JS 教程 —— 数组和对象

我们上次讲了变量。这是一种用来存单个值的容器。但是如果我们要存多个值呢？比如我们现在需要统计一个班级的人名，或者统计每天的气温，这显然就不能用反复声明大量的变量进行存储。所以我们就要引出本章的主角：数组和对象。

数组

数组是一系列按顺序排列的值的集合。它可以一下子存下一系列的值，就相当于一些按顺序排列的变量。比如我们现在要去超市买东西，依次买了很多物品，并且记录了它们的价格。

数组包裹在一对中括号内 (`[]`)，里面的值按顺序排列，并且使用英文逗号进行分隔 (`,`)。可以通过在数组名后面加上一个 (`[]`)，然后在括号内使用通过**从零开始**的下标来找到对应的值。下面我们给出了一系列的例子。

```
> let shopping = ['bread', 'milk', 'cheese', 'hummus', 'noodles'];
undefined
> shopping; // 打印 shopping 数组
[ 'bread', 'milk', 'cheese', 'hummus', 'noodles' ]
> let price = [ 5.00, 3.50, 12.50, 15.00, 7.50];
undefined
> price; // 打印 price 数组
[ 5, 3.5, 12.5, 15, 7.5 ]
> shopping[0]; // 使用下标进行查找，从 0 开始到比长度小一的值(这里是 4)结束
'bread'
> price[0]; // price 的第一个元素
5
> price[100]; // 越界之后值会变成 undefined
undefined
> shopping[3];
'hummus'
> price[2] = 114; // 可以给对应的下标赋值，这里是 2，就是第三个
114
> price; // 可以看到第三个发生变化了
[ 5, 3.5, 114, 15, 7.5 ]
> shopping[2] = 1234; // 还可以给原来的赋值不同类型，就和普通变量一样
1234
> shopping;
[ 'bread', 'milk', 1234, 'hummus', 'noodles' ]
> let anEmptyArray = []; // 可以先声明一个空的数组，以后再往里面加东西
undefined
> anEmptyArray; // 现在它是空的
```

```
[]
>
```

数组里的值甚至可以是另一个数组，这叫做数组嵌套，下面演示了嵌套数组的行为。

```
> let random = ['tree', 795, [0, 1, 2]]; // 声明一个嵌套的数组
undefined
> random[2][1]; // 访问嵌套数组，第一个[]导航到数组，第二个继续导航
1
> random[0] = ['acacia', 'birch', 'spruce', 'oak']; // 赋值为一个数组
[ 'acacia', 'birch', 'spruce', 'oak' ]
> random; // 现在里面有两个嵌套数组了
[ [ 'acacia', 'birch', 'spruce', 'oak' ], 795, [ 0, 1, 2 ] ]
> random[0][2]; // 同样可以访问
'spruce'
> random[0][3] = ['oak leaves', 'oak log']; // 数组套了三层了
[ 'oak leaves', 'oak log' ]
> random[0][3][1]; // 访问到第三层的数组
'oak log'
>
```

数组方法

length

最常用的数组方法，可能就是获取数组长度了。数组长度可以通过 `length` 方法来获取，就是在数组名之后加上一个 `.length`，例如 `sequence.length` 就可以了解我们下面定义的数组的长度。

```
> let sequence = [1, 1, 2, 3, 5, 8, 13]; // 声明一个数组
undefined
> sequence.length; // 打印数组的长度
7
>
```

length 属性最常用的时候，就是循环遍历一个数组中的所有项目。比如说下面的这个代码：

```
let sequence = [1, 1, 2, 3, 5, 8, 13];
for (let i = 0; i < sequence.length; i = i + 1) {
 console.log(sequence[i]);
}
```

我们以后会详细了解循环，但是这里先稍微提下这里主要干的事情：

- 在数组中的元素编号 0 开始循环。

- 在元素编号等于数组长度的时候停止循环。 这适用于任何长度的数组，但在这种情况下，它将在编号 7 的时候终止循环（还记得数组的编号是从 0 开始的吗？0 到 6 就是 7）。
- 对于每个元素，使用 console.log() 将其打印到浏览器控制台。

字符串和数组之间的转换

有时候你会需要把一个有规律的字符串转化成数组来处理数据，比方说这样的字符串

```
let myData = 'Manchester,London,Liverpool,Birmingham,Leeds,Carlisle';
```

显然，这里是用 , 来分割的一系列单词。如果我们要把它转成一个数组，就可以用 split() 方法，它会对字符串进行处理，然后返回一个数组。方法的用法是在字符串之后加上一个 .split()，在括号内指定分隔符（默认是空格）。

```
> let myData = 'Manchester,London,Liverpool,Birmingham,Leeds,Carlisle';
undefined
> let myArray = myData.split(','); // 转换成数组
undefined
> myArray; // 现在已经转成数组了
[
  'Manchester',
  'London',
  'Liverpool',
  'Birmingham',
  'Leeds',
  'Carlisle'
]
> myArray[2]; // 第三个
'Liverpool'
> myArray[myArray.length - 1]; // 最后一个
'Carlisle'
>
```

如果要把数组转成对应的字符串，可以用相反的方法 .join() ，用法和前面的相同。（假设我们这里是紧跟着上面执行的）

```
> let myNewString = myArray.join(',');
undefined
> myNewString;
'Manchester,London,Liverpool,Birmingham,Leeds,Carlisle'
>
```

添加和删除数组项

这里我们可以使用 push() 和 pop() 方法在**数组尾部**进行添加和删除元素。使用 push() 方法之后会返回新数组的长度，使用 pop() 方法之后会返回被删除的那个值。也许你可以用变量存下它们。

```
> let myArray = ['Manchester', 'London', 'Liverpool', 'Birmingham',
'Leeds', 'Carlisle'];
undefined
> myArray.push('Cardiff'); // 添加一个元素
7
> myArray;
[
  'Manchester',
  'London',
  'Liverpool',
  'Birmingham',
  'Leeds',
  'Carlisle',
  'Cardiff'
]
> myArray.push('Bradford', 'Brighton'); // 添加两个元素
9
> myArray;
[
  'Manchester', 'London',
  'Liverpool',  'Birmingham',
  'Leeds',      'Carlisle',
  'Cardiff',    'Bradford',
  'Brighton'
]
> let newLength = myArray.push('Bristol'); // 存下新的长度
undefined
> myArray;
[
  'Manchester', 'London',
  'Liverpool',  'Birmingham',
  'Leeds',      'Carlisle',
  'Cardiff',    'Bradford',
  'Brighton',   'Bristol'
]
> newLength; // 现在的新长度
10
> myArray.pop(); // 删除最后一个元素
'Bristol'
> let removedItem = myArray.pop(); // 存下删掉的元素
undefined
> myArray;
[
  'Manchester',
  'London',
  'Liverpool',
  'Birmingham',
  'Leeds',
  'Carlisle',
  'Cardiff',
  'Bradford'
]
```

```
> removedItem; // 删掉的元素
'Brighton'
>
```

采取这个方法我们可以模拟一个栈操作。栈是一种 LIFO（先进先出）的数据结构。通俗的讲起来就像是叠盘子，后面的叠在上面，拿的时候就拿走最上面的。最后叠的最先拿出来，就是一个先进先出了。以后我们会单独开章节讲数据结构，当作拓展的内容。

对象

我们刚刚提到了数组，这是一种按顺序存值的方法。它用途非常广泛，但是因为只能按下标找值，在有些时候可能会比较难用。比如现在我们要存一个人的相关信息，比如名字，性别，年龄，简介这些，如果用数组就会遇到一个难题：必须给这些信息确定一个顺序，规定第一个是名字，第二个是性别等等。这样写会带来理解上面的困难（一堆乱七八糟的数字可不是很好阅读的）。如果使用一系列的单一变量去存，会搞出一大堆变量。那么有没有更好的方法呢？答案是使用对象。

对象 (Object) 是包含一系列相关数据和方法的集合（通常由一些变量和函数组成，我们称之为对象里面的属性和方法）。由于我们目前还没有讲函数，我们暂时先讲对象的数值相关的东西，以后在函数部分再提对象的方法。

对象使用一对大括号 (`{}`) 来表示，里面包含着很多用逗号分开的**成员**，每一个成员都拥有一个**名字**（下面的 name, age 这些），和一个**值**（如["Linus", "Torvalds"], 52 都是值）。每一个名字和值(Name and Value)之间由冒号（:）分隔，然后就可以通过成员的名字来查找对应的值。下面是一个例子。为了看得更清楚，我们在其中插入了换行，一行一个成员。我们推荐在以后编程实践当中适当使用换行，因为可以让代码更加整洁。

```
let person = {
  name: ["Linus", "Torvalds"],
  age: 52,
  gender: 'male',
  saying: 'NVIDIA, **** YOU!'
};
```

可以使用**点表示法**来查找对应的值，就是在对象标识符之后加上一个点，在点后面加上键名称就可以了，比如 `person.age` 就可以对应 `person` 的 `age` 对应的值，即 `52` 。

还有另一种表示法是**括号表示法**，用法类似数组，是在一个中括号 `[]` 内加上键名称来对应的，例如 `person["saying"]` 就可以对应到那个 `saying` 。

我们建议使用点表示法，因为更加简洁，方便打字。

下面是在 NodeJS 里面运行的结果，换行之后那三个点是 node 加上的，表示上一句话还没结束。这不是代码的一部分，所以自己写代码的时候不要加上去。

```
> let person = {
...   name: ["Linus", "Torvalds"],
...   age: 52,
...   gender: 'male',
...   saying: 'NVIDIA, **** YOU!'
```

```
... };
undefined
> person;
{
  name: [ 'Linus', 'Torvalds' ],
  age: 52,
  gender: 'male',
  saying: 'NVIDIA, **** YOU!'
}
> person.age; // 点表示法
52
> person['saying']; // 括号表示法
'NVIDIA, **** YOU!'
>
```

子命名空间

可以用一个对象来做另一个对象成员的值。比如我们现在可以把 `person` 的 `name` 从一个数组换成一个对象。就像是下面这样。

```
let person = {
  name: {
    firstName: 'Linus',
    lastName: 'Torvalds'
  },
  age: 52,
  gender: 'male',
  saying: 'NVIDIA, **** YOU!'
};
```

这样就可以用点表示法进行多重的查找，比如 `person.name.firstName` 这样子

```
> let person = {
...   name: {
.....     firstName: 'Linus',
.....     lastName: 'Torvalds'
.....   },
...   age: 52,
...   gender: 'male',
...   saying: 'NVIDIA, **** YOU!'
... };
undefined
> person.name.firstName;
'Linus'
> person.name.lastName;
'Torvalds'
>
```

这样非常有用，因为你可以在对象里面嵌套另一个对象，从而有结构地保存一系列的数据，起到方便阅读和维护的作用。

设置对象成员

有时候我们需要对对象进行一系列的操作。比方说我们可能会修改一个成员，可能会加上一个还没有的成员，或者一个成员不要了我们可以删除它。这些都是可以做到的。

现在我们再创建一次上文中的 `person` 对象，然后对这个变量进行一系列操作：

```
let person = {
  name: ["Linus", "Torvalds"],
  age: 52,
  gender: 'male',
  saying: 'NVIDIA, **** YOU!'
};
```

现在你可以试着修改其中的成员：

```
> person.name = 'Tux'; // 设置名字
'Tux'
> person.age = 25; // 设置年龄
25
> person; // 现在已经变化了
{ name: 'Tux',
  age: 25,
  gender: 'male',
  saying: 'NVIDIA, **** YOU!' }
>
```

你可以删除其中的成员