

JavaScript

JavaScript（以下简称JS）是一个编程语言，允许用户在浏览器页面上完成复杂的事情。浏览器页面并不总是静态的，往往显示一些需要动态更新的内容，交互式地图，动画，以及视频等。一个完整的JavaScript包括核心（ECMAScript），应用程序编程接口即API（比如DOM(Document Object Model)，BOM(Browser Object Model))，以及其他第三方API。JavaScript与HTML、CSS一同配合共同完成一个复杂页面的显示。

JS组成：ECMAScript、DOM、BOM

ECMAScript 6（简称ES6）是于2015年6月正式发布的JavaScript语言的标准，正式名为ECMAScript 2015（ES2015）

执行环境

浏览器

NodeJS环境

特点

1. 解释性语言
2. 被内置于浏览器或者NodeJS平台中的JS解析器解析执行，执行前无需编译
3. 弱类型语言
4. 从上往下顺序解析执行

1.JS类型

1. 内部JS, 写在`<script></script>`标签内的JS代码是内部JS
2. 外部JS, 使用`<script src=""></script>` script标签的src属性引入的js文件就是外部JS

1.1HelloWorld

在浏览器控制台输出HelloWorld

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <title>Document</title>

  <script>
    var str = 'Hello World';
    console.log(str);
  </script>

</head>

<body>

</body>

</html>
```

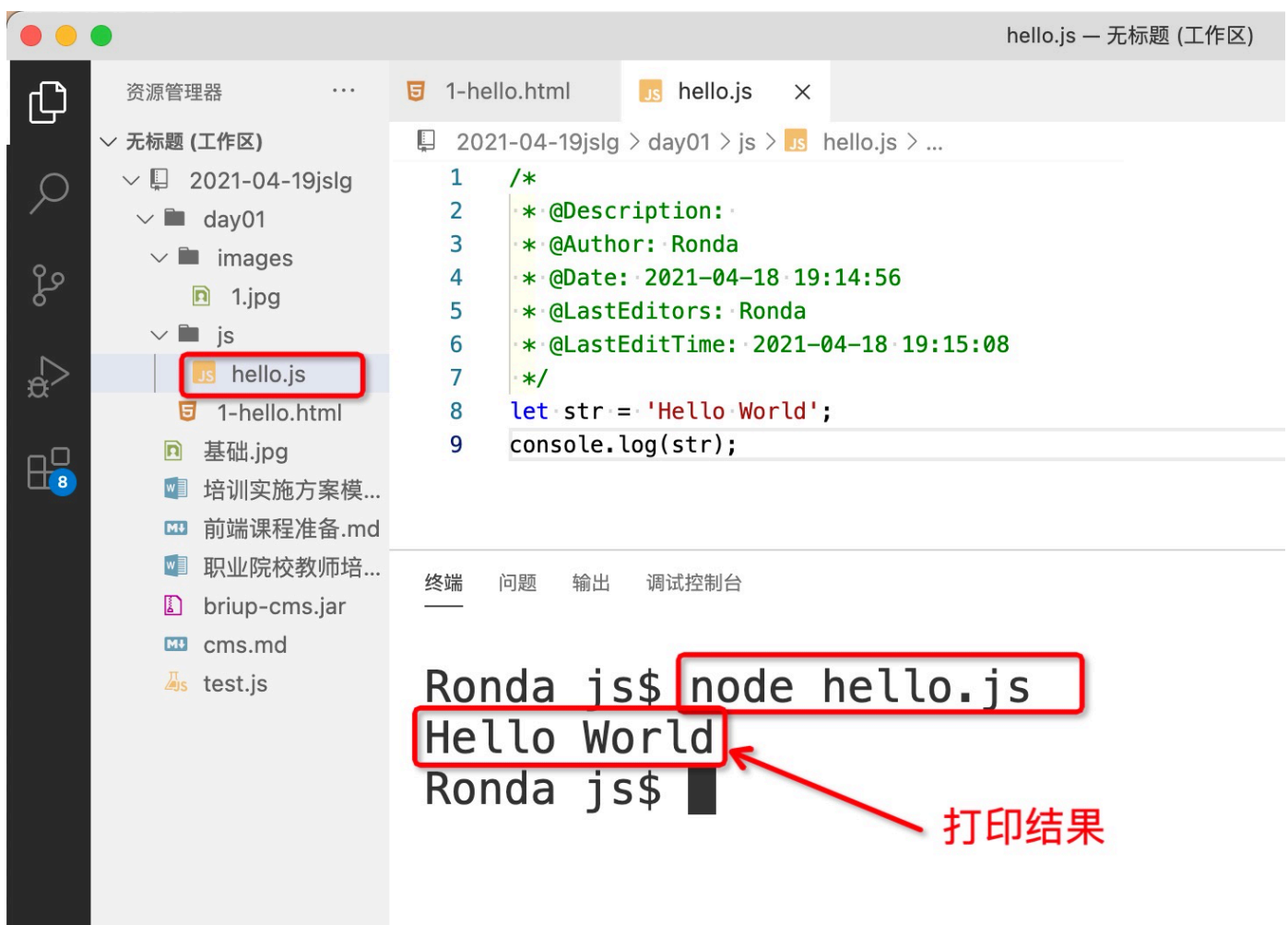
在html页面引入外部js文件

```
<!-- 用script标签的src属性指向外部js文件，则代表引入外部js文件 -->
<script src="./js/hello.js"></script>
```

在Nodejs环境中打印HelloWorld

创建一个hello.js文件，终端进入该文件所在目录，执行node hello.js命令即可

```
let str = 'Hello World';
console.log(str);
```



1.2注释

与绝大多数语言类似，JavaScript也需要注释来说明其代码含义，或者用来进行代码调试，注释后的代码会被浏览器忽略不被执行。

单行注释

```
// I am a comment
```

多行注释

```
/*  
  I am also  
  a comment  
*/
```

2.声明变量

2.1.var关键字

声明 `var message;`

初始化 `message = "hello"`

声明并初始化 `var message = "hello";`

定义多个变量 `var message= "hello",found=false, age = 29;`

变量名的命名规则：

- 变量名由字母，数字，下划线以及\$组成。
- 不要使用下划线或者数字作为变量名的开头
- 变量名应该具有一定的意义，使用小驼峰命名规则 `var userAgeTotal = "";`

- 不要使用关键字或是保留字
- 变量声明会被提升

```
console.log(a); //不会报错
var a = 3;
//等价于:
var a; //在所有代码执行之前, js解释器会将js中所有的var声明的变量提升。
console.log(a);
a=3;
```

在ES5中, 我们通常使用 `var` 声明一个变量, 但是 `var` 有很多奇葩的特性, 这些特性与常规的编程语言都不太相同, 在ES6中, 又提供了与常规语言类似的声明变量的方法。

2.2.let关键字

使用 `let` 来声明一个变量, `let` 对比 `var` 有如下特点:

- 变量声明不会被提升, 即在变量声明之前无法使用该变量。

```
// 报错, 不能在变量声明前使用该变量
console.log(a);
let a = 1;
```

- 具有局部作用域, 即 `let` 声明的变量只能在对应代码块中使用。

```
{
  let a = 1;
}
// 报错
console.log(a);
```

- 不允许重复声明

```
let a = 1;
// 下方代码报错，不能重复声明a变量
let a = 2;
```

2.3.const关键字

`const` 用于声明常量，`const` 具有与 `let` 相同的特性，此外还有一些其他特性。

- 常量声明不会被提升，即在变量声明之前无法使用该常量。

```
// 报错，不能在常量声明前使用该变量
console.log(b);
const b = 2;
```

- 具有局部作用域，即 `const` 声明的常量只能在对应代码块中使用。

```
{
  const b = 2;
}
// 报错
console.log(b);
```

- 不允许重复声明。

```
const b = 2;
// 下方代码报错，不能重复声明b常量
const b = 3;
```

- `const` 声明的常量在声明的时候就需要赋值，并且只能赋值一次，不能修改。

```
// 报错, 没有赋值
const b;
const c = 2;
// 报错, 重复赋值
c = 3;
```

3.数据类型

基本数据类型(Undefined, Null, Boolean, Number,String)

引用数据类型(Object,Array,Function)

3.1基本数据类型

Undefined类型

未定义类型 undefined

```
let a ;
let b = undefined;
```

Null类型

空引用数据类型 null '无' '空'

```
let a = null;
```

Boolean类型

布尔类型, 取值为 true/false, 通常用于条件判断

```
let a = false;  
let b = true;
```

Number类型

数字类型。整数/浮点数

```
let a = 100;
```

String类型

字符串类型，需要使用单引号或者双引号括起来

```
let a = "true";  
let b = '1';  
let c = 'hello';
```

引用数据类型

在JS中除了以上基本数据类型，其他所有类型都可以归结为引用数据类型。

对象Object

无序属性的集合，其属性可以包含基本值，对象，或者函数。可以将对象想象成散列表:键值对，其中值可以是数据或者函数。ECMAScript中的对象其实就是一组数据(属性)和功能(方法)的集合。

对象是一个包含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）

现实生活中，每一个人都是一个对象。对象有它的属性，如身高和体重，方法有走路或跑步等；所有人都有这些属性，但是每个人的属性都不尽相同，每个人都拥有这些方法，但是方法被执行的时间都不尽相同。

在JavaScript中，几乎所有的事物都是对象。

之前我们已经学写了JavaScript的变量

以下代码为变量 **person** 设置值为 "张三"：

```
let person = "张三";
```

对象也是一个变量，但对象可以包含多个值（多个变量），每个值以 **name:value** 键值对的方式 呈现。

```
let person = {name:"张三", height:1.78, gender: 'male'};
```

1.对象的创建

对象的初始化有两种方式，构造函数模式和字面量模式

2.字面量模式

对象使用"{}"作为对象的边界，对象是由多个属性组成，属性与属性之间通过","隔开，属性名与属性值通过":"隔开;属性名一般不添加引号（当属性名中出现特殊字符的时候需要添加引号），属性值如果是字符串的一定添加引号。

```
let obj = {  
  name: "terry",  
  age: 12,  
  sayName: function() {  
    console.log("my name is ", this.name);  
  }  
}
```

3.构造函数模式

使用Object或者使用自定义构造函数来初始化对象(例如Student)

```
let obj = new Object();  
obj.name = "terry";  
obj.age = 12;  
obj.sayName = function() {  
  console.log("my name is ", this.name);  
}  
//等价于 <==>  
let obj = {};  
obj.name = "terry";  
obj.age = 12;
```

4.对象的访问

- 属性访问

属性访问方式也有两种，点访问、中括号访问

点后面直接跟的是对象的属性，如果属性存在可以访问到，如果属性不存在，得到undefined。中括号中放的是变量，中括号可以将该变量进行解析。

```
obj.name          //'terry'  
let name = "name"  
obj['name']       //'terry'  
name = "age"  
obj['name']       //12
```

- 方法的访问

方法的访问主要是为了执行该对象中的方法，需要按照函数调用的方式使用

```
//以下执行结果不一样  
obj.sayName;  
obj.sayName(); //方法的使用
```

5.新增删除对象中的属性

只能删除对象的自有属性

```
delete obj.pro  
delete obj["prname"]  
delete obj.sayName //从obj对象中删除sayName属性
```

新增属性

```
obj.newprname="value"
```

数组Array

1.数组基础

ECMAScript数组是有序列表，是存放多个值的集合。

有以下特性：

每一项都可以保存任何类型的数据。

数组的大小是可以动态调整。

数组的length属性：可读可写，可以通过设置length的值从数组的末尾移除项或向数组中添加新项

js中的数组是可以存放任意数据类型值的集合，数组的元素可以是任意数据类型，数组的长度可以动态调整。

2.数组创建

- 字面量创建数组

由一对包括元素的方括号"[]"表示，元素之间以逗号","隔开

```
let names = ["terry", "larry", "tom"]
let name = "name"
let arr = [12, name, true, "larry", {}, function() {}, [], null];
console.log(arr, arr[2]);
console.log(arr.length); // arr.length
```

- 构造函数创建数组

通过Array构造函数来创建数组

```

let names = new Array();    // 等价于 let names = [];
// 一个参数, 如果是number类型的整数, 则代表的是数组的长度。如果是
number类型的小数, 则报错。如果是其他类型, 则当做数组元素放进去。
//let arr = new Array(length); 创建一个长度为length的数组
let names = new Array(3);
// 创建一个包含3个元素的数组 let arr =
[undefined,undefined,undefined];

let ages = new Array(2.4); //Error: Invalid array length

let names = new Array('terry') //创建一个包含1个元素的数组, 该
元素的值为'terry'

// 两个参数或者多个参数, 当做数组元素放进去
let names = new Array('terry', 'robin') //创建一个数组, 数组中的
元素使用实参初始化

```

3. 数组访问

通过索引访问数组, 数组的索引从0开始, 数组的索引超过数组长度会访问到undefined值而不会报错。数组的长度通过length属性获取

a) [index] 直接访问, 索引可以超过索引范围, 只不过访问的值为undefined

b) length-1=Max(index)

c) length+N length-N 开辟新的内存空间 数组元素的删除

```

let arr = ["terry", "larry", "tom"]
console.log(arr[0])    // "terry"
arr[7]=12;
console.log(arr, arr.length, arr[5], arr[7]) // [...] 8
undefined 12

```

数组的遍历：

普通的for循环、增强版for循环、while循环、do-while循环

```
let arr = [1,2,3];
for(let i=0;i<arr.length;i++){
  let item = arr[i];
}
for(let index in arr){
  let val = arr[index]
}
```

4.数组API

数组

1.数组基础

ECMAScript数组是有序列表，是存放多个值的集合。

有以下特性：

每一项都可以保存任何类型的数据。

数组的大小是可以动态调整。

数组的length属性：可读可写，可以通过设置length的值从数组的末尾移除项或向数组中添加新项

js中的数组是可以存放任意数据类型值的集合，数组的元素可以是任意数据类型，数组的长度可以动态调整。

2 数组创建

初始化

使用数组字面量

使用Array构造函数

- 字面量创建数组

由一对包括元素的方括号"[]"表示，元素之间以逗号","隔开

```
var names = ["terry", "larry", "tom"]
var name = "name"
// 可以接收任何类型的数据当做数组的元素
var arr = [12, name, true, "larry", {}, function() {}, [], null];
// 通过数组下标获取对应的数据
console.log(arr, arr[2]);
// 获取数组的长度
console.log(arr.length); // arr.length
```

- 构造函数创建数组

通过Array构造函数来创建数组

```
var names = new Array(); // 等价于 var names = [];
// 如果一个参数，并且是number类型的整数，则代表的是数组的长度。如果是number类型的小数，则报错。如果是其他类型，则当做数组元素放进去。
// var arr = new Array(length); 创建一个长度为length的数组
var names = new Array(3);
// 创建一个包含3个元素的数组 var arr =
[undefined, undefined, undefined];
// var ages = new Array(2.4); // Error: Invalid array length
var ages = new Array("2.4");
var names = new Array('terry') // 创建一个包含1个元素的数组，该元素的值为'terry'
// 两个参数或者多个参数，当做数组元素放进去
var names = new Array('terry', 'robin') // 创建一个数组，数组中的元素使用实参初始化
```

3. 数组访问

访问数组元素

数组变量名[索引]

```
//如果索引小于数组的长度，返回对应项的值
var arr = ["terry", "larry", "boss"];
arr[0] ;    //访问数组中第一个元素,返回值为terry
```

```
//如果索引大于等于数组的长度，返回undefined
var arr = ["terry", "larry", "boss"];
arr[10]     //undefined
```

```
//如果给索引大于等于数组的长度的位置设置了值，数组自动增加到该索引值加1的长度
var arr = ["terry", "larry", "boss"];
arr[3] = "jacky";    //添加元素,数组长度变为4
//注意！数组最多可以包含4 294 967 295个项
```

通过索引访问数组，数组的索引从0开始，数组的索引超过数组长度会访问到undefined值而不会报错。数组的长度通过length属性获取

a) [index] 直接访问,索引可以超过索引范围，只不过访问的值为undefined

b) length-1=Max(index)

c) length+N length-N 开辟新的内存空间 数组元素的删除

```
var arr = ["terry", "larry", "tom"]
// a.[index]
console.log(arr[0])    //"terry"
// b.length-1=Max(index)
```



```
arr[arr.length - 1]
// length+N length-N 并不是直接使用length加或减 而是设置新的长度
// 开辟新空间
arr.length = 4
//数组元素删除
arr.length = 2

// 数组的遍历:
// 普通的for循环、增强版for循环、while循环、do-while循环
var arr = [1,2,3];
for(var i=0;i<arr.length;i++){
    var item = arr[i];
}
for(var index in arr){
    var val = arr[index]
}
```

4.数组API

4.1.数组序列化

toString() 在默认情况下都会以逗号分隔字符串的形式返回数组项
join() 使用指定的字符串用来分隔数组字符串

```
var arr = [1,5,2,8,10,{a:1}];
console.log(arr);//[ 1, 5, 2, 8, 10, { a: 1 } ]
console.log(arr.toString());//"1,5,2,8,10,[object Object]"
console.log(arr.join("")); //"152810[object Object]"
console.log(arr.join("-")); //"1-5-2-8-10-[object Object]"

// 数组也可以使用序列化工具进行转换
var result = JSON.stringify(arr);
console.log(result);//"[1,5,2,8,10,{"a":1}]"
console.log(JSON.parse(result));//[ 1, 5, 2, 8, 10, { a: 1 } ]
```

4.2.构造函数的方法

- Array.isArray()

用来判断某个变量是否是一个数组对象

- Array.from()

从类数组对象或者可迭代对象中创建一个新的数组实例。

```
var myArr = Array.from("BRIUP");
console.log(myArr);
//输出结果为["B","R","I","U","P"]
```

- Array.of()

根据一组参数来创建新的数组实例，支持任意的参数数量和类型。

```
Array.of(7);           // [7]
Array.of(1, 2, 3);     // [1, 2, 3]
```

4.3.栈与队列方法

- **Array.prototype.push()**

push() 方法可向数组的末尾添加一个或多个元素，并返回新的长度。

注意： 新元素将添加在数组的末尾。

注意： 此方法改变数组的长度。

语法

```
array.push(item1, item2, ..., itemX)
```

参数值

参数	描述
<i>item1, item2, ..., itemX</i>	必需。要添加到数组的元素。

返回值

类型	描述
Number	数组新长度

数组中添加新元素：

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
// push() 添加元素到末尾 参数是要添加的元素 返回值是修改之后数  
组的长度【改变原数组】  
fruits.push("Kiwi")  
console.log(fruits);  
//fruits 结果输出: Banana,Orange,Apple,Mango,Kiwi
```

- **Array.prototype.pop()**

pop() 方法用于删除数组的最后一个元素并返回删除的元素。

注意： 此方法改变数组的长度！

语法

```
array.pop()
```

返回值

类型	描述
所有类型	返回删除的元素。

移除最后一个数组元素：

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.pop();
console.log(fruits);
//fruits 结果输出: Banana,Orange,Apple
```

- **Array.prototype.shift()**

shift() 方法用于把数组的第一个元素从其中删除，并返回第一个元素的值。

注意： 此方法改变数组的长度！

语法

```
array.shift()
```

返回值

类型	描述
任何类型 (*)	数组原来的第一个元素的值（移除的元素）。

从数组中移除元素:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.shift()
console.log(fruits);
//fruits结果输出:Orange,Apple,Mango
```

- **Array.prototype.unshift()**

unshift() 方法可向数组的开头添加一个或更多元素，并返回新的长度。

注意： 该方法将改变数组的数目。

语法

```
array.unshift(item1,item2, ..., itemX)
```

参数值

参数	描述
<i>item1,item2, ..., itemX</i>	可选。向数组起始位置添加一个或者多个元素。

返回值

Type 描述 Number 数组新长度

将新项添加到数组起始位置:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.unshift("Lemon", "Pineapple");
console.log(fruits)
//fruits 将输出:
Lemon, Pineapple, Banana, Orange, Apple, Mango
```

4.4.排序方法

- **Array.prototype.reverse()**

reverse() 方法用于颠倒数组中元素的顺序。

语法

```
array.reverse()
```

返回值

类型	描述
Array	颠倒顺序后的数组

颠倒数组中元素的顺序:

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.reverse();
console.log(fruits)
//fruits结果输出: Mango, Apple, Orange, Banana
```

- **Array.prototype.sort()**

sort() 方法用于对数组的元素进行排序。如果调用该方法时没有使用参数，将按字母顺序（Ascall编码）对数组中的元素进行排序，说得更精确点，是按照字符编码的顺序进行排序。要实现这一点，首先应把数组的元素都转换成字符串（如有必要），以便进行比较。如果要想进行升序或是降序排序的话，要提供比较函数。

排序顺序可以是字母或数字，并按升序或降序。

默认排序顺序为按字母升序。

语法

```
array.sort(sortfunction)
```

参数值

参数	描述
<i>sortfunction</i>	可选。规定排序顺序。必须是函数。

数组排序：

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
fruits.sort();
console.log(fruits)
//fruits 输出结果: Apple,Banana,Mango,Orange
```

注意：当数字是按字母顺序排列时"40"将排在"5"前面。

使用数字排序，你必须通过一个函数作为参数来调用。

函数指定数字是按照升序还是降序排列。

关于升序和降序的条件

当 $a > b$ 时,

$a - b > 0$, 排序结果 $\implies b, a$ (升序)

$b - a < 0$, 排序结果 $\implies a, b$ (降序)

当 $b > a$ 时,

$a - b < 0$, 排序结果 $\implies a, b$ (升序)

$b - a > 0$, 排序结果 $\implies b, a$ (降序)

当 $a = b$ 时,

$a - b = b - a = 0$, 排序结果 \implies 保持不变

无论 $a > b$ 还是 $b > a$, `return a-b` 总能得到升序的结果, 而 `return b-a` 总能得到降序的结果。

数字排序（升序）：

```
var points = [40,100,1,5,25,10];
points.sort(function(a,b){
    return a-b
});
//points输出结果: 1,5,10,25,40,100
```

数字排序（降序）：

```
var points = [40,100,1,5,25,10];
points.sort(function(a,b){
    return b-a
});
//points输出结果: 100,40,25,10,5,1
```


4.5.操作方法

- **Array.prototype.concat()**

concat() 方法用于连接两个或多个数组。

该方法不会改变现有的数组，而仅仅会返回被连接数组的一个副本。

语法

```
array1.concat(array2,array3,...,arrayX)
```

参数值

参数	描述
<i>array2, array3, ..., arrayX</i>	必需。该参数可以是具体的值，也可以是数组对象。可以是任意多个。

返回值

Type	描述
Array 对象	返回一个新的数组。该数组是通过把所有 arrayX 参数添加到 arrayObject 中生成的。如果要进行 concat() 操作的参数是数组，那么添加的是数组中的元素，而不是数组。

合并三个数组的值：

```
var hege = ["Cecilie", "Lone"];
var stale = ["Emil", "Tobias", "Linus"];
var kai = ["Robin"];
var children = hege.concat(stale,kai);
//children 输出结果：Cecilie,Lone,Emil,Tobias,Linus,Robin
```

• **Array.prototype.slice()**

slice() 方法可从已有的数组中返回选定的元素。

slice()方法可提取字符串的某个部分，并以新的字符串返回被提取的部分。

语法

```
array.slice(start, end)
```

参数值

参数	描述
start	可选。规定从何处开始选取。如果是负数，那么它规定从数组尾部开始算起的位置。如果该参数为负数，则表示从原数组中的倒数第几个元素开始提取，slice(-2) 表示提取原数组中的倒数第二个元素到最后一个元素（包含最后一个元素）。
end	可选。规定从何处结束选取。该参数是数组片断结束处的数组下标。如果没有指定该参数，那么切分的数组包含从 start 到数组结束的所有元素。如果该参数为负数，则它表示在原数组中的倒数第几个元素结束抽取。slice(-2,-1) 表示抽取了原数组中的倒数第二个元素到最后一个元素（不包含最后一个元素，也就是只有倒数第二个元素）。

返回值

Type	描述
Array	返回一个新的数组，包含从 start 到 end （不包括该元素）的 arrayObject 中的元素。

在数组中读取元素：

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

- **Array.prototype.splice()**

splice() 方法用于添加或删除数组中的元素。

注意：这种方法会改变原始数组。

语法

```
array.splice(index,howmany,item1,.....,itemX)
```

参数

参数	描述
<i>index</i>	必需。规定从何处添加/删除元素。该参数是开始插入和（或）删除的数组元素的下标，必须是数字。
<i>howmany</i>	可选。规定应该删除多少元素。必须是数字，但可以是"0"。如果未规定此参数，则删除从 index 开始到原数组结尾的所有元素。
<i>item1, ..., itemX</i>	可选。要添加到数组的新元素

返回值

Type	描述
Array	如果从 arrayObject 中删除了元素，则返回的是含有被删除的元素的数组。

数组中添加新元素：

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
//从数组索引位置 2 开始，不删除元素，添加两个新的元  
素"Lemon","Kiwi"  
fruits.splice(2,0,"Lemon","Kiwi");  
//fruits输出结果: Banana,Orange,Lemon,Kiwi,Apple,Mango
```

移除数组的第三个元素，并在数组第三个位置添加新元素：

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2,1,"Lemon","Kiwi");  
  
//fruits输出结果: Banana,Orange,Lemon,Kiwi,Mango
```

从第三个位置开始删除数组后的两个元素：

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2,2);  
  
//fruits输出结果: Banana,Orange
```

4.6.位置方法

- **Array.prototype.indexOf()**

indexOf() 方法可返回数组中某个指定的元素位置。

该方法将从头到尾地检索数组，看它是否含有对应的元素。开始检索的位置在数组 start 处或数组的开头（没有指定 start 参数时）。如果找到一个 item，则返回 item 的第一次出现的位置。开始位置的索引为 0。

如果在数组中没找到指定元素则返回 -1。

语法

```
array.indexOf(item,start)
```

参数

参数	描述
item	必须。查找的元素。
start	可选的整数参数。规定在数组中开始检索的位置。它的合法取值是 0 到 stringObject.length - 1。如省略该参数，则将从字符串的首字符开始检索。

返回值

类型	描述
Number	元素在数组中的位置，如果没有搜索到则返回 -1。

查找数组中的 "Apple" 元素：

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];
var a = fruits.indexOf("Apple");

//a结果输出：2
//以上输出结果意味着 "Apple" 元素位于数组中的第 3 个位置。
```

- **Array.prototype.lastIndexOf()**

lastIndexOf() 方法可返回一个指定的元素在数组中最后出现的位置，从该字符串的后面向前查找。

如果要检索的元素没有出现，则该方法返回 -1。

该方法将从尾到头地检索数组中指定元素 item。开始检索的位置在数组的 start 处或数组的结尾（没有指定 start 参数时）。如果找到一个 item，则返回 item 从尾向前检索第一个次出现在数组的位置。数组的索引开始位置是从 0 开始的。

如果在数组中没找到指定元素则返回 -1。

语法

```
array.lastIndexOf(item,start)
```

参数

参数	描述
item	必需。规定需检索的字符串值。
start	可选的整数参数。规定在字符串中开始检索的位置。它的合法取值是 0 到 stringObject.length - 1。如省略该参数，则将从字符串的最后一个字符处开始检索。

返回值

Type	描述
Number	如果在 stringObject 中的 fromindex 位置之前存在 searchvalue，则返回的是出现的最后一个 searchvalue 的位置。

查找数组元素 "Apple"出现的位置：

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
var a = fruits.lastIndexOf("Apple");
```

//a输出结果：2

//以上实例输出结果意味着 "Apple" 位于数组中的第 2 个位置。

4.7.迭代方法

• Array.prototype.every()

every() 方法用于检测数组所有元素是否都符合指定条件（通过函数提供）。

every() 方法使用指定函数检测数组中的所有元素：

- 如果数组中检测到有一个元素不满足，则整个表达式返回 *false*，且剩余的元素不会进行检测。
- 如果所有元素都满足条件，则返回 *true*。

注意： every() 不会对空数组进行检测。

注意： every() 不会改变原始数组。

语法

```
array.every(function(currentValue,index,arr),  
thisValue)
```

参数

参数	描述
<i>function(currentValue, index, arr)</i>	必须。函数，数组中的每个元素都会执行这个函数 函数参数见下表
<i>thisValue</i>	可选。对象作为该执行回调时使用，传递给函数，用作 "this" 的值。如果省略了 thisValue，"this" 的值为 "undefined"

参数	描述
<i>currentValue</i>	必须。当前元素的值
<i>index</i>	可选。当前元素的索引值
<i>arr</i>	可选。当前元素属于的数组对象

返回值

布尔值。如果所有元素都通过检测返回 true，否则返回 false。

检测数组 **ages** 的所有元素是否都大于等于 18：

```
var ages = [32, 33, 16, 40];

var result = ages.every(function (age) {
    return age >= 18
})
console.log(result); //输出结果为:false
```

- **Array.prototype.some()**

some() 方法用于检测数组中的元素是否满足指定条件（函数提供）。

some() 方法会依次执行数组的每个元素：

- 如果有一个元素满足条件，则表达式返回`true`，剩余的元素不会再执行检测。
- 如果没有满足条件的元素，则返回`false`。

注意： some() 不会对空数组进行检测。

注意： some() 不会改变原始数组。

语法

```
array.some(function(currentValue,index,arr),thisValue)
```

参数

参数	描述
<code>function(currentValue, index,arr)</code>	必须。函数，数组中的每个元素都会执行这个函数 函数参数见下表
<code>thisValue</code>	可选。对象作为该执行回调时使用，传递给函数，用作 "this" 的值。如果省略了 thisValue，"this" 的值为 "undefined"

参数	描述
<code>currentValue</code>	必须。当前元素的值
<code>index</code>	可选。当前元素的索引值
<code>arr</code>	可选。当前元素属于的数组对象

返回值

布尔值。如果数组中有元素满足条件返回 true，否则返回 false。

检测数组中是否有元素大于 18:

```
var ages = [32, 33, 16, 40];

var result = ages.some(function (age) {
    return age >= 18
})
console.log(result); //输出结果为:true
```

- **Array.prototype.filter()**

filter() 方法创建一个新的数组，新数组中的元素是通过检查指定数组中符合条件的所有元素。

注意： filter() 不会对空数组进行检测。

注意： filter() 不会改变原始数组。

语法

```
array.filter(function(currentValue,index,arr),
thisValue)
```

参数

参数	描述
<i>function(currentValue, index, arr)</i>	必须。函数，数组中的每个元素都会执行这个函数 函数参数见下表
<i>thisValue</i>	可选。对象作为该执行回调时使用，传递给函数，用作 "this" 的值。如果省略了 thisValue ， "this" 的值为 "undefined"

参数	描述
<i>currentValue</i>	必须。当前元素的值
<i>index</i>	可选。当前元素的索引值
<i>arr</i>	可选。当前元素属于的数组对象

返回值

返回数组，包含了符合条件的所有元素。如果没有符合条件的元素则返回空数组。

返回数组 **ages** 中所有元素都大于 18 的元素:

```
var ages = [32, 33, 16, 40];

var result = ages.filter(function (age) {
    return age >= 18
})
console.log(result); //输出结果为:[ 32, 33, 40 ]
```

- **Array.prototype.map()**

map() 方法返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。

map() 方法按照原始数组元素顺序依次处理元素。

注意： map() 不会对空数组进行检测。

注意： map() 不会改变原始数组。

语法

```
array.map(function(currentValue,index,arr), thisValue)
```

参数

参数	描述
<i>function(currentValue, index,arr)</i>	必须。函数，数组中的每个元素都会执行这个函数 函数参数见下表
<i>thisValue</i>	可选。对象作为该执行回调时使用，传递给函数，用作 "this" 的值。如果省略了 thisValue，或者传入 null、undefined，那么回调函数的 this 为全局对象。

参数	描述
<i>currentValue</i>	必须。当前元素的值
<i>index</i>	可选。当前元素的索引值
<i>arr</i>	可选。当前元素属于的数组对象

返回值

返回一个新数组，数组中的元素为原始数组元素调用函数处理后的值。

返回一个数组，数组中元素为原始数组的平方根：

```
var numbers = [4, 9, 16, 25];

var result = numbers.map(function (number) {
    return Math.sqrt(number)
})
console.log(result); //输出结果为:[ 2, 3, 4, 5 ]
```

- **Array.prototype.forEach()**

forEach() 方法用于调用数组的每个元素，并将元素传递给回调函数。

注意: forEach() 对于空数组是不会执行回调函数的。

语法

```
array.forEach(function(currentValue, index, arr),
thisValue)
```

参数

参数	描述
<i>function(currentValue, index, arr)</i>	必需。数组中每个元素需要调用的函数。 函数参数见下表
<i>thisValue</i>	可选。传递给函数的值一般用 "this" 值。 如果这个参数为空， "undefined" 会传递给 "this" 值

参数	描述
<i>currentValue</i>	必需。当前元素
<i>index</i>	可选。当前元素的索引值。
<i>arr</i>	可选。当前元素所属的数组对象。

返回值

undefined

列出数组的每个元素：

```
var numbers = [4, 9, 16, 25];

numbers.forEach(function (number, index) {
    console.log(index + ' -- ' + number);
})

// 0 -- 4
// 1 -- 9
// 2 -- 16
// 3 -- 25
```

5.函数Function

1.函数介绍

函数介绍

函数允许我们封装一系列代码来完成特定任务。当想要完成某一任务时，只需要调用相应的代码即可。方法（method）一般为定义在对象中的函数。浏览器为我们提供了很多内置方法，我们不需要编写代码，只需要调用方法即可完成特定功能。

函数的作用：

功能的封装，直接调用，代码复用率提高

构建对象的模板（构造函数）

函数实际上是对象，每个函数都是Function类型的实例，并且都与其他引用类型一样具有属性和方法，由于函数是对象，因此函数名实际上也是一个指向函数对象的指针，不会与某个函数绑定。

2.函数声明

自定义函数

函数由function关键字声明，后面紧跟函数名，函数名后面为形参列表，列表后大括号括起来的内容为函数体。也可以将一个匿名函数（没有函数名的函数）赋值给一个函数变量，这种方式称为函数表达式。

解析器在向执行环境中加载数据时，会率先读取函数声明，并使其在执行任何代码之前可用；当执行器执行到函数表达式的代码的时候才会真正的解释执行。

表示方法:

函数声明

```
function 函数名(形参列表){  
    //函数体  
}
```

函数表达式

```
let 函数名 = function(形参列表){  
    //函数体  
}
```

- 函数声明

函数声明与var变量声明类似，会进行提升

```
function add(a,b){
    let result = a + b;
    return result;
    //返回值//返回执行的结果给被调用的
}
let total = add(1,2)

foo();//函数声明提升到代码的最前边，可以直接调用函数
function foo(){
    console.log("hello world");
    //return;
    //console.log("1");//return之后的语句不执行
    //如果没有返回的内容，则在写代码的时候不关注返回值
    //没有return:代码执行到大括号
}
```

3.arguments

ECMAScript函数的参数与大多数其他语言中的函数的参数有所不同，ECMAScript函数不介意传递参数的个数以及参数类型，这是因为函数的参数在函数内容是使用一个类数组对象来表示的。这个类数组对象就是arguments。

arguments是一个类数组对象，包含着传入函数中的所有参数。arguments主要用途是保存函数参数，但是这个对象还有一个名为callee的属性，该属性是一个指针，指向拥有这个arguments对象的函数。

```
//length声明时希望的参数的个数
function add(a,b){
    var result = a + b;
    return result;
}
```


`console.log(add.length);` //表示函数希望接受的命名参数的个数，即形参的个数。

```
function add(a,b){
    console.log(arguments[0],arguments[1],arguments[2],arguments[3]);
    console.log(a+b);
}
add(10);
//10 undefined undefined undefined
//NaN
add(10,20);
//10 20 undefined undefined
//30
add(10,20,30);
//10 20 30 undefined
//30
add(10,20,30,40);
//10 20 30 40
//30
```

4.this

面向对象语言中 this 表示当前对象的一个引用。

但在 JavaScript 中 this 不是固定不变的，它会随着执行环境的改变而改变。

- 在方法中，this 表示该方法所属的对象。
- 如果单独使用，this 表示全局对象。
- 在函数中，this 表示全局对象。
- 在事件中，this 表示接收事件的元素。

实例

```
var person = {
  firstName: "LeBron",
  lastName : "James",
  id        : 23,
  fullName  : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

方法中的 this

在对象方法中， this 指向调用它所在方法的对象。

在上面一个实例中， this 表示 person 对象。

fullName 方法所属的对象就是 person。

```
fullName : function() { return this.firstName + " " +
this.lastName; }
```

单独使用 this

单独使用 this，则它指向全局(Global)对象。

在浏览器中， window 就是该全局对象为 **[object Window]**:

```
var x = this;
```

函数中使用 this（默认）

在函数中，函数的所有者默认绑定到 this 上。

在浏览器中， window 就是该全局对象为 **[object Window]**:

```
function myFunction() { return this; }
```

事件中的 this

在 HTML 事件句柄中，this 指向了接收事件的 HTML 元素：

```
<button onclick="this.style.display='none'"> 点我后我就消失了  
</button>
```

4.流程控制语句

4.1.if语句

condition表示任意表达式，该表达式求值的结果不一定是布尔类型，如果不是布尔类型，ECMAScript会调用Boolean() 转换函数将这个表达式结果转换为一个布尔类型，当该值为true时，执行if代码块中的内容。

```
if(condition){  
    statement  
}
```

当condition为true时，执行if代码块中的内容，否则，执行else代码块中的内容，一般情况下，如果代码块中代码只有一行，可以省略大括号。

```
if(condition){  
    statement1  
} else {  
    statement2  
}
```

多条件分支，当condition1为true时，执行statement1,否则当condition2为true时执行statement2，当condition1,condition2都为false的时候执行statement3。

```
if(condition1){  
    statement1  
} else if(condition2){  
    statement2  
} else {  
    statement3  
}
```

4.2.switch语句

switch 语句用于基于不同条件执行不同动作。

- 计算一次 switch 表达式
- 把表达式的值与每个 case 的值进行对比
- 如果存在匹配，则执行关联代码

```
switch(表达式) {  
    case n:  
        代码块  
        break;  
    case n:  
        代码块  
        break;  
    default:  
        默认代码块  
}
```

使用switch输出今天是星期几。

```
let day = '';  
switch (new Date().getDay()) {  
  case 0:  
    day = "星期天";  
    break;  
  case 1:  
    day = "星期一";  
    break;  
  case 2:  
    day = "星期二";  
    break;  
  case 3:  
    day = "星期三";  
    break;  
  case 4:  
    day = "星期四";  
    break;  
  case 5:  
    day = "星期五";  
    break;  
  case 6:  
    day = "星期六";  
}  
console.log(`今天是${day}`);
```

`new Date()`用来获取当前日期。

`new Date().getDay()` 方法返回 0 至 6 之间的周名数字（weekday number）。

4.3.循环

循环有for(){ }循环，while(){ }循环，do{}while()循环

循环三要素：初始条件，结束条件，迭代条件

Initializer，初始化值，一般为数字，仅会执行一次。也可以写到循环体外
exit-condition，结束条件，通常使用逻辑运算符进行结束循环判断。每次执行循环体之前均会执行该代码。

final-expression，每次执行完循环体代码后执行，通常用于迭代使其更加靠近结束条件。

以for循环为例：

```
for(Initializer;exit-condition;final-expression){  
    //to do  
}
```

基本使用

```
for (let i = 1; i < 10; i++) {  
    console.log(i);  
}
```

5.箭头函数

JavaScript中，经常使用回调函数，箭头函数的出现大大简化了回调函数的写法，当然，除了作为函数参数，箭头函数也可以出现在其他地方。

```
// 箭头函数  
  
// 没有形参 放一个()就可以  
let test = () => {}
```

// 等同于

```
function test() {}
```

// 只有一个形参 可以省略() 直接设置形参

```
let test = a => {}
```

// 等同于

```
function test(a) {}
```

// 有两个或多个形参 ()不能省略 逗号隔开

```
let test = (a, b) => {}
```

// 等同于

```
function test(a, b) {}
```

// 箭头函数的返回值

// 函数代码块中有多行代码时 我们不能省略{}

```
let test = () => {  
  console.log('Hello World');  
  console.log('Hello ES6');  
}  
test()
```

// 函数代码块中只有一行代码 该代码为返回操作时 我们可以省略{}和
return

```
const test = (num1, num2) => {  
  return num1 + num2  
}  
const test = (num1, num2) => num1 + num2  
console.log(test2(10, 20));
```

// 函数代码块中只有一行代码 我们可以省略{ }

```
const test = () => {
```

```
console.log('Hello World');  
}  
const test = () => console.log('Hello World')  
test()
```

6. 解构

ES6提供的一种高级的声明变量的方式：解构，准确来说是一种模式匹配。

对象解构

等号左边的变量放到大括号内部，匹配右侧对象中的数据。对象的属性没有次序，变量必须与属性同名，才能取到正确的值

```
let { foo, bar } = { foo: "aaa", bar: "bbb" }; // foo =  
"aaa"; bar = "bbb"
```

如果变量名与属性名不一致，必须写成下面这样。

```
let { foo: baz } = { foo: 'aaa', bar: 'bbb' }; //baz =  
"aaa"
```

这实际上说明，对象的解构赋值是下面形式的简写。

```
let { foo: foo, bar: bar } = { foo: "aaa", bar: "bbb" };
```

数组解构

```
let [a, b, c] = [1, 2, 3];
```