



# 第13课：数组格式深入讲解

主讲老师：万章



## 四知

数组的创建和检测

数组的栈与队列操作

数组的高级方法



# 数组的建立和检测

## 数组的建立之构造函数

ECMAScript 数组的每一项可以保存任何类型的数据。也就是说，可以用数组的第一个位置来保存字符串，用第二个位置来保存数值，用第三个位置来保存对象，以此类推。

而且，ECMAScript 数组的大小是可以动态调整的，即可以随着数据的添加自动增长以容纳新增数据。

```
var colors = new Array();
```

使用Array 构造函数

```
var colors = new Array(20);
```

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

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

也可以向Array 构造函数传递数组中应该包含的项。

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

```
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
```

## 数组的建立之数组字面量

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

```
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
var values = [1, 2, ]; // 不要这样! 这样会创建一个包含2 或3 项的数组
var options = [, , , , , ]; // 不要这样! 这样会创建一个包含5 或6 项的数组
```

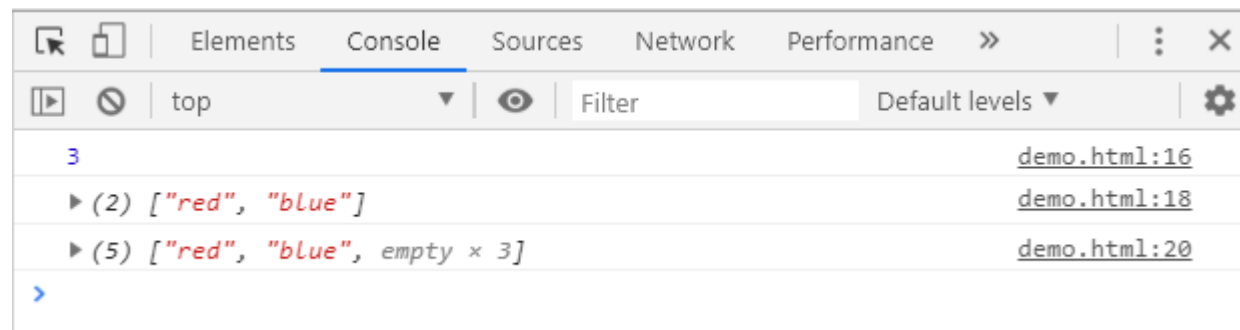
**注意点:**如果使用数组字面量的方式创建数组的话, 那么用逗号隔开的各个数组项的最后一个, 最好不要是空的, 这样js解析引擎不同的版本会展示完全不同的结果

## 数组的建立之特殊情况

```
var colors = ["red", "blue", "green"]; // 创建一个包含3 个字符串的数组
console.log(colors.length); // 输出3
colors.length=2;
console.log(colors); // 输出["red", "blue"];
colors.length=5;
console.log(colors); // 输出["red", "blue", empty, empty, empty];
```



edge浏览器



谷歌浏览器

假设数组的长度为a

那么我们给数组设置一个length的属性值b, 那么当

$b < a$ 时, 就把数组在第b个以后的项目删除

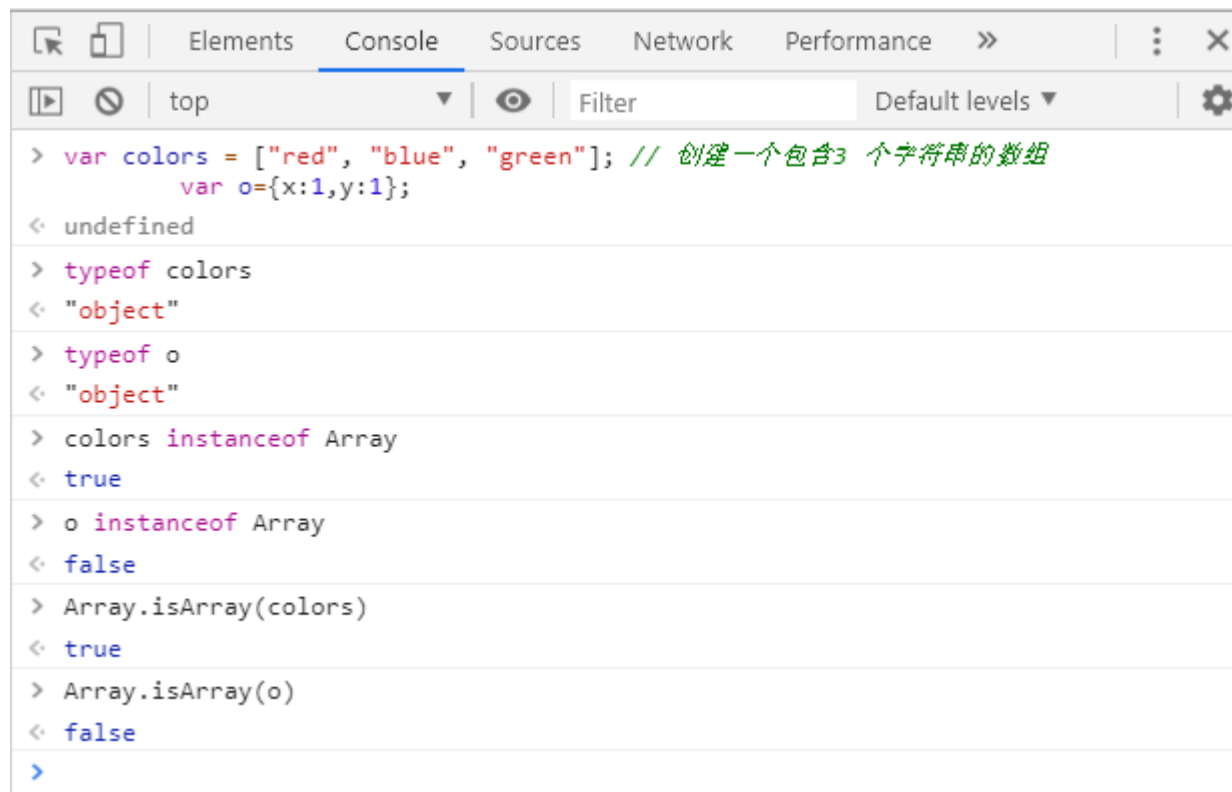
$b > a$ 时, 就把数组在第a个之后再增加(b-a)个空项目

# 数组的检测

**typeof**操作符把数组归类到了对象中, 所以要判断一个数据是对象还是数组我们需要新的操作符来实现

**instanceof** 操作符

**Array.isArray()**



```
> var colors = ["red", "blue", "green"]; // 创建一个包含3 个字符串的数组
    var o={x:1,y:1};
< undefined
> typeof colors
< "object"
> typeof o
< "object"
> colors instanceof Array
< true
> o instanceof Array
< false
> Array.isArray(colors)
< true
> Array.isArray(o)
< false
>
```

**Array.isArray(待检测变量)**

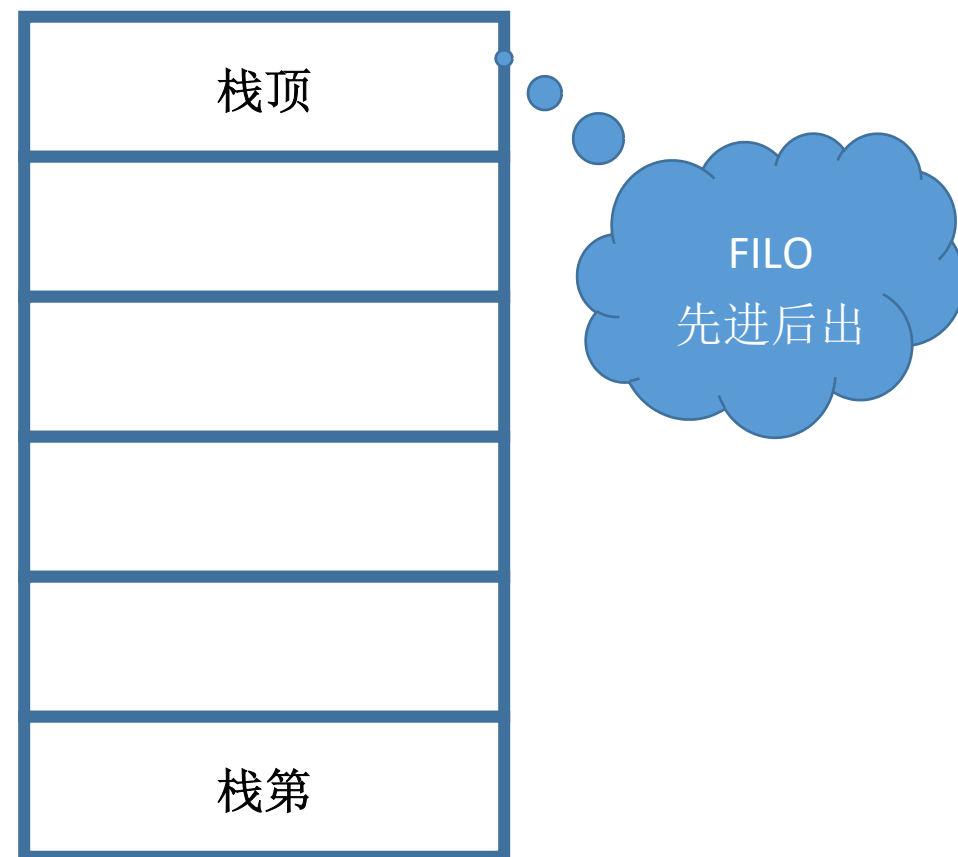
待检测变量    instanceof    Array



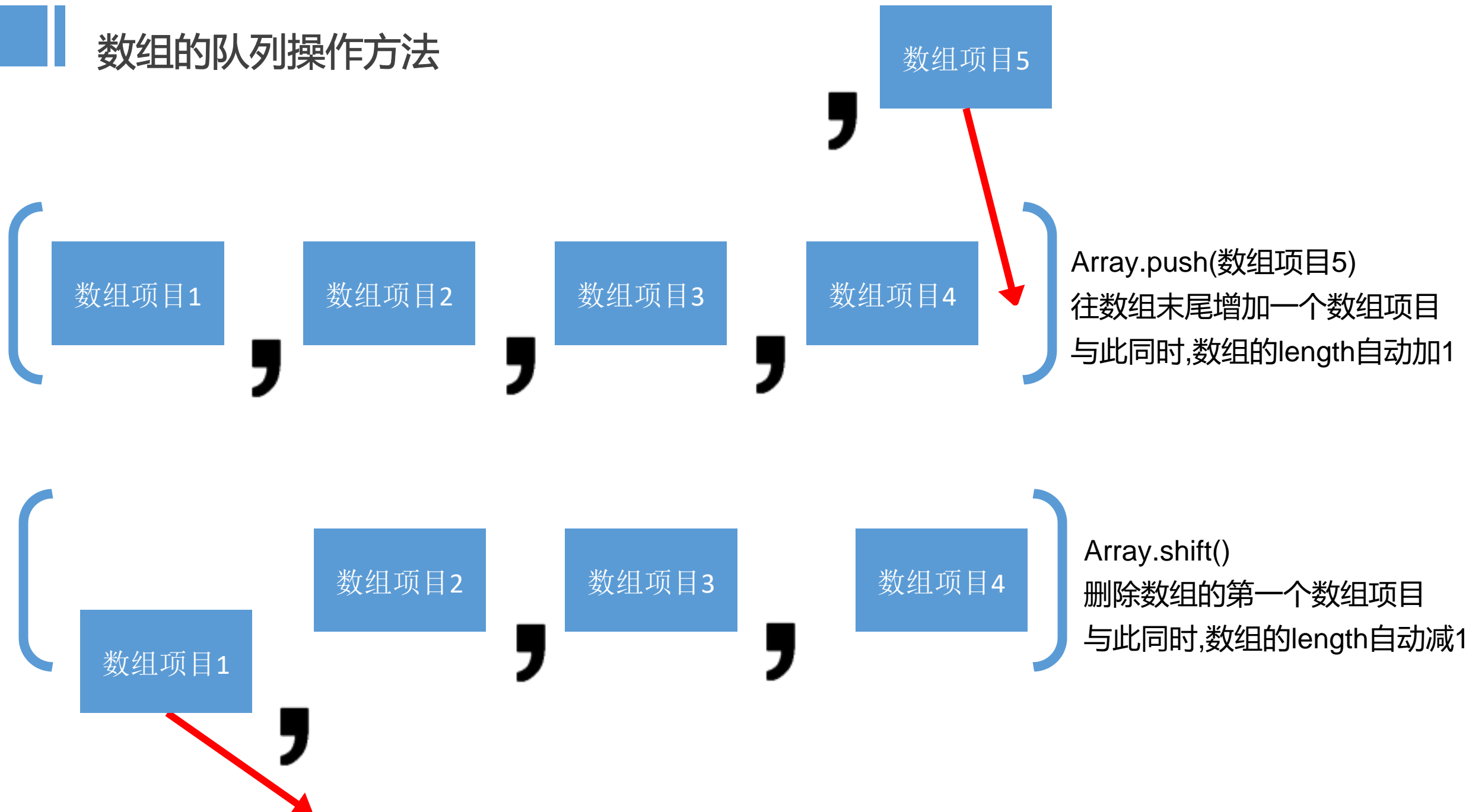
# 数组的栈和队列的操作



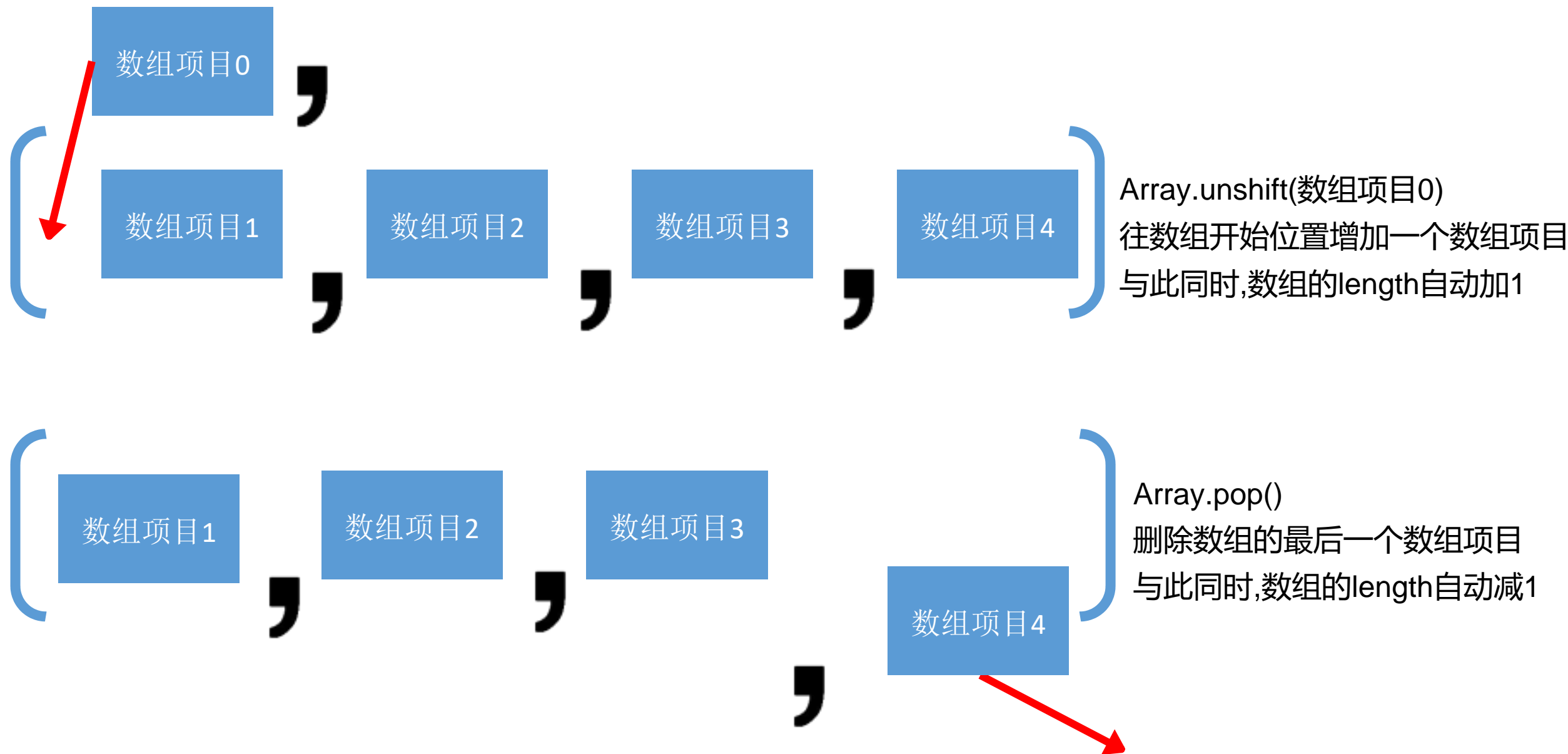
# 栈和队列的概念



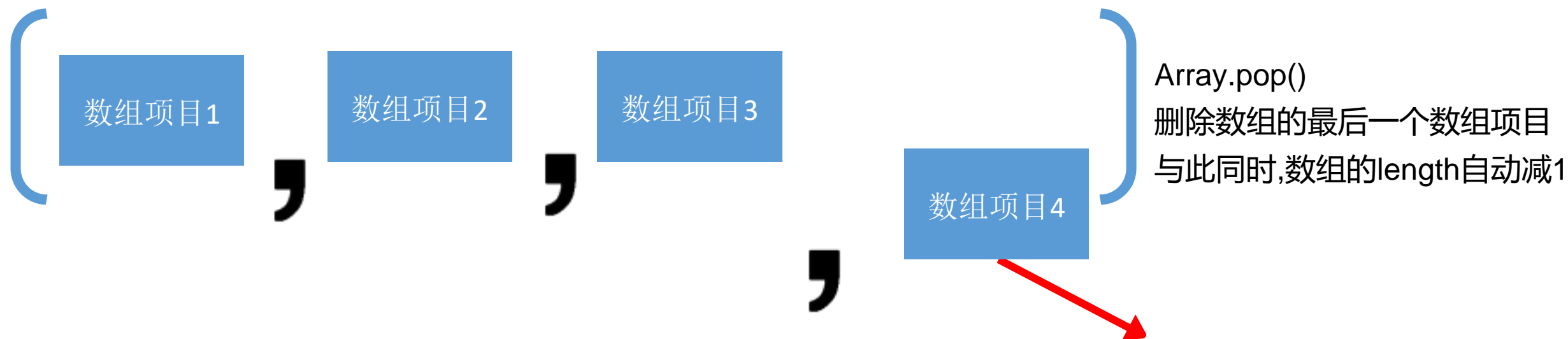
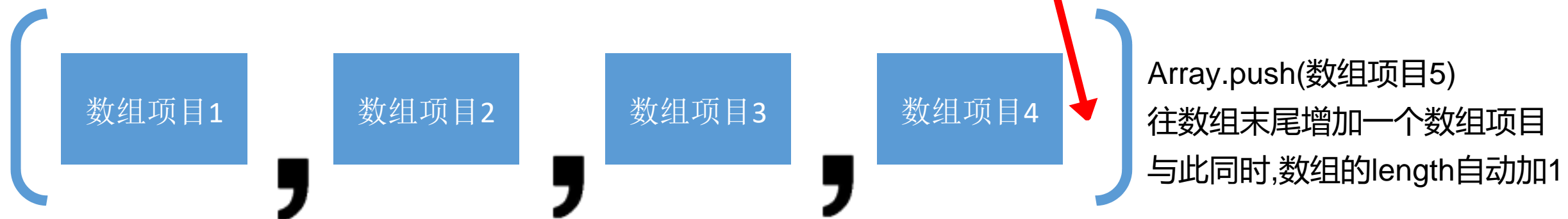
## 数组的队列操作方法



## 数组的反向队列操作方法



## 数组的栈操作方法





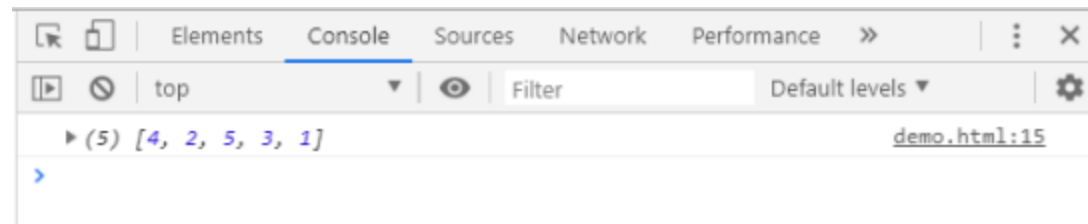
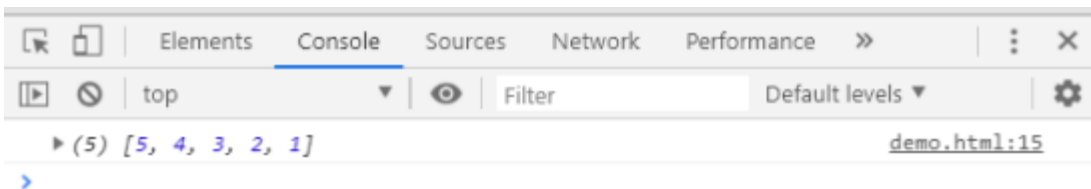
# 数组的高级方法

## 数组的高级方法之重排序

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

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

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



reverse()方法会反转数组项的顺序

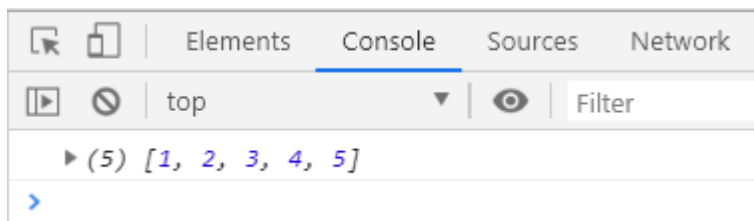
直接在原始的数组上排序, 只是单纯的调转数字的数组项目的整体顺序而已

**该方法会改变原始的数组**

## 数组的高级方法之重排序

### sort()的默认功效

```
var values = [1, 3, 5, 2, 4];  
values.sort();  
console.log(values); //5,4,3,2,1
```



```
var values = [10, 3, 50, 2, 4];  
values.sort();  
console.log(values); //5,4,3,2,1
```



sort()方法会按照字符串的先后顺序对数组的每一个数组项目的**字符顺序来进行排序的**  
如果数组的某个项目不是字符串, 那么会先把数组项目转换为字符串,再来按照字符串的比较方式来排序  
**该方法会改变原始的数组**

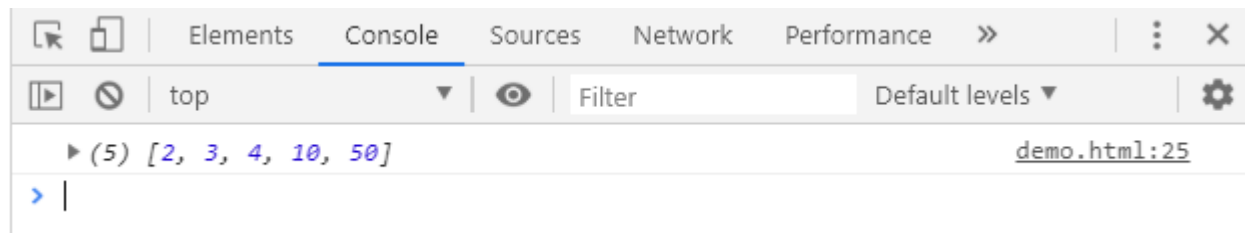
## 数组的高级方法之重排序

sort()方法可以传入一个参数, 该参数必须为一个函数(被称为比较函数),比较函数接收两个参数

```
var values = [10, 3, 50, 2, 4];

values.sort(function (value1, value2) { //value1 和 value2这两个参数表示任意两个数组项目两两对比
  if (value1 < value2) { //任意两个项目的比较算法
    return -1; //如果返回的是-1的话, 就是value1 在数组前 value2 在数组后
  } else if (value1 > value2) { //任意两个项目的比较算法
    return 1; //如果返回的是1的话, 就是value2 在数组前 value1在数组后
  } else {
    return 0; //返回0的话, value1和value2的位置随意互换
  }
});

console.log(values); //5,4,3,2,1
```





## 数组的高级方法之重排序

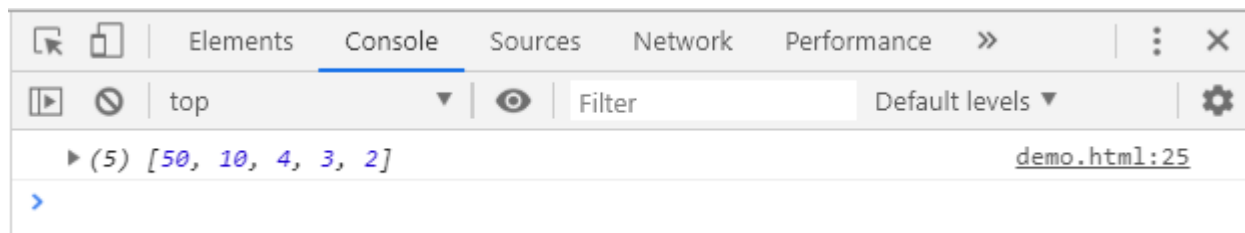
**特别注意**, 任意两个数组项目的排列顺序是由返回参数决定的

参数有三类, 负数, 正数和0, 不论比较算法是啥, 反正数组最后的排序结果就按照参数所规定的的来  
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

```
var values = [10, 3, 50, 2, 4];

values.sort(function (value1, value2) { //value1 和 value2这两个参数表示任意两个数组项目两两对比
    if (value1 > value2) { //任意两个项目的比较算法
        return -1; //如果返回的是-1的话, 就是value1 在数组前 value2 在数组后
    } else if (value1 < value2) { //任意两个项目的比较算法
        return 1; //如果返回的是1的话, 就是value2 在数组前 value1在数组后
    } else {
        return 0; //返回0的话, value1和value2的位置随意互换
    }
});

console.log(values); //5,4,3,2,1
```



## 数组的高级方法之concat()操作方法

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。

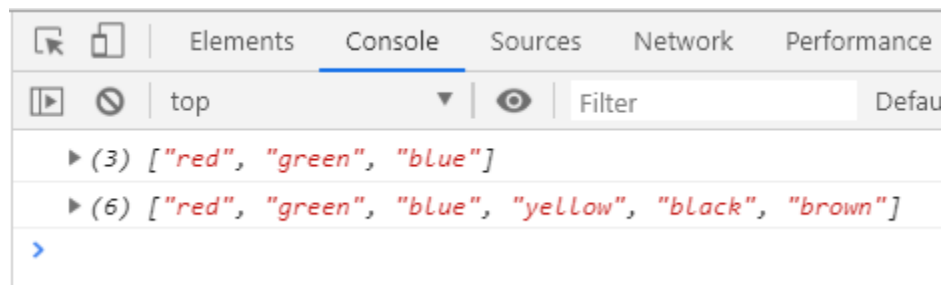
简单来说, **concat()方法不会改变原数组**

```
var colors = ["red", "green", "blue"];
var colors2=colors.concat();
console.log(colors);
console.log(colors2);
```



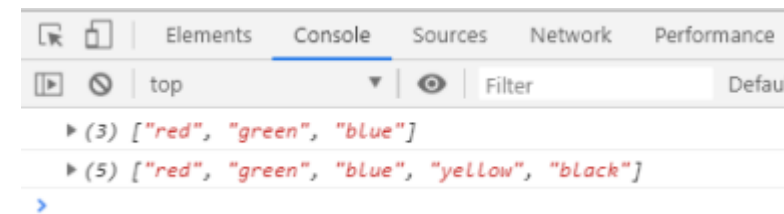
在没有给concat()方法传递参数的情况下，它只是复制当前数组并返回副本。副本和原来的数组在内存的两个位置(如果是对象的话另说 对

```
var colors = ["red", "green", "blue"];
var colors2 = colors.concat("yellow", ["black", "brown"]);
console.log(colors); //red,green,blue
console.log(colors2); //red,green,blue,yellow,black,brown
```



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。

```
var colors = ["red", "green", "blue"];
var colors2 = colors.concat("yellow", "black");
console.log(colors); //red,green,blue
console.log(colors2); //red,green,blue,yellow,black
```



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

## 数组的高级方法之slice()操作方法

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。

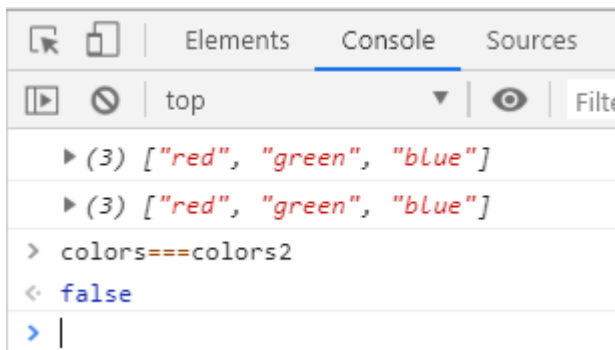
**情况1:**如果不传入参数的情况下，slice()方法返回完整的数组

**情况2:**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。

**情况3:**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。(左闭右开区间)

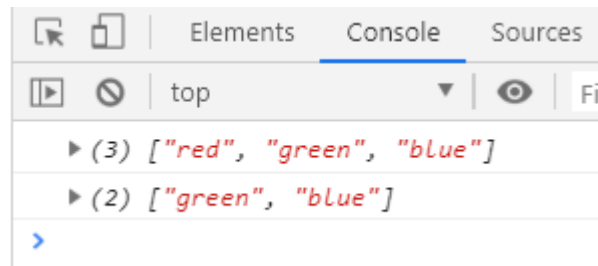
**注意，slice()方法不会影响原始数组**

```
var colors = ["red", "green", "blue"];
var colors2 = colors.slice();
console.log(colors); //red,green,blue
console.log(colors2); //red,green,blue
```



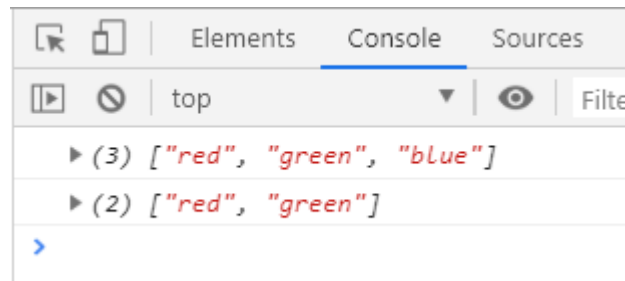
**情况1**

```
var colors = ["red", "green", "blue"];
var colors2 = colors.slice(1);
console.log(colors); //red,green,blue
console.log(colors2); //green,blue
```



**情况2**

```
var colors = ["red", "green", "blue"];
var colors2 = colors.slice(0,2);
console.log(colors); //red,green,blue
console.log(colors2); //red,green
```



**情况3**

## 数组的高级方法之splice()操作方法

splice()是最强大的数组方法, 主要功能有以下三大类

**注意, splice()方法会影响原始数组**

```
var colors = ["red", "green", "blue", "yellow", "purple"];
```

**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"。

## 数组的高级方法之splice()操作方法的删除模型

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。

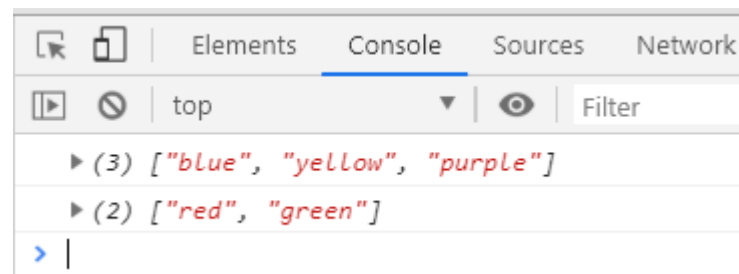
例如，`splice(0,2)`会删除数组中的前两项。

`var colors = ["red", "green", "blue", "yellow", "purple"];`

`splice(0,2)`

`var colors = ["red", "green", "blue", "yellow", "purple"];`

```
var colors = ["red", "green", "blue", "yellow", "purple"];
var colors2 = colors.splice(0,2);
console.log(colors); //"blue", "yellow", "purple"
console.log(colors2); //"red", "green"
```



注意，`splice()`方法会影响原始数组

## 数组的高级方法之splice()操作方法的插入模型

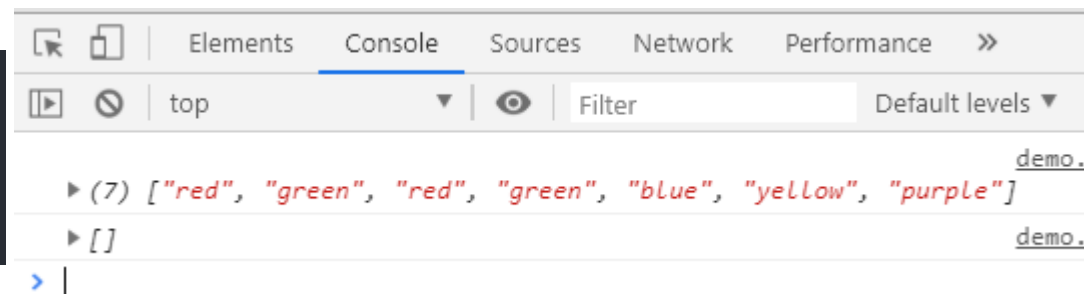
插入：可以向指定位置插入任意数量的项，只需提供3 个参数：起始位置、0（要删除的项数）和要插入的项。如果要插入多个项，可以再传入第四、第五，以至任意多个项。例如，`splice(2,0,"red","green")`会从当前数组的位置2 开始插入字符串"red"和"green"。

```
var colors = ["red", "green", "blue", "yellow", "purple"];
```

```
splice(2,0,"red","green")
```

```
var colors = ["red", "green", "red", "green", "blue", "yellow", "purple"];
```

```
var colors = ["red", "green", "blue", "yellow", "purple"];
var colors2 = colors.splice(2,0,"red","green");
console.log(colors); //[ "red", "green", "red", "green", "blue", "yellow", "purple" ]
console.log(colors2); //空数组
```



注意，`splice()`方法会影响原始数组

## 数组的高级方法之splice()操作方法的替换模型

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"。

```
var colors = ["red", "green", "blue", "yellow", "purple"];
```

```
splice(2,2,"red","green")
```

```
var colors = ["red", "green", "blue", "yellow", "purple"];
```

```
splice(2,2,"red","green")
```

```
var colors = ["red", "green", "red", "green", "purple"];
```

先删除指定项目

再删除指定项目

## 数组的高级方法之位置方法

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===）。

```
var numbers = [1, 2, 3, 4, 5, 4, 3, 2, 1];
console.log(numbers.indexOf(4)); //3
console.log(numbers.lastIndexOf(4)); //5
console.log(numbers.indexOf(4, 4)); //5
console.log(numbers.lastIndexOf(4, 4)); //3
```

普通数据类型的位置查找

```
var person = {
  name: "Nicholas"
};
var people = [{
  name: "Nicholas"
}];

var morePeople = [person];

console.log(people.indexOf(person)); //-1
console.log(morePeople.indexOf(person)); //0
```

引用类型的位置查找