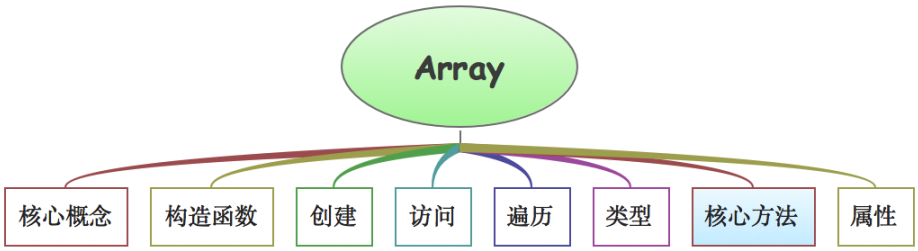
Array

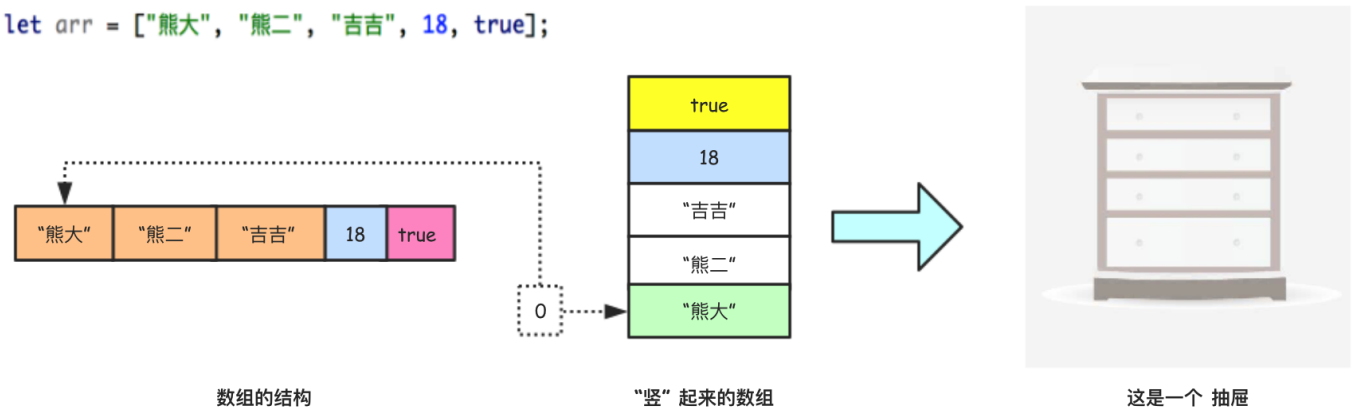
> 本文将介绍JavaScript语言中的数组，全文内容包括但不限于数组的简单介绍、数组的创建、数组中元素的访问、数组的类型以及数组相关的核心方法等内容，需要指出的是 ES6 + 的数组相关特性并没有被纳入到本文的讨论范围中，它们将在单独的篇章中被介绍。

数组核心概念



> 在JavaScript语言中，数组的构造函数为 Array ，如果我们使用 typeof 关键字来检查数组的类型会得到 object 的结果，这也从侧面说明了数组本质上是Object对象类型的数据，可以认为数组是特殊的对象(关于这一点，这篇文章将不做展开)。

```
var arr = [1,2,3,90];
console.log(typeof arr);  //object
console.log(arr);         // [1, 2, 3, 90]
```



数组的结构　　"竖" 起来的数组　　这是一个 抽屉

数组 数组是值的有序集合 譬如：[1,3,5,7,9]，数组中的每个值称为元素。
索引 每个元素在数组中都有一个位置，用数字表示称为索引，索引值默认从 0 开始依次递增。
类型 JavaScript中的数组是 无类型的（数组元素可以是任意类型且允许存在不同类型的元素）。
动态 JavaScript中的数组是 动态的 ，会根据需要增长或缩减，无需提前声明大小且不用关心空间分配。

长度 JavaScript中每个数组都拥有 length属性，通常该属性的值为数组的长度。
备注 JavaScript中的数组是对象的特殊形式 ，但通常数组的实现是经过优化的，性能更好。

数组的创建

通常，创建数组有两种方式，一种是直接通过字面量的方式创建，一种是通过Array构造函数的方式创建，在具体写代码的时候又有一些注意点。

① 字面量方式创建数组

字面量(直接量)创建数组是最简单的方式，只需要直接使用 [ ] 并在中括号中设置数组元素即可。

```
var empty  = [];                                    //没有元素的空数组
var ages   = [12,16,18,88,101];                     //有五个元素的数字数组
var books  = ["老虎老虎","你好，忧愁","兄弟","或者"]; //有四个元素的字符串数组
var music  = ["那个女孩","take me to your heart"]; //有两个元素的字符串数组

/*稍微复杂的数组：该数组拥有6个元素，包含字符串、数值、数组、对象和布尔值等类型的元素*/
var dataArr1 = ["前端开发",5,"1904",true,["张三","李四"],{"address":"广州市天河区","num":88}];
/*数组直接量中的值(元素)不一定非的是常量，可以是任意的表达式(变量)*/
var dataArr2 = [ages, books,{name:"zs"}];
var dataArr3 = [10,,30];
console.log(dataArr3.length,dataArr3[1]);   //3, undefined
var dataArr4 = [,,];
console.log(dataArr4.length,dataArr4);      //2,[undefined,undefined]
```

② 使用构造函数创建数组

调用构造函数Array是创建数组的第二种方法，在调用构造函数的时候可以有多种方式。

语法 new Array() new Array(length) new Array(ele1,ele2,..)

```
/*使用构造函数Array来创建数组实例*/
  /*01-不传递任何参数 初始化空数组*/
  var arr1 = new Array();     //空数组 等价于 var arr1 = [];
  console.log(arr1.length);   //0

  /*02-传递数组的长度参数*/
  var arr2 = new Array(5);
  console.log(arr2.length);   //5

  /*03-传递数组的元素*/
  var arr3 = new Array(3,4,5,"demoString");
  console.log(arr3);          //[3,4,5,"demoString"]
  console.log(arr3.length);   //4

  /*其它用法*/
  var arr4 = new Array;      //空数组 当Array构造函数没有参数时可以省略()
  console.log(arr4.length);

  var arr5 = Array("demo1","demo2",true);
  console.log(arr5);                  //["demo1", "demo2", true]
  console.log(Array.isArray(arr5));   //检查是否是数组 true
```

```
console.log(arr5.length);           //3
```

说明 在使用Array构造函数来创建数组的时候， 如果没有传递参数，那么()可以被省略 ，这种情况下会初始化得到一个空的数组，如果仅仅传递一个参数且该参数是数值那么将会初始化得到一个指定长度的空数组(数组中每个元素值均为 undefined)，调用构造函数Array的时候，new关键字可以省略。

数组的访问以及length属性

数组其实就是一组数据，我们对数组的可以有很多操作(添加、移除、排序、翻转、遍历等)，这些操作主要通过对应的方法和结构来实现，这里先简单介绍数组的 length属性 、 索引下标访问 。

我们可以直接通过下标(索引)来操作数组，这些操作包括添加、修改和读取。

```
/*演示通过[下标]索引操作数组*/
var arr = ["苹果","西瓜","橘子","晓夏","描夏"];

/*通过索引访问指定元素  索引范围[0 , length - 1]*/
console.log(arr[0]);              //苹果
console.log(arr[arr.length - 1]); //描夏
console.log(arr[arr.length]);     //undefined

/*通过索引修改指定元素*/
arr[0] = "百香果";
console.log(arr);                 //["百香果","西瓜","橘子","晓夏","描夏"];

/*通过索引来添加指定元素*/
arr[arr.length] = "米桃儿";
arr[arr.length] = "香蕉";
console.log(arr);                 //["百香果", "西瓜", "橘子", "晓夏", "描夏", "米桃儿", "香蕉"]
console.log(arr.length);          //7
```

在数组中 length 是一个可读可写的属性。当我们读取某个数组的length属性时，表示获取数组的长度(最大索引，特殊情况除外)。当我们对数组的length属性进行写操作的时候，会影响数组的结构。下面简单总结为两点：

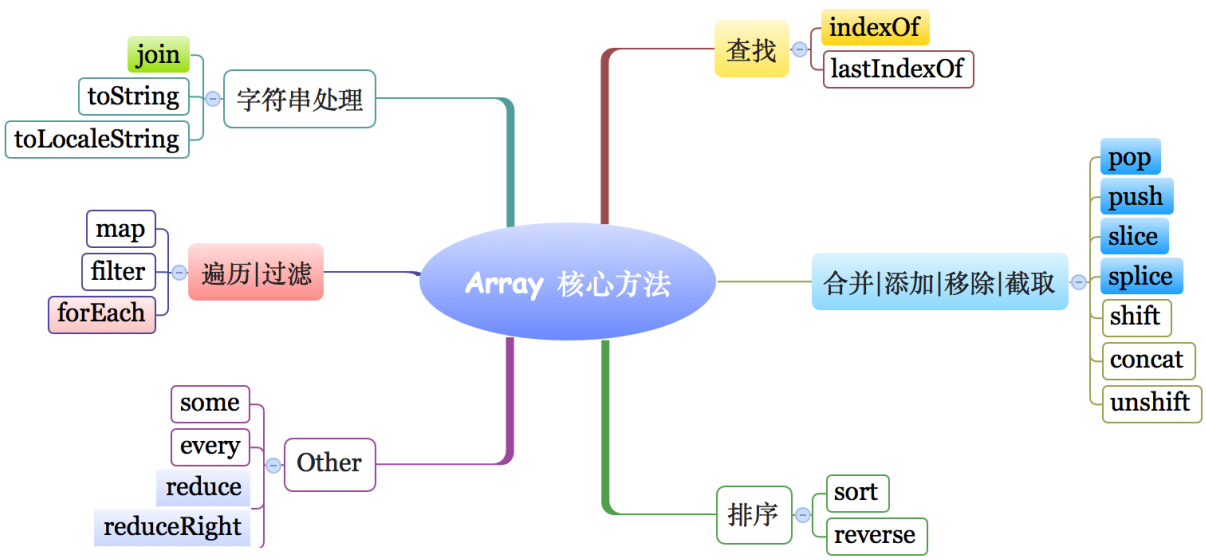
> ❏ 为数组元素赋值的时候，如果它的索引i大于或等于当前长度，那么length值将设置为i + 1；
> ❏ 设置length属性小于当前长度的非负整数时，数组中那些索引值超出的元素将被删除。

```
/*01-第一种情况 索引 >= 数组.length*/
var arr1 = [123,"string",true];
console.log(arr1.length);         //数组的长度为3
arr1[5] = "测试元素";
console.log(arr1);                // [123, "string", true, empty × 2, "测试元素"]
console.log(arr1.length);         //6 数组的长度被设置为 5 + 1 == 6

/*02-第二种情况 设置length的值 < 数组.length*/
var arr2 = ["苹果","西瓜","香蕉","橙子","橘子"];
console.log(arr2);
console.log(arr2.length);         //5
arr2.length = 3;
```

```
console.log(arr2);                  // ["苹果", "西瓜", "香蕉"]
console.log(arr2.length);           //3 数组中超出的元素被删除
```

数组常用方法



添加 | 移除 | 合并 | 截取操作

数组的合并方法 concat( )

语法　array.concat(value1,...)

作用　concat() 方法用来衔接(拼接合并)数组，会将合并后的新数组返回。

参数　concat() 方法的参数非常灵活可以是N个普通元素或者是数组。

```
var arr = [1,2,3];
arr.concat(4);              //返回[1,2,3,4];
console.log(arr);           //[1,2,3]  注意arr数组本身并没有被修改

console.log([1, 2, 3].concat(4, 5));          //[1,2,3,4,5]
console.log([1, 2, 3].concat([4, 5]));        //[1,2,3,4,5]
console.log([1, 2, 3].concat([4, 5],6));      //[1,2,3,4,5,6]
console.log([1, 2, 3].concat([4, 5],[6,7]));//[1,2,3,4,5,6]
console.log([1, 2, 3].concat(4,[5 ,[6,7]]));//[1,2,3,4,5,[6,7]];
```

数组元素的添加 push( )和unshift( )

语法　array.push(value,...)　array.unshift(value,...)

说明　push() 和 unshift() 这两个方法，它们均直接修改当前数组本身并返回最新的数组长度。

作用　unshift() 方法把元素插入到开头原本元素顺次后移，　push() 方法把元素追加到数组的屁股。

```
/*数组的添加(插入操作)*/
/*01-push() 追加到数组末尾*/
var arr1 = ["苹果", "橙子"];
```

```
console.log(arr1.push(123));          //3    把123追到到数组末尾并更新length值
console.log(arr1);                    //["苹果", "橙子", 123]
console.log(arr1.length)              //3
/*多个参数值的情况*/
console.log(arr1.push("猴子","斑马"));  //5    追加数据 + 更新length值
console.log(arr1);                    //["苹果", "橙子", 123, "猴子", "斑马"]

/*02-unshift() 插入到数组开头*/
var arr2 = ["苹果","橙子"];

console.log(arr2[0]);
arr2.unshift("老虎");
console.log(arr2[0]);
console.log(arr2);
console.log(arr2.length);
/*多个参数值的情况*/
console.log(arr2.unshift(100, "犀牛"));  //5
console.log(arr2);                      //[100, "犀牛", "老虎", "苹果", "橙子"]
```

数组元素的移除 pop()和shift()

语法 array.pop() array.shift()

作用 array.pop() 和 array.shift() 方法方法删除数组的最后一个(第一个)元素并返回删除项。

说明 如果数组为空，那么它们不会修改数组本身而是直接返回undefined。

```
var arr = ["苹果","橙子"];
console.log(arr.pop());     //"橙子" 删除最后一个元素
console.log(arr);           //["苹果"]
console.log(arr.pop());     //"苹果"
console.log(arr);           //[]
console.log(arr.pop());     //undefined
/*操作arr数组*/
arr.push("百香果","哈密瓜");
console.log(arr);           //["百香果","哈密瓜"]
console.log(arr.shift());   //"百香果"
console.log(arr);           //["哈密瓜"]
```

数组的截取方法 slice()

语法 array.slice(start,end)

作用 slice() 方法截取数组中指定的元素并保存到新数组中返回。

说明 start 和 end 参数均可以接收负数值(表示倒数)， end 缺省表示默认截取到数组末尾。

```
var arr1 = [18,"文顶顶","广州市","0415",true];
var arr2 = [18,"wendingding","广州市","0415",true];

console.log(arr1.slice(2, 4));    // ["广州市", "0415"]
console.log(arr1);                // [18,"文顶顶","广州市","0415",true];
console.log(arr1.slice(2));       // ["广州市","0415",true]

  /*参数是负数的情况*/
console.log(arr2.slice(2, -1));   //["广州市", "0415"]
console.log(arr2.slice(2, -2));   //["广州市"]
```

数组的插入、删除和替换方法 splice()

语法 `array.splice( start,deleteCount,value,...)`

作用 `splice()` 将删除从start索引开始的零个或多个元素并使用参数列表中的值来替换它们。

说明 在有必要时，数组中所有的元素都会移动以保持连续性，该方法修改的是数组本身。

```
 var arr = [18,"文顶顶","广州市","0415",true];
/* 删除数组中的数据
 * 参数1：从索引为1的位置开始
 * 参数3：删除3个元素
 * 返回值：把删除的元素保存到新数组中返回
 * 原数组：移除指定的元素(其它的元素会移动等)
 * */
 console.log(arr.splice(1, 3));    //["文顶顶","广州市","0415"]
 console.log(arr);                 //[18,true]

 /*插入(新增)数组中的数据*/
 console.log(arr.splice(1, 0, "demoA", "demoB", "北京市")); //[]
 console.log(arr);         //[18, "demoA", "demoB", "北京市", true]
```

> **备注** `splice()` 方法在使用的时候，如果传递的第二个参数值为0，那么表示插入操作。具体执行的时候，会把参数列表中的数据依次插入到指定索引位置，数组中已有的元素则顺序后移。

数组转换为字符串的操作

JavaScript中数组转换为字符串的相关方法主要有 `toString()` 、 `toLocaleString()` 以及 `join()` ，前两个方法用来把数组转换为字符串形式输出， `join()` 方法把数组的元素按既定格式拼接后输出。

语法 `array.toString()` `array.toLocaleString()` `array.join([separator])`

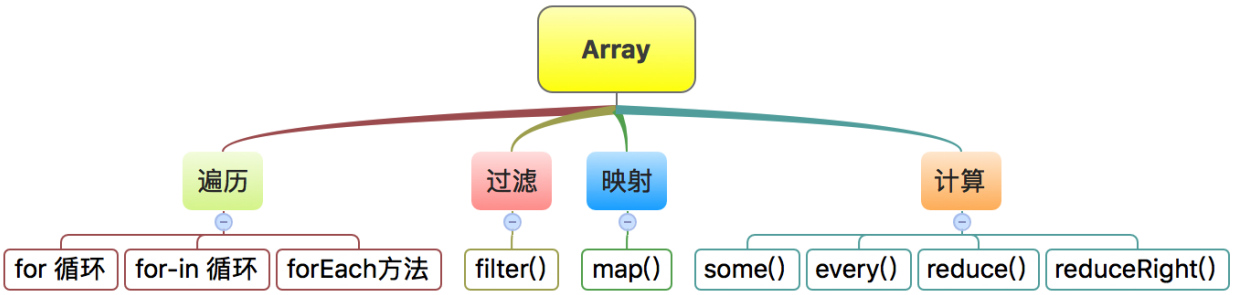
说明 `toString()` 方法在执行的时候内部会分别拿到每个元素来调用toString方法，然后把结果以逗号的形式拼接成字符串返回， `toLocaleString()` 方法和 `toString()` 用法一样，而且大多数情况下得到的结果是一致的，区别在于它每个元素调用的是toLocaleString方法。 `join()` 方法会先将数组的每个元素都转换为字符串，并通过在中间插入的 separator 字符串将它们衔接起来最后返回。

注意 如果 `join()` 方法的 `separator` 参数缺省，那么默认为逗号和toString方法等价。

```
/*数组中的每个元素本身就是字符串 这种情况直接衔接*/
var arr = ["广东省","广州市","天河区","体育中心"];
console.log(arr.toString());         //"广东省","广州市","天河区","体育中心"
console.log(arr.toLocaleString());   //"广东省","广州市","天河区","体育中心"

/*参数缺省的情况，分隔符默认为逗号*/
console.log(arr.join());             //"广东省","广州市","天河区","体育中心"
console.log(arr.join("-"));          //"广东省"-"广州市"-"天河区"-"体育中心"
```

数组的遍历和计算等操作



数组的遍历

JavaScript中数组的遍历有多种方式，我们可以用 `for循环结构` 来遍历数组，也可以使用专门遍历对象的 `for..in结构` ，当然还有ES5推出的 `forEach方法` ，下面将通过代码的方式一一介绍。

```
/*01-for循环结构遍历数组*/
var arr = [18,"wendingding","广州","0408"];
for (var i = 0,len = arr.length;i<len;i++)
{
  console.log(i, arr[i]);
}

/*02-for..in循环结构遍历数组*/
for (index in arr)
{
  /*index 当前的key即索引值 对应的是0, 1, 2, 3...*/
  /*arr   遍历的数组对象*/
  console.log(index, arr[index]);
}

/*03-forEach方法遍历数组*/
arr.forEach(function (value,index,arrT) {
  /*index 索引值*/
  /*value 当前索引值对应的数组元素*/
  /*arrT  其实就是遍历的数组本身，在这里其实就是arr的引用*/
  console.log(index, value);
})

//遍历的结果
// 0 18
// 1 "wendingding"
// 2 "广州"
// 3 "0408"
```

说明 上面介绍了遍历数组的三种方法，在开发中具体使用的时候具体选择使用哪种方式需要看特定的业务场景，需要注意的是在遍历数组的时候不建议使用 for..in 循环，这是因为 for..in 在使用的时候会枚举从原型中继承来的成员。 `forEach()` 方法会按照索引从小到大来遍历数组，并对数组中的每一个元素调用一次回调函数。每次在调用回调函数的时候，带有三个参数 `可以简单表示为f(arr[i],i,arr)` ,回调函数的返回值都会被忽略。此外， `forEach()` 方法的语法有两种形式，这里列出。

语法 `array.forEach(f)` `array.forEach(f,o)`

> 如果在调用forEach()方法的时候指定了第二个参数 o ，那么调用的时候函数的this将被绑定给 o ,若未指定第二个参数，那么回调函数中的this默认指向全局对象，在严格模式下this指向的是null。

数组的过滤和映射

语法 array.filter(predicate) array.filter(predicate,o)

数组的 filter() 方法用来对数组进行过滤，该方法返回一个只包含通过 断言 元素的新数组。
filter() 方法在执行的时候会按照索引从小到大的顺序来遍历数组，并对数组的每个元素都调用一次 predicate 断言函数。当每次调用该函数的时候， predicate 断言函数都可以带有三个参数 可以简单表示为 predicate(arr[i],i,arr) ，若函数返回真值，那么当前元素就会被添加到新创建的数组中。

```
var arr = [18,5,"广州","0408",101,9,21];
var res = arr.filter(function (value,index,arrT) {
  /*index 当前的索引*/
  /*value 当前索引对应的元素*/
  /*arrT  遍历的数组其实就是arr*/
  return typeof value == "number" && value >= 18;
})
console.log(arr); //[18,5,"广州","0408",101,9,21];
console.log(res); //[18, 101, 21]
```

语法 array.map(f) array.map(f,o)

数组的 map() 方法会根据f函数来计算每个元素并把它们返回组成一个新的数组，通常我们把这个方法称为数组映射方法。

```
var arr = [18,5,"广州","0408",101,9,21];
var res = arr.map(function (value, index, arrT) {
  console.log(index,value);

  /*检查当前元素是否是数字，如果是那么就放大一倍*/
  if (typeof value == "number")
  {
    return value * 2;
  }
  /*如果没有显示的return，那么默认的返回值为undefined*/
})
console.log(res);

// 0 18
// 1 5
// 2 "广州"
// 3 "0408"
// 4 101
// 5 9
// 6 21
// [36, 10, undefined, undefined, 202, 18, 42]
```

数组计算的相关方法

这里将一起介绍数组中计算相关的四个方法，其中 every() 方法和 some() 方法用于对数组元素进行断言测试，而 reduce() 方法和 reduceRight() 方法则用于计算数组元素的值，它们常用于数值数组。

every() 方法测试断言函数是否对每个元素均为真， some() 方法测试是否有元素满足断言函数，这两个函数返回的结果为布尔类型值的 true 或 false，every() 函数要求如果所有的元素都满足断言函数的测试条件，那么就返回true，而 some() 方法只要有一个元素满足测试条件，那就返回 true 。

```
var isAllEvenNumber1 = [2,4,6,8,10].every(function (value,index,arrT) {
  /*index 当前的索引*/
  /*value 当前的元素*/
  /*arrT  操作的数组对象 [2,4,6,8,10]*/
  console.log(index, value, arrT);
  return value % 2 == 0;
})

var isAllEvenNumber2 = [1,4,6,8,10].every(function (value,index,arrT) {
  return value % 2 == 0;
})

var isHasEvenNumber = [1,3,5,7,10].some(function (value,index,arrT) {
  return value % 2 == 0;
})

console.log(isAllEvenNumber1);   //true
console.log(isAllEvenNumber2);   //false
console.log(isHasEvenNumber);    //true
```

语法 array.reduce(f,[inital]) array.reduceRight(f,[inital])

参数 参数 f 是一个函数，用于把两个值合并缩减为一个新值， 可选的 inital 是缩减的初始值。

返回 返回的是数组的化简值，也就是最后一次调用f函数时的返回值。

对比 reduceRight() 方法在于它在进行缩减计算的时候按照索引从大到小的顺序执行。

```
/*01-reduce方法演示*/
var arr = [1,2,3,4];
var res = arr.reduce(function (x,y,i,arrT) {
  /*x    和 y 分别是当前计算的两个元素 按索引从小到大*/
  /*i    对应的是y这个元素的索引*/
  /*arrT 对应的是计算的数组，在这里其实就是arr*/
  return x *y;
})

console.log(res); //24 计算方式(((1 * 2) *3) *4)
/*
* 第一次计算的时候x = 1,y = 2 x * y 得到的结果2 该值又作为下一次计算的x值
* 第二次计算的时候x = 2,y = 3 x * y 得到的结果6 该值又作为下一次计算的x值
* 第三次计算的时候x = 6,y = 4 x * y 得到的结果24
* 因为已经是最后一次调用，把24这个结果返回
* */

console.log([2, 4, 8].reduce(function (x, y) {
  return x + y;
}, 20));
// (((20 + 2) + 4) + 8) = 34
```

```
// (((20 + 2) + 4) + 8) = 34

/*案例：计算数组元素的累加和*/
console.log([1, 2, 3, 4, 5, 6, 7, 8, 10].reduce(function (x,y) {
  return x + y;
}));

// 累加和为46

 /*02-reduceRight方法演示*/
 console.log([1, 2, 3, 4].reduceRight(function (x, y) {
    return x * y;
 }, 2));

//计算方式:((((2 * 4) * 3) *2) *1) == 48
```

数组的排序和翻转

JavaScript中的 `reverse()` 方法用于将数组的顺序颠倒过来，该方法操作的是原数组。

```
var arr1 = [1,2,3,4];
arr1.reverse();
console.log(arr1);      //[4, 3, 2, 1]

var arr2 = ["demoA","demoB",true,18];
arr2.reverse();
console.log(arr2);      //[18, true, "demoB", "demoA"]
```

如果需要对数组进行排序，那么可以使用 `sort()` 方法。 `sort()` 方法会在原数组中对数组元素进行排序而不创建新数组， `sort()` 方法默认按照字符编码的顺序对元素进行排序，当然也可以提供比较函数来指定排序的方式。

```
var arr1 = [33,4,1111,22,100];
var arr2 = ["a","c","f","e","d","b"];
/*默认按照字符编码顺序排序*/
console.log(arr1.sort());   //[100, 1111, 22, 33, 4]
console.log(arr2.sort());   //["a", "b", "c", "d", "e", "f"]

console.log([12,20,5,32,101].sort(function (a,b) {
  return a - b;
}));
//排序1 [5, 12, 20, 32, 101]

console.log([12,20,5,32,101].sort(function (a,b) {
  return b - a;
}));
//排序2 [101, 32, 20, 12, 5]
```

排序函数 排序函数接收两个参数，这里是 a 和 b 它们依据函数的返回值来确定是升序还是降序排列。

数组元素的搜索

搜索说明 同string字符串类型一样，数组也提供了 `indexOf()` 和 `lastIndexOf()` 方法来实现元素的查找功能，它们的区别在于一个从前往后查找一个从后往前查找，如果在数组中找到指定的元素那么就返回对应的索引，如果没有找到那么就返回 `-1` 。

```
var arr = [18,"文顶顶","北京市","Node",18];
/*indexOf     从左往右搜索*/
console.log(arr.indexOf("文顶顶"));         //1
console.log(arr.indexOf(18));              //0
/*lastIndexOf 从右往左搜索*/
console.log(arr.lastIndexOf("北京市"));     //2
console.log(arr.lastIndexOf(18));          //4

console.log(arr.indexOf("天津"));          //-1
console.log(arr.lastIndexOf("天津"));      //-1
```

数组的类型

在ES5之前，要通过代码判断一个数据是否是数组可能会比较麻烦。我们知道typeof关键字可用于检查数据的类型，譬如得到的是结果如果是 string那我们就能知道该数据是字符串，如果结果是 number那这个数据自然就是数字。

我们并不能通过 `typeof 数组实例` 的方式来判断数组，因为你得到的结果将会是 `object` ，值得庆幸的是在ES5中，Array提供了isArray()函数来检查数组实例。

```
/*01- typeof*/
var arr = [1,2,3,9];
console.log(typeof arr);                            //object   无法检查数组的类型

/*02- 调用Object.prototype.toString()方法来检测*/
console.log(Object.prototype.toString.call(arr)); //"[object Array]"


/*03- isArray()方法*/
console.log(Array.isArray(arr));                    //true
```