JavaScript 设计时一个没考虑周到的地方，我们在使用时应当加以注意。当然我们完全可以直接使用方便的数组字面量来创建数组。

### 插入与移除

我们的数组不会是一成不变的。如果我们有一列学生，我们有可能会在前面、后面或中间安插新学生的座位，如果有学生要离开这一列，我们也需要将他的座位移走。JavaScript 中提供了一些方便的操作，来在数组中实施以上现实中的需求。

JavaScript 提供了三个*数组方法*。方法是一些函数，它以属性的形式被访问，并按规定的方式使用。“方法”一词反映了它们的本质：通过一些固定方式来完成指定的需求。

push 方法用于在数组末尾加入一个元素，并得到数组的新长度。

let list = ["red", "orange", "yellow", "green"];  
alert(list.length); // 4  
list.push("blue"); // 加入 "blue" 这个字符串  
alert(list.length); // 5  
alert(list); // red,orange,yellow,green,blue

我们可以直接查看数组内部所包含的元素，如运行结果所显示的那样，list 数组末尾添加了 "blue" 这个字符串。我们也可以一次多添加几个元素：

alert(list.push("purple", "pink")); // 添加两个元素并显示数组的新长度 7  
alert(list.length); // 7  
alert(list); // red,orange,yellow,green,blue,purple,pink

unshift 方法将一个或多个元素添加到数组的开头，并得到数组的新长度。

let flowers = ["Calendula", "Chrysanthemum", "Cosmos", "Dianthus"];  
alert(flowers.length); // 4  
alert(flowers.unshift("Dahlia")); // 把大丽花加进来，现在长度是 5  
alert(flowers.length); // 是的！就是5！

和 push 类似，它也可以一次添加多个元素。

此外还有一个 concat 方法，它用于将两个数组拼在一起，并得到合并后的新数组。

// 现在有两支来自日本的观光团，他们要进行合并以统一行程。  
let visitors\_1 = ["Jissoji Goro", "Kagawa Nui", "Okamura Nariakira",  
 "Ehara Kiyumi", "Okita Miu", "Tanji Chigusa"];  
let visitors\_2 = ["Abo Akasuki", "Kirishima Kayoko", "Kuwahara Toichi",  
 "Amachi Hisato, Nakayama Norihisa"];  
alert(visitors\_1.length); // 现在第一支观光团有 6 人。  
alert(visitors\_2.length); // 第二支观光团有 4 人。  
let visitors = visitors\_1.concat(visitors\_2); // 合并！  
alert(visitors.length); // 现在新的观光团有十个人。  
alert(visitors); // Jissoji Goro,Kagawa Nui,  
 // Okamura Nariakira,Ehara Kiyumi,  
 // Okita Miu,Tanji Chigusa,...

提供了加入元素的方法，自然也提供了移除元素的途径。

pop 方法与 push 相对，用于移除数组的最后一个元素，并得到这个移除的元素值。

let list = [1, 2, 3, 4, 5];  
alert(list.pop()); // 5  
alert(list); // 1,2,3,4

shift 方法与 unshift 方法相对，用于移除数组的第一个元素，并得到移除的元素值。

let list = [1, 2, 3, 4, 5];  
alert(list.shift()); // 1  
alert(list); // 2,3,4,5

如果操作的数组是什么元素也没有的空数组，那么执行方法时只能得到一个 undefined 值。

let empty = [];  
alert(empty.pop()); // undefined

我们可以利用刚刚了解到的几个数组方法来模拟一个排队的场景。

let queue = []; // 现在还没人开始排队。  
queue.push("Delphium Vitulus");  
queue.push("Palinurus Catullus");  
queue.push("Amatia Laevinus"); // 到现在为止，队伍里有三个人。  
alert("现在队伍情况：" + queue);  
queue.push(prompt("请输入新来者的名字：")); // 新来者的名字由你决定  
alert(queue.shift() + "离开了，大家向前一位。");  
alert("又离开了一位：" + queue.shift());  
queue.push("Talmudia Zosimus");  
queue.push(queue.shift())); // 队伍第一位又重新开始排队  
alert(queue.pop() + "等不住了，走了");  
alert("现在队伍里共有" + queue.length + “人。"");

### 排序与查找