# 数组进阶

## 创建数组

创建数组的基本方式有两种。**第一种是使用 Array 构造函数**，如下面的代码所示。

|  |
| --- |
| var colors = new Array(); |

如果预先知道数组要保存的项目数量，也可以给构造函数传递该数量，而该数量会自动变成 length属性的值。例如，下面的代码将创建 length 值为 20 的数组。

|  |
| --- |
| var colors = new Array(20); |

也可以向 Array 构造函数传递数组中应该包含的项。以下代码创建了一个包含 3 个字符串值的数组：

|  |
| --- |
| var colors = new Array("red", "blue", "green"); |

当然，给构造函数传递一个值也可以创建数组。但这时候问题就复杂一点了，因为如果传递的是数值，则会按照该数值创建包含给定项数的数组；而如果传递的是其他类型的参数，则会创建包含那个值的只有一项的数组。下面就两个例子：

|  |
| --- |
| var colors = new Array(3); // 创建一个包含 3 项的数组  var names = new Array("Greg"); // 创建一个包含 1 项，即字符串"Greg"的数组 |

另外，在使用 Array 构造函数时也可以省略 new 操作符。如下面的例子所示，省略 new 操作符的结果相同：

|  |
| --- |
| var colors = Array(3); // 创建一个包含 3 项的数组  var names = Array("Greg"); // 创建一个包含 1 项，即字符串"Greg"的数组 |

创建数组的**第二种基本方式是使用数组字面量表示法**。数组字面量由一对包含数组项的方括号表示，多个数组项之间以逗号隔开，如下所示：

|  |
| --- |
| var colors = ["red", "blue", "green"]; // 创建一个包含 3 个字符串的数组  var names = []; // 创建一个空数组 |

在读取和设置数组的值时，要使用方括号并提供相应值的基于 0 的数字索引，如下所示：

|  |
| --- |
| var colors = ["red", "blue", "green"]; // 定义一个字符串数组  alert(colors[0]); // 显示第一项  colors[2] = "black"; // 修改第三项  colors[3] = "brown"; // 新增第四项 |

方括号中的索引表示要访问的值。如果索引小于数组中的项数，则返回对应项的值，就像这个例子中的 colors[0]会显示"red"一样。设置数组的值也使用相同的语法，但会替换指定位置的值。如果设置某个值的索引超过了数组现有项数，如这个例子中的 colors[3]所示，数组就会自动增加到该索引值加 1 的长度（就这个例子而言，索引是 3，因此数组长度就是 4）。

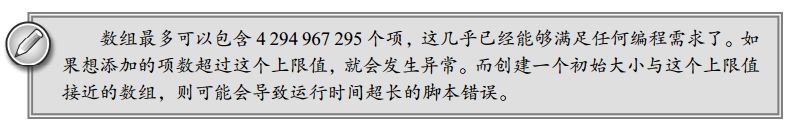
数组的项数保存在其 length 属性中，这个属性始终会返回 0 或更大的值，如下面这个例子所示：

|  |
| --- |
| var colors = ["red", "blue", "green"]; // 创建一个包含 3 个字符串的数组  var names = []; // 创建一个空数组  alert(colors.length); //3  alert(names.length); //0 |

**数组的 length 属性很有特点——它不是只读的**。因此，通过设置这个属性，可以从数组的末尾移除项或向数组中添加新项。

|  |
| --- |
| var colors = ["red", "blue", "green"]; // 创建一个包含 3 个字符串的数组  colors.length = 2;  alert(colors[2]); //undefined  var colors = ["red", "blue", "green"]; // 创建一个包含 3 个字符串的数组  colors.length = 4;  alert(colors[3]); //undefined  var colors = ["red", "blue", "green"]; // 创建一个包含 3 个字符串的数组  colors[colors.length] = "black"; //（在位置 3）添加一种颜色  colors[colors.length] = "brown"; //（在位置 4）再添加一种颜色 |

**利用 length 属性也可以方便地在数组末尾添加新项**，如下所示：



## 检测数组

ECMAScript 5 新增了 **Array.isArray()**方法。这个方法的目的是最终确定某个值到底是不是数组，而不管它是在哪个全局执行环境中创建的。这个方法的用法如下。

|  |
| --- |
| if (Array.isArray(value)){  //对数组执行某些操作  } |

## 转换方法

所有对象都具有 toLocaleString()、 toString()和 valueOf()方法。其中，调用数组的 toString()方法会返回由数组中每个值的字符串形式拼接而成的一个以逗号分隔的字符串。而

调用 valueOf()返回的还是数组。实际上，为了创建这个字符串会调用数组每一项的 toString()方法。来看下面这个例子。

|  |
| --- |
| var colors = ["red", "blue", "green"]; // 创建一个包含 3 个字符串的数组  alert(colors.toString());  alert(colors.valueOf());  alert(colors); |

## 栈方法

ECMAScript 数组也提供了一种让数组的行为类似于其他数据结构的方法。具体说来，数组可以表现得就像栈一样，后者是一种可以限制插入和删除项的数据结构。栈是一种 LIFO（Last-In-First-Out，后进先出）的数据结构，也就是最新添加的项最早被移除。而栈中项的插入（叫做推入）和移除（叫做弹出），只发生在一个位置——栈的顶部。 ECMAScript 为数组专门提供了 push()和 pop()方法，以便实现类似栈的行为。

push()方法可以接收任意数量的参数，把它们逐个添加到数组末尾，并返回修改后数组的长度。而pop()方法则从数组末尾移除最后一项，减少数组的 length 值，然后返回移除的项。

|  |
| --- |
| var colors = new Array(); // 创建一个数组  var count = colors.push("red", "green"); // 推入两项  alert(count); //2  count = colors.push("black"); // 推入另一项  alert(count); //3  var item = colors.pop(); // 取得最后一项  alert(item); //"black"  alert(colors.length); //2 |

## 队列方法

栈数据结构的访问规则是 LIFO（后进先出），而队列数据结构的访问规则是 FIFO（First-In-First-Out，先进先出）。队列在列表的末端添加项，从列表的前端移除项。由于 push()是向数组末端添加项的方法，因此要模拟队列只需一个从数组前端取得项的方法。实现这一操作的数组方法就是 shift()，它能够移除数组中的第一个项并返回该项，同时将数组长度减 1。结合使用 shift()和 push()方法，可以像使用队列一样使用数组。

|  |
| --- |
| var colors = new Array(); //创建一个数组  var count = colors.push("red", "green"); //推入两项  alert(count); //2  count = colors.push("black"); //推入另一项  alert(count); //3  var item = colors.shift(); //取得第一项  alert(item); //"red"  alert(colors.length); //2 |

ECMAScript 还为数组提供了一个 unshift()方法。顾名思义，unshift()与 shift()的用途相反：它能在数组前端添加任意个项并返回新数组的长度。因此，同时使用 unshift()和 pop()方法，可以从相反的方向来模拟队列，即在数组的前端添加项，从数组末端移除项，如下面的例子所示。

|  |
| --- |
| var colors = new Array(); //创建一个数组  var count = colors.unshift("red", "green"); //推入两项  alert(count); //2  count = colors.unshift("black"); //推入另一项  alert(count); //3  var item = colors.pop(); //取得最后一项  alert(item); //"green"  alert(colors.length); //2 |

## 重排序方法

数组中已经存在两个可以直接用来重排序的方法：**reverse()和sort()。**

reverse()方法会反转数组项的顺序。请看下面这个例子。

|  |
| --- |
| var values = [1, 2, 3, 4, 5];  values.reverse();  alert(values); //5,4,3,2,1 |

这里数组的初始值及顺序是1、2、3、4、5。而调用数组的reverse()方法后，其值的顺序变成了5、4、3、2、1。这个方法的作用相当直观明了，但不够灵活，因此才有了sort()方法。

**在默认情况下，sort()方法按升序排列数组项**——即最小的值位于最前面，最大的值排在最后面。为了实现排序，sort()方法会调用每个数组项的toString()转型方法，然后比较得到的字符串，以确定如何排序。**即使数组中的每一项都是数值，sort()方法比较的也是字符串**，如下所示。

|  |
| --- |
| var values = [0, 1, 5, 10, 15];  values.sort();  alert(values); //0,1,10,15,5 |

可见，即使例子中值的顺序没有问题，但sort()方法也会根据测试字符串的结果改变原来的顺序。因为数值5 虽然小于10，但在进行字符串比较时，"10"则位于"5"的前面，于是数组的顺序就被修改了。不用说，这种排序方式在很多情况下都不是最佳方案。因此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之前则返回一个负数，如果两个参数相等则返回0，如果第一个参数应该位于第二个之后则返回一个正数。以下就是一个简单的比较函数：

|  |
| --- |
| function compare(value1, value2) {  if (value1 < value2) {  return -1;  } else if (value1 > value2) {  return 1;  } else {  return 0;  }  } |

这个比较函数可以适用于大多数数据类型，只要将其作为参数传递给sort()方法即可，如下面这个例子所示。

|  |
| --- |
| var values = [0, 1, 5, 10, 15];  alert(values); //0,1,5,10,15 |

在将比较函数传递到sort()方法之后，数值仍然保持了正确的升序。当然，也可以通过比较函数产生降序排序的结果，只要交换比较函数返回的值即可。

|  |
| --- |
| function compare(value1, value2) {  if (value1 < value2) {  return 1;  } else if (value1 > value2) {  return -1;  } else {  return 0;  }  }  var values = [0, 1, 5, 10, 15];  values.sort(compare);  alert(values); // 15,10,5,1,0 |

在这个修改后的例子中，比较函数在第一个值应该位于第二个之后的情况下返回1，而在第一个值应该在第二个之前的情况下返回1。交换返回值的意思是让更大的值排位更靠前，也就是对数组按照降序排序。当然，如果只想反转数组原来的顺序，使用reverse()方法要更快一些。

对于数值类型或者其valueOf()方法会返回数值类型的对象类型，可以使用一个更简单的比较函数。这个函数只要用第二个值减第一个值即可。

|  |
| --- |
| function compare(value1, value2){  return value2 - value1;  } |

由于比较函数通过返回一个小于零、等于零或大于零的值来影响排序结果，因此减法操作就可以适当地处理所有这些情况。

## 操作方法

### concat()

ECMAScript 为操作已经包含在数组中的项提供了很多方法。其中，concat()方法可以基于当前数组中的所有项创建一个新数组。具体来说，这个方法会先创建当前数组一个副本，然后将接收到的参数添加到这个副本的末尾，最后返回新构建的数组。在没有给concat()方法传递参数的情况下，它只是复制当前数组并返回副本。如果传递给concat()方法的是一或多个数组，则该方法会将这些数组中的每一项都添加到结果数组中。如果传递的值不是数组，这些值就会被简单地添加到结果数组的末尾。

|  |
| --- |
| var colors = ["red", "green", "blue"];  var colors2 = colors.concat("yellow", ["black", "brown"]);  alert(colors); //red,green,blue  alert(colors2); //red,green,blue,yellow,black,brown |

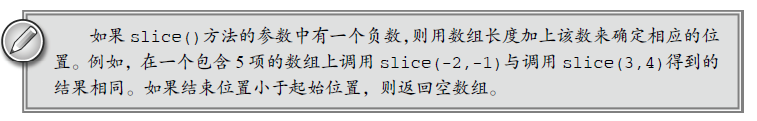
以上代码开始定义了一个包含3 个值的数组colors。然后，基于colors 调用了concat()方法，并传入字符串"yellow"和一个包含"black"和"brown"的数组。最终，结果数组colors2 中包含了"red"、"green"、"blue"、"yellow"、"black"和"brown"。至于原来的数组colors，其值仍然保持不变。

### slice()

下一个方法是slice()，它能够基于当前数组中的一或多个项创建一个新数组。slice()方法可以接受一或两个参数，即要返回项的起始和结束位置。在只有一个参数的情况下，slice()方法返回从该参数指定位置开始到当前数组末尾的所有项。如果有两个参数，该方法返回起始和结束位置之间的项——但不包括结束位置的项。注意，slice()方法不会影响原始数组。请看下面的例子。

|  |
| --- |
| var colors = ["red", "green", "blue", "yellow", "purple"];  var colors2 = colors.slice(1);  var colors3 = colors.slice(1,4);  alert(colors2); //green,blue,yellow,purple  alert(colors3); //green,blue,yellow |

在这个例子中，开始定义的数组colors 包含5 项。调用slice()并传入1 会得到一个包含4 项的新数组；因为是从位置1 开始复制，所以会包含"green"而不会包含"red"。这个新数组colors2 中包含的是"green"、"blue"、"yellow"和"purple"。接着，我们再次调用slice()并传入了1 和4，表示复制从位置1 开始，到位置3 结束。结果数组colors3 中包含了"green"、"blue"和"yellow"。



### splice()

这个方法恐怕要算是最强大的数组方法了，它有很多种用法。

splice()的主要用途是向数组的中部插入项，但使用这种方法的方式则有如下3 种。

* **删除**：可以删除任意数量的项，只需指定2 个参数：要删除的第一项的位置和要删除的项数。例如，splice(0,2)会删除数组中的前两项。
* **插入**：可以向指定位置插入任意数量的项，只需提供3 个参数：起始位置、0（要删除的项数）和要插入的项。如果要插入多个项，可以再传入第四、第五，以至任意多个项。例如，splice(2,0,"red","green")会从当前数组的位置2 开始插入字符串"red"和"green"。
* **替换**：可以向指定位置插入任意数量的项，且同时删除任意数量的项，只需指定3 个参数：起始位置、要删除的项数和要插入的任意数量的项。插入的项数不必与删除的项数相等。例如，splice (2,1,"red","green")会删除当前数组位置2 的项，然后再从位置2 开始插入 字符串"red"和"green"。

splice()方法始终都会返回一个数组，该数组中包含从原始数组中删除的项（如果没有删除任何项，则返回一个空数组）。下面的代码展示了上述3 种使用splice()方法的方式。

|  |
| --- |
| var colors = ["red", "green", "blue"];  var removed = colors.splice(0,1); // 删除第一项  alert(colors); // green,blue  alert(removed); // red，返回的数组中只包含一项  removed = colors.splice(1, 0, "yellow", "orange"); // 从位置1 开始插入两项  alert(colors); // green,yellow,orange,blue  alert(removed); // 返回的是一个空数组  removed = colors.splice(1, 1, "red", "purple"); // 插入两项，删除一项  alert(colors); // green,red,purple,orange,blue  alert(removed); // yellow，返回的数组中只包含一项 |

## 位置方法

ECMAScript 5 为数组实例添加了两个位置方法：indexOf()和lastIndexOf()。这两个方法都接收两个参数：要查找的项和（可选的）表示查找起点位置的索引。其中，indexOf()方法从数组的开头（位置0）开始向后查找，lastIndexOf()方法则从数组的末尾开始向前查找。

这两个方法都返回要查找的项在数组中的位置，或者在没找到的情况下返回1。在比较第一个参数与数组中的每一项时，会使用全等操作符；也就是说，要求查找的项必须严格相等（就像使用===一样）。

|  |
| --- |
| var numbers = [1,2,3,4,5,4,3,2,1];  alert(numbers.indexOf(4)); //3  alert(numbers.lastIndexOf(4)); //5  alert(numbers.indexOf(4, 4)); //5  alert(numbers.lastIndexOf(4, 4)); //3  var person = { name: "Nicholas" };  var people = [{ name: "Nicholas" }];  var morePeople = [person];  alert(people.indexOf(person)); //-1  alert(morePeople.indexOf(person)); //0 |

## 迭代方法

ECMAScript 5 为数组定义了5 个迭代方法。**每个方法都接收两个参数**：要在每一项上运行的函数和（可选的）运行该函数的作用域对象——影响this 的值。传入这些方法中的函数会接收三个参数：**数组项的值、该项在数组中的位置和数组对象本身**。根据使用的方法不同，这个函数执行后的返回值可能会也可能不会影响方法的返回值。以下是这5 个迭代方法的作用。

* every()：对数组中的每一项运行给定函数，如果该函数对每一项都返回true，则返回true。
* filter()：对数组中的每一项运行给定函数，返回该函数会返回true 的项组成的数组。
* forEach()：对数组中的每一项运行给定函数。这个方法没有返回值。
* map()：对数组中的每一项运行给定函数，返回每次函数调用的结果组成的数组。
* some()：对数组中的每一项运行给定函数，如果该函数对任一项返回true，则返回true。

以上方法都不会修改数组中的包含的值。

在这些方法中，最相似的是every()和some()，它们都用于查询数组中的项是否满足某个条件。

**对every()来说，传入的函数必须对每一项都返回true，这个方法才返回true；否则，它就返回false。而some()方法则是只要传入的函数对数组中的某一项返回true，就会返回true。**

|  |
| --- |
| var numbers = [1,2,3,4,5,4,3,2,1];  var everyResult = numbers.every(function(item, index, array){  return (item > 2);  });  alert(everyResult); //false  var someResult = numbers.some(function(item, index, array){  return (item > 2);  });  alert(someResult); //true |

filter()函数，它利用指定的函数确定是否在返回的数组中包含某一项。例如，要

返回一个所有数值都大于2 的数组，可以使用以下代码。

|  |
| --- |
| var numbers = [1,2,3,4,5,4,3,2,1];  var filterResult = numbers.filter(function(item, index, array){  return (item > 2);  });  alert(filterResult); //[3,4,5,4,3] |

这里，通过调用filter()方法创建并返回了包含3、4、5、4、3 的数组，因为传入的函数对它们每一项都返回true。**这个方法对查询符合某些条件的所有数组项非常有用。**

map()也返回一个数组，而这个数组的每一项都是在原始数组中的对应项上运行传入函数的结果。

例如，可以给数组中的每一项乘以2，然后返回这些乘积组成的数组，如下所示。

|  |
| --- |
| var numbers = [1,2,3,4,5,4,3,2,1];  var mapResult = numbers.map(function(item, index, array){  return item \* 2;  });  alert(mapResult); //[2,4,6,8,10,8,6,4,2] |

以上代码返回的数组中包含给每个数乘以2 之后的结果。这个方法适合创建包含的项与另一个数组一一对应的数组。

最后一个方法是forEach()，它只是对数组中的每一项运行传入的函数。这个方法没有返回值，

本质上与使用for 循环迭代数组一样。来看一个例子。

|  |
| --- |
| var numbers = [1,2,3,4,5,4,3,2,1];  numbers.forEach(function(item, index, array){  //执行某些操作  }); |

## 归并方法

ECMAScript 5 还新增了两个归并数组的方法：reduce()和reduceRight()。这两个方法都会迭代数组的所有项，然后构建一个最终返回的值。其中，reduce()方法从数组的第一项开始，逐个遍历到最后。而reduceRight()则从数组的最后一项开始，向前遍历到第一项。

这两个方法都接收**两个参数：**一个在每一项上调用的函数和（可选的）作为归并基础的初始值。传给reduce()和reduceRight()的函数接收4 个参数：**前一个值、当前值、项的索引和数组对象。这个函数返回的任何值都会作为第一个参数自动传给下一项。**第一次迭代发生在数组的第二项上，因此第一个参数是数组的第一项，第二个参数就是数组的第二项。

使用reduce()方法可以执行求数组中所有值之和的操作，比如：

|  |
| --- |
| var values = [1,2,3,4,5];  var sum = values.reduce(function(prev, cur, index, array){  return prev + cur;  });  alert(sum); //15 |

第一次执行回调函数，prev 是1，cur 是2。第二次，prev 是3（1 加2 的结果），cur 是3（数组的第三项）。这个过程会持续到把数组中的每一项都访问一遍，最后返回结果。

reduceRight()的作用类似，只不过方向相反而已。来看下面这个例子。

|  |
| --- |
| var values = [1,2,3,4,5];  var sum = values.reduceRight(function(prev, cur, index, array){  return prev + cur;  });  alert(sum); //15 |

在这个例子中，第一次执行回调函数，prev 是5，cur 是4。当然，最终结果相同，因为执行的都是简单相加的操作。

使用reduce()还是reduceRight()，主要取决于要从哪头开始遍历数组。除此之外，它们完全相同。