数组

- 什么是数组?
- 数组，是有序的元素序列。
- 通俗讲，数组就是有序的数据集合。
- 数组属于对象类型。
- 数组的作用：用于在单个变量中存储多个值。

创建一个数组

- 数组就是一个 `[]`
- 在 `[]` 里面存储着各种各样的数据，按照顺序依次排好

字面量创建一个数组

- 直接使用 `[]` 的方式创建一个数组

```
// 创建一个空数组
var arr1 = []

// 创建一个有内容的数组
var arr2 = [1, 2, 3]
```

内置构造函数创建数组

- 使用 js 的内置构造函数 `Array` 创建一个数组

```
// 创建一个空数组
var arr1 = new Array()

// 创建一个长度为 10 的数组
var arr2 = new Array(10)

// 创建一个有内容的数组
var arr3 = new Array(1, 2, 3)
```

数组的 length

- length: 长度的意思

- length 就是表示数组的长度，数组里面有多少个成员，length 就是多少

```
// 创建一个数组
var arr = [1, 2, 3]

console.log(arr.length) // 3
```

数组的索引

- 索引，也叫做下标，是指一个数据在数组里面排在第几个的位置
- 注意： **在所有的语言里面，索引都是从 0 开始的**
- 在 js 里面也一样，数组的索引从 0 开始

```
// 创建一个数组
var arr = ['hello', 'world']
```

- 上面这个数组中，**第 0 个** 数据就是字符串 `hello`，**第 1 个** 数据就是字符串 `world`
- 想获取数组中的第几个就使用 `数组[索引]` 来获取

```
var arr = ['hello', 'world']

console.log(arr[0]) // hello
console.log(arr[1]) // world
```

数组的常用方法

- 数组是一个复杂数据类型，我们在操作它的时候就不能再想基本数据类型一样操作了
- 比如我们想改变一个数组

```
// 创建一个数组
var arr = [1, 2, 3]

// 我们想把数组变成只有 1 和 2
arr = [1, 2]
```

  - 这样肯定是不合理，因为这样不是在改变之前的数组
  - 相当于心弄了一个数组给到 arr 这个变量了
  - 相当于把 arr 里面存储的地址给换了，也就是把存储空间换掉了，而不是在之前的空间里面修改
  - 所以我们就需要借助一些方法，在不改变存储空间的情况下，把存储空间里面的数据改变了

数组常用方法之 push

- `push` 是用来在数组的末尾追加一个元素

```
var arr = [1, 2, 3]

// 使用 push 方法追加一个元素在末尾
arr.push(4)

console.log(arr) // [1, 2, 3, 4]
```

数组常用方法之 pop

- `pop` 是用来删除数组末尾的一个元素

```
var arr = [1, 2, 3]

// 使用 pop 方法删除末尾的一个元素
arr.pop()

console.log(arr) // [1, 2]
```

数组常用方法之 unshift

- `unshift` 是在数组的最前面添加一个元素

```
var arr = [1, 2, 3]

// 使用 unshift 方法想数组的最前面添加一个元素
arr.unshift(4)

console.log(arr) // [4, 1, 2, 3]
```

数组常用方法之 shift

- `shift` 是删除数组最前面的一个元素

```
var arr = [1, 2, 3]

// 使用 shift 方法删除数组最前面的一个元素
arr.shift()


console.log(arr) // [2, 3]
```

数组常用方法之 splice

- `splice` 是截取数组中的某些内容，按照数组的索引来截取
- 语法：`splice(从哪一个索引位置开始，截取多少个，替换的新元素)`（第三个参数可以不写）

```
var arr = [1, 2, 3, 4, 5]

// 使用 splice 方法截取数组
arr.splice(1, 2)

console.log(arr) // [1, 4, 5]
```

  - `arr.splice(1, 2)` 表示从索引 1 开始截取 2 个内容
  - 第三个参数没有写，就是没有新内容替换掉截取位置

```
var arr = [1, 2, 3, 4, 5]

// 使用 splice 方法截取数组
arr.splice(1, 2, '我是新内容')

console.log(arr) // [1, '我是新内容', 4, 5]
```

  - `arr.splice(1, 2, '我是新内容')` 表示从索引 1 开始截取 2 个内容
  - 然后用第三个参数把截取完空出来的位置填充

数组常用方法之 reverse

- `reverse` 是用来反转数组使用的

```
var arr = [1, 2, 3]

// 使用 reverse 方法来反转数组
arr.reverse()

console.log(arr) // [3, 2, 1]
```

数组常用方法之 sort

- `sort` 是用来给数组排序的

```
var arr = [2, 3, 1,11,7,22]

// 使用 sort 方法给数组排序
//用法一
arr.sort()

console.log(arr) // [1, 2, 3]
```

注：以上方法都会改变原数组！

数组常用方法之 concat

- `concat` 是把多个数组进行拼接
- 和之前的方法有一些不一样的地方，就是 `concat` 不会改变原始数组，而是返回一个新的数组

```
var arr = [1, 2, 3]

// 使用 concat 方法拼接数组
var newArr = arr.concat([4, 5, 6])

console.log(arr) // [1, 2, 3]
console.log(newArr) // [1, 2, 3, 4, 5, 6]
```

  - 注意： **concat 方法不会改变原始数组**

数组常用方法之 join

- `join` 是把数组里面的每一项内容链接起来，变成一个字符串
- 可以自己定义每一项之间链接的内容 `join(要以什么内容链接)`
- 不会改变原始数组，而是把链接好的字符串返回

```
var arr = [1, 2, 3]

// 使用 join 链接数组
var str = arr.join('-')

console.log(arr) // [1, 2, 3]
console.log(str) // 1-2-3
```

  - 注意： **join 方法不会改变原始数组，而是返回链接好的字符串**

数组常用方法之 indexOf

- `indexOf` 用来找到数组中某一项的索引
- 语法： `indexOf(你要找的数组中的项)`

```
var arr = [1, 2, 3, 4, 5]

// 使用 indexOf 超找数组中的某一项
var index = arr.indexOf(3)

console.log(index) // 2
```

  - 我们要找的是数组中值为 3 的那一项
  - 返回的就是值为 3 的那一项在该数组中的索引

- 如果你要找的内容在数组中没有，那么就会返回 -1

```
var arr = [1, 2, 3, 4, 5]

// 使用 indexOf 超找数组中的某一项
var index = arr.indexOf(10)

console.log(index) // -1
```

  - 你要找的值在数组中不存在，那么就会返回 -1

slice()

- slice() 方法可从已有的数组中返回选定的元素。

  slice()方法可提取数组的某个部分，并以新的数组返回被提取的部分。

  语法：*array*.slice(*start*, *end*)

  注意： slice() 方法不会改变原始数组。

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var citrus = fruits.slice(1,3);
citrus 的结果是：Orange,Lemon
```

  如果某个参数为负，则从数组的结尾开始计数。

```
var fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
var myBest = fruits.slice(-3,-1);
myBest 的结果是：Lemon,Apple
```

注：以上方法不会改变原数组！

for 和 for in 循环

- 因为数组的索引就可以获取数组中的内容
- 数组的索引又是按照 0 ~ n 顺序排列
- 我们就可以使用 for 循环来循环数组，因为 for 循环我们也可以设置成 0 ~ n 顺序增加
- 我们把这个行为叫做 **遍历**

```
var arr = [1, 2, 3, 4, 5]

// 使用 for 循环遍历数组
for (var i = 0; i < arr.length; i++) {
  console.log(arr[i])
}

// 会在控制台依次打印出 1,  2,  3,  4,  5
```

  - `i < arr.length` 因为 length 就是数组的长度，就是一个数字，所以我们可以直接用它来决定循环次数
  - `console.log(arr[i])` 因为随着循环，i 的值会从 0 开始依次增加
  - 所以我们实际上就相当于在打印 `arr[0]` / `arr[1]` / ...

- 因为 **对象** 是没有索引的，所以我们没有办法使用 for 循环来遍历
- 这里我们使用 for in 循环来遍历对象
- 先来看一段代码

```
var obj = {
  name: 'Jack',
  age: 18
}

for (var key in obj) {
  console.log(key)
}

// 会在控制台打印两次内容，分别是 name 和 age
```

  - for in 循环的遍历是按照对象中有多少成员来决定了
  - 有多少成员，就会执行多少次
  - `key` 是我们自己定义的一个变量，就和 for 循环的时候我们定义的 i 一个道理
  - 在每次循环的过程中，key 就代表着对象中某一个成员的 **属性名**

for...of

- **for...of语句**在可迭代对象（包括 `Array`，`Map`，`Set`，`String`，arguments 对象等等）上创建一个迭代循环，调用自定义迭代钩子，并为每个不同属性的值执行语句。

语法：

```
for (variable of iterable) {
    statement
}
variable在每次迭代中，将不同属性的值分配给变量。iterable可枚举其枚举属性的对象。
```

- 示例

```
迭代Array
var iterable = [10, 20, 30];

for (var value of iterable) {
  value += 1;
  console.log(value);
}
// 11
// 21
// 31

迭代String
var iterable = 'boo';

for (var value of iterable) {
  console.log(value);
}
// "b"
// "o"
// "o"
```

forEach

- 和 for 循环一个作用，就是用来遍历数组的
- 语法：`arr.forEach(function (item, index, arr) {})`

```
var arr = [1, 2, 3]

// 使用 forEach 遍历数组
arr.forEach(function (item, index, arr) {
  // item 就是数组中的每一项
  // index 就是数组的索引
  // arr 就是原始数组
  console.log('数组的第 ' + index + ' 项的值是 ' + item + ', 原始数组是', arr)
})
```

  - forEach() 的时候传递的那个函数，会根据数组的长度执行
  - 数组的长度是多少，这个函数就会执行多少回

map

- 和 forEach 类似，只不过可以对数组中的每一项进行操作，返回一个新的数组

```
var arr = [1, 2, 3]

// 使用 map 遍历数组
var newArr = arr.map(function (item, index, arr) {
  // item 就是数组中的每一项
  // index 就是数组的索引
  // arr 就是原始数组
  return item + 10
})

console.log(newArr) // [11, 12, 13]
```

filter

- 和 map 的使用方式类似，按照我们的条件来筛选数组
- 把原始数组中满足条件的筛选出来，组成一个新的数组返回

```
var arr = [1, 2, 3]

// 使用 filter 过滤数组
var newArr = arr.filter(function (item, index, arr) {
  // item 就是数组中的每一项
  // index 就是数组的索引
  // arr 就是原始数组
  return item > 1
})

console.log(newArr) // [2, 3]
```

  - 我们设置的条件就是 `> 1`
  - 返回的新数组就会是原始数组中所有 `> 1` 的项
  - some满足条件返回true，没有则返回false

数组的排序

- 排序，就是把一个乱序的数组，通过我们的处理，让他变成一个有序的数组

函数排序

```
sort( [fn] ) 排序，返回数组

arr.sort(); //默认按照字符编码排序，先比较第一位

arr.sort(function (a,b) { //升序，只能对数值排序
    return a-b;
});

arr.sort(function (a,b) { //降序，只能对数值排序
    return b-a;
});
```

冒泡排序

- 先遍历数组，让挨着的两个进行比较，如果前一个比后一个大，那么就把两个换个位置
- 数组遍历一遍以后，那么最后一个数字就是最大的那个了
- 然后进行第二遍的遍历，还是按照之前的规则，第二大的数字就会跑到倒数第二的位置
- 以此类推，最后就会按照顺序把数组排好了

1. 我们先来准备一个乱序的数组

```
var arr = [3, 1, 5, 6, 4, 9, 7, 2, 8]
```

   - 接下来我们就会用代码让数组排序

2. 先不着急循环，先来看数组里面内容换个位置

```
// 假定我现在要让数组中的第 0 项和第 1 项换个位置
// 需要借助第三个变量
var tmp = arr[0]
arr[0] = arr[1]
arr[1] = tmp
```

3. 第一次遍历数组，把最大的放到最后面去

```
for (var i = 0; i < arr.length; i++) {
  // 判断，如果数组中的当前一个比后一个大，那么两个交换一下位置
  if (arr[i] > arr[i + 1]) {
    var tmp = arr[i]
    arr[i] = arr[i + 1]
    arr[i + 1] = tmp
  }
}

// 遍历完毕以后，数组就会变成 [3, 1, 5, 6, 4, 7, 2, 8, 9]
```

   - 第一次结束以后，数组中的最后一个，就会是最大的那个数字

- 然后我们把上面的这段代码执行多次。数组有多少项就执行多少次

4. 按照数组的长度来遍历多少次

```
for (var j = 0; j < arr.length; j++) {
  for (var i = 0; i < arr.length; i++) {
    // 判断，如果数组中的当前一个比后一个大，那么两个交换一下位置
    if (arr[i] > arr[i + 1]) {
      var tmp = arr[i]
      arr[i] = arr[i + 1]
      arr[i + 1] = tmp
    }
  }
}

// 结束以后，数组就排序好了
```

5. 给一些优化

- 想象一个问题，假设数组长度是 9，第八次排完以后
- 后面八个数字已经按照顺序排列好了，剩下的那个最小的一定是在最前面
- 那么第九次就已经没有意义了，因为最小的已经在最前面了，不会再有任何换位置出现了
- 那么我们第九次遍历就不需要了，所以我们可以减少一次

```
for (var j = 0; j < arr.length - 1; j++) {
  for (var i = 0; i < arr.length; i++) {
    // 判断，如果数组中的当前一个比后一个大，那么两个交换一下位置
    if (arr[i] > arr[i + 1]) {
      var tmp = arr[i]
      arr[i] = arr[i + 1]
      arr[i + 1] = tmp
    }
  }
}
```

- 第二个问题，第一次的时候，已经把最大的数字放在最后面了
- 那么第二次的时候，其实倒数第二个和最后一个就不用比了
- 因为我们就是要把倒数第二大的放在倒数第二的位置，即使比较了，也不会换位置
- 第三次就要倒数第三个数字就不用再和后两个比较了
- 以此类推，那么其实每次遍历的时候，就遍历当前次数 - 1 次

```
for (var j = 0; j < arr.length - 1; j++) {
  for (var i = 0; i < arr.length - 1 - j; i++) {
    // 判断，如果数组中的当前一个比后一个大，那么两个交换一下位置
    if (arr[i] > arr[i + 1]) {
      var tmp = arr[i]
      arr[i] = arr[i + 1]
      arr[i + 1] = tmp
    }
  }
}
```

6. 至此，一个冒泡排序就完成了

选择排序

- 先假定数组中的第 0 个就是最小的数字的索引
- 然后遍历数组，只要有一个数字比我小，那么就替换之前记录的索引
- 知道数组遍历结束后，就能找到最小的那个索引，然后让最小的索引换到第 0 个的位置
- 再来第二趟遍历，假定第 1 个是最小的数字的索引
- 在遍历一次数组，找到比我小的那个数字的索引
- 遍历结束后换个位置
- 依次类推，也可以把数组排序好

1. 准备一个数组

```
var arr = [3, 1, 5, 6, 4, 9, 7, 2, 8]
```

2. 假定数组中的第 0 个是最小数字的索引

```
var minIndex = 0
```

3. 遍历数组，判断，只要数字比我小，那么就替换掉原先记录的索引

```
var minIndex = 0
for (var i = 0; i < arr.length; i++) {
  if (arr[i] < arr[minIndex]) {
    minIndex = i
  }
}

// 遍历结束后找到最小的索引
// 让第 minIndex 个和第 0 个交换
var tmp = arr[minIndex]
arr[minIndex] = arr[0]

arr[0] = tmp
```

4. 按照数组的长度重复执行上面的代码

```
for (var j = 0; j < arr.length; j++) {
  // 因为第一遍的时候假定第 0 个，第二遍的时候假定第 1 个
  // 所以我们要假定第 j 个就行
  var minIndex = j

  // 因为之前已经把最小的放在最前面了，后面的循环就不需要判断前面的了
  // 直接从 j + 1 开始
  for (var i = j + 1; i < arr.length; i++) {
    if (arr[i] < arr[minIndex]) {
      minIndex = i
    }
  }

  // 遍历结束后找到最小的索引
  // 第一堂的时候是和第 0 个交换，第二趟的时候是和第 1 个交换
  // 我们直接和第 j 个交换就行
  var tmp = arr[minIndex]
  arr[minIndex] = arr[j]
  arr[j] = tmp
}
```

5. 一些优化

- 和之前一样，倒数第二次排序完毕以后，就已经排好了，最后一次没有必要了

```
for (var j = 0; j < arr.length - 1; j++) {
  var minIndex = j

  for (var i = j + 1; i < arr.length; i++) {
    if (arr[i] < arr[minIndex]) {
      minIndex = i
    }
  }

  var tmp = arr[minIndex]
  arr[minIndex] = arr[j]
  arr[j] = tmp
}
```

- 在交换变量之前，可以判断一下，如果我们遍历后得到的索引和当前的索引一直
- 那么就证明当前这个就是目前最小的，那么就没有必要交换
- 做一我们要判断，最小作引和当前作引不一样的时候，才交换

```
for (var j = 0; j < arr.length - 1; j++) {
  var minIndex = j

  for (var i = j + 1; i < arr.length; i++) {
```

```
      if (arr[i] < arr[minIndex]) {
        minIndex = i
      }
    }

    if (minIndex !== j) {
      var tmp = arr[minIndex]
      arr[minIndex] = arr[j]
      arr[j] = tmp
    }
  }
```

6. 至此，选择排序完成

快速排序

- 找中点，分左右，递归运算.....

```
function quickSort(arr){
    // 递归出口
    if (arr.length <= 1) return arr;
    // 找中点(中点的下标及值)
    var midIndex = parseInt( arr.length/2 );
    var mid = arr[midIndex];
    // 分左右
    var left = [];
    var right = [];
    for (var i = 0; i < arr.length; i++){
        if (arr[i] === mid) {
            continue; //跳过本次循环
        }
        if (arr[i] < mid) {//与中点比较分左右
            left.push(arr[i]);
        } else {
            right.push(arr[i]);
        }
    }
    // 递归运算(左中右三个数组合并)
    return quickSort(left).concat([mid],quickSort(right));
}
```