**ES5的严格模式：**

所谓严格模式，从字面上就很好理解，即更严格的模式 在这种模式下执行，浏览器会对JS的要求更苛刻，语法格式要求更细致，更符合逻辑。

怪异模式：就是我们之前一直使用的开发模式，就叫怪异模式。因为很多时候出来的结果是非常怪异的，所以才称之为怪异模式。

**全局严格模式**

"use strict";

n = 10;

console.log(n);

**局部严格模式**

function fn(){

"use strict";

n = 10

console.log(n);

}

**设立严格模式的目的**

1.消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;

2.代码运行的一些不安全之处，保证代码运行的安全；

3.提高编译器效率，增加运行速度；

**严格模式的特点：**

**1. 不可以省略var let const声明变量**

不用关键字声明变量会报错

**2. 禁止函数使用this关键字指向全局变量**

严格模式下this指向undefined

**3. 禁止使用八进制方法**

8进制的数以0开头 ，使用会报错

**4. 不允许在非函数的代码块内声明函数**

大括号内不能再声明函数了 除非最外层的大括号也是一个函数

**5. 严格模式下,arguments变量,形参是不会变(不同步)**

怪异模式下可以直接修改形参 并且arguments数组是同步的

严格模式下能修改形参，但是arguments数组不会同步，不会报错，但是不会生效

**6.eval作用域：**

var x=20;

eval(" 'use strict'; var x=10");

console.log(x)//20

严格模式下 eval（）函数有单独的函数作用域

eval（）函数内部的变量声明和修改不会影响全局变量，

怪异模式下eval函数内部可以修改全局变量

**7.delete变量报错**

在严格模式下，如果对一个变量的直接引用、函数的参数或者函数名使用delete操作，将会抛出语法错误（SyntaxError）。因此，为避免严格模式下的语法错误，必须以delete object.property或delete object[‘property’]的形式使用delete运算符。

如果你试图删除的属性不存在，那么delete将不会起任何作用，但仍会返回true任何使用 var 声明的属性不能从全局作用域或函数的作用域中删除。

这样的话，delete操作不能删除任何在全局作用域中的函数（无论这个函数是来自于函数声明或函数表达式）

除了在全局作用域中的函数不能被删除，在对象(object)中的函数是能够用delete操作删除的。

任何用let或const声明的属性不能够从它被声明的作用域中删除。

不可设置的(Non-configurable)属性不能被移除。这意味着像Math, Array, Object内置对象的属性以及使用Object.defineProperty()方法设置为不可设置的属性不能被删除。

**this的指向**

1.在函数内部，this默认指向window

2.在严格模式下，函数内部的this指向undefined

3.在事件里，this默认指向当前事件的绑定对象

4.箭头函数内部的this，指向上下文对象(环境变量)

5.new操作符，改变this指向

6.定时器，延时器的函数里，this默认指向window

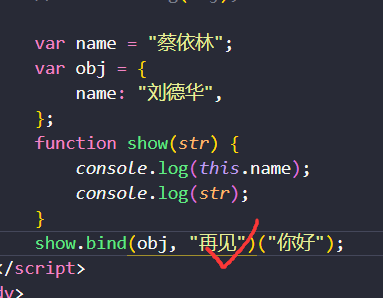
7.对象的方法中的this，默认指向该对象

8.bind函数，改变this的指向，但是bind不会立即执行，所以要加小括号

在定时器延时器的函数后面直接加.bind(obj)

给bind传的参数会直接传进被调用的函数且优先级更高

用call()和apply()调用函数也可以改变this的指向 会立即执行



**ES5数组扩展**

some()、every()、map()、forEach()、filter()、reduce()

**some()**

只要有一个元素满足条件就返回true，否则返回false

var arr = [11, 22, 33, 44, 55];

var res = arr.some(function (item, index, arr) {

return item > 30;

});

var res = arr.some(item => item>30);//箭头函数

console.log(res); //true

**every()**

所有元素都满足条件就返回true，否则返回false

var arr = [11, 22, 33, 44, 55];

var res = arr.every(function (item, index, arr) {

    return item > 20;

});

var res = arr.some(item => item > 20);//箭头函数

console.log(res); //false

**forEach()**

它只负责遍历， break和return 不能中断foreach的循环

想要中断可以强制抛出异常，结合try....catch

try {

arr.forEach(function (item, index, arr) {

console.log(item, index, arr);

    if (item > 30) {

throw new Error("");

}

});

} catch (error) {

}

**map()**

能做遍历，还能返回结果，结果是一个新数组

var arr = [11, 22, 33, 44, 55];

var arr2 = arr.map(function (item, index, arr) {

return item \* 10;

});

console.log(arr2);// [110, 220, 330, 440, 550]

**filter()**

过滤出数组中你想要的元素， 不改变原数组

var arr = [11, 22, 33, 44, 55];

var arr2 = arr.filter(function (item, index, arr) {

return item > 30;

});

console.log(arr2);//[33, 44, 55]

**reduce()**

参数：第一个为函数

函数：4个形参

1.pre 初始值，也可以是上一次执行的结果

如果有初始值 一开始pre是初始值

如果函数没有返回结果 第二次循环pre是undefined；

如果有返回值 第二次pre为上一次执行的结果

如果没有初始值 一开始pre是数组的第一个元素

cur为数组的第二个元素

2.cur 当前的元素

3.index 下标

4.arr 原来的数组

第二个为p初始值

var arr = [11, 22, 33, 44, 55];

var res = arr.reduce(function (pre, cur) {

pre.push(cur \* 10);

return pre;

}, []);

console.log(res);// [110, 220, 330, 440, 550];

**forEach、map以及reduce的相同点：**

都是用于处理数组

都不兼容IE8及以下

**forEach、map以及reduce的不同点：**

forEach 方法是将数组中的每一个值取出做一些程序员想让他们做的事情

map 方法是将数组中的每一个值放入一个方法中做一些程序员想让他们做的事情后可以返回一个新的数组

reduce 方法 将数组中的每一个值与前面的值处理后得到的最终值

**ES6**

**let 块级作用域**

利用块级作用域可以有效解决以下问题

1.防止全局变量泄露

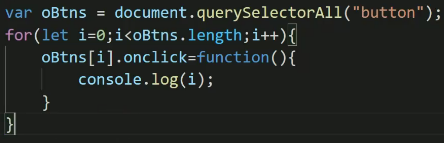
2.防止变量提升带来的覆盖问题

严格遵循 先定义 后使用 避免变量的提前声明

在代码块内声明就只能在代码块内使用{ }

**let解决异步事件的原理**

用let声明变量绑定响应函数的时候i不用藏值了



let 与var相比区别是，在for循环中用var就是一个全局的i，所以每次会被后面的给替换。使用let 循环几次就会生成几个i，也就是说在每次进入循环的时候都会把之前的i赋值给一个新的i

// let底层原理就是匿名函数自调，闭包

for (let i = 0; i < btns.length; i++) {

// (function(i){

btns[i].onclick = function() {

console.log(i);

}

// })(i);

}

**const**

const 声明的是常量，一旦声明，值将是不可变的

const 也具有块级作用域 , const 不可重复声明

const PI = 3.1415;

console.log(PI);

PI = 3;

console.log(PI); //报错

因为引用类型的变量在栈内存中存储的是（在堆内存中开辟的一个空间的地址）

所以只要栈内存中存放的地址不被修改，修改堆内存中的内容是可以的

**let和const有什么区别？**

都是ES6提出的新的声明变量的关键字

都能避免变量提升（提前声明）

都能实现块级作用域

在同一个代码块内 同一个变量只能被声明一次

let声明变量的值可以被修改，

而const如果是值类型不可以被修改，

如果是引用类型地址不能被修改，属性和方法可以修改

**ES6-String新增方法:**

传统上，JavaScript只有 indexOf 方法，可以用来确定一个字符串是否包含在另一个字符串中。ES6又提供了三种新方法。

**includes()：**

返回布尔值，表示是否找到了参数字符串。

console.log(str.includes("帅"));

console.log(str.indexOf("帅") > 0);//等价于这种写法

**startsWith()：**

返回布尔值，表示参数字符串是否在源字符串的头部。

console.log(str.startsWith("台"));

**endsWith()：**

返回布尔值，表示参数字符串是否在源字符串的尾部。

console.log(str.endsWith("帅"));

**repeat()：**

返回一个新字符串，表示将原字符串重复n次.

console.log(str.repeat(1000));

**模板字符中，支持字符串插值**

document.write(`Hello ${first} ${last}!`); //注意引号

**padStart()、padEnd()**

ES2017 引入了字符串补全长度的功能。如果某个字符串不够指定长度，会在头部或尾部补全。padStart()用于头部补全，padEnd()用于尾部补全。

一共接受两个参数，第一个参数用来指定字符串的最小长度，第二个参数是用来补全的字符串。

如果原字符串的长度，等于或大于指定的最小长度，则返回原字符串。

如果用来补全的字符串与原字符串，两者的长度之和超过了指定的最小长度，则会截去超出位数的补全字符串。

如果省略第二个参数，默认使用空格补全长度。

'1'.padStart(10, '0') // "0000000001"

'12'.padStart(10, '0') // "0000000012"

'123456'.padStart(10, '0') // "0000123456"

另一个用途是提示字符串格式。

'12'.padStart(10, 'YYYY-MM-DD') // "YYYY-MM-12"

'09-12'.padStart(10, 'YYYY-MM-DD') // "YYYY-09-12"

**ES6数组扩展**

**Array.from()**

用于将两类对象转为真正的数组：

类似数组的对象（array-like object）

就是一个对象多了一个length属性

可遍历（iterable）的对象（包括ES6新增的数据结构Set和Map）

Array.from(arrayLike, mapFn, thisArg)

Array.from()还可以接受第二个参数，作用类似于数组的map方法，用来对每个元素进行处理.

var oBtns = document.getElementsByTagName("button");

console.log(oBtns);//伪数组 不能forEach

oBtns = Array.from(oBtns);

console.log(oBtns);//数组 可以forEach

oBtns.forEach(function (item, index) {

item.onclick = function () {

console.log(index);

};

});

**Array.of():**

方法用于将一组值，转换为数组

var arr = Array.of(11, 22, 33, 44);

console.log(arr);//[11,22,33,44]

**find()**

数组实例的find方法，用于找出第一个符合条件的数组成员。它的参数是一个回调函数，所有数组成员依次执行该回调函数，直到找出第一个返回值为true的成员，然后返回该成员。如果没有符合条件的成员，则返回undefined。

var arr = [{ name: "刘德华", age: 20 },{ name: "周星驰", age: 40 },];

var sName = "刘德华";

var obj = arr.find(function (item, index, arr) {

//当item.name == sName成立时,返回数组中符合条件的元素

return item.name == sName;

});

var obj = arr.find(item => item.name == sName);//箭头函数

console.log(obj);//{ name: "刘德华", age: 20 }

**findIndex()**

var arr = [{ name: "刘德华", age: 20 },{ name: "周星驰", age: 40 },];

var str = "周星驰";

var index = arr.findIndex(function (item) {

return item.name === str;

});

var index = arr.find(item => item.name == sName);//箭头函数

console.log(index);//1

**Array.isArray()**

判断是否为一个数组

**ES6函数扩展**

//1.普通写法

function fn(num) {

return num \* 10;

}

console.log(fn(10));//100

//2.匿名函数

var fn = function (num) {

return num \* 10;

};

console.log(fn(10));//100

//3.拉姆达表达式 语法糖 箭头函数

var fn = (num) => {

return num \* 10;

};

console.log(fn(10));//100

//4.箭头函数

var fn = (num) => num \* 10;

console.log(fn(10));//100

**箭头函数中的this指向**

<script>

var name = "蔡依林";

var obj = {

name: "刘德华",

showA: function () {

// showA是obj的方法 用的是普通函数的写法 this指向obj

console.log(this.name); //刘德华

var fn = () => {

//this指向箭头函数外的环境变量  还是obj

console.log(this.name); //刘德华

};

fn();

},

showB: () => {

//showB也是obj的方法 但是用的是箭头函数的写法 this指向window

console.log(this.name); //蔡依林

var fn = () => {

//this指向箭头函数外的环境变量 还是window

console.log(this.name); //蔡依林

};

fn();

},

};

obj.showA();

obj.showB();

</script>

**形参的默认值**

function sayHello2(name='qianfeng'){

document.write(`Hello ${name}`); //注意引号

}

有形参以形参为准，没有形参则以默认值为准

**ES6对象扩展**

**属性的简写:**

var foo = 'bar';

var baz = {foo}; // 等同于 var baz = {foo: 'bar'};

**方法的简写:**

var o = {

method() {

return "Hello!";

}

};

**属性名表达式**

obj['a'+'bc'] = 123;

**方法名表达式**

var obj = {

["h" + "ello"]() {

return "hi";

},

};

console.log(obj.hello());//hi

**Object.is( , )**

用来比较两个值是否严格相等。它与==全等运算符不一样，不会强制转换数据类型。与严格比较运算符（===）的行为基本一致， 不同之处只有两个：一是+0不等于-0，二是NaN等于自身，以下情况返回true

都是 undefined

都是 null

都是 true 或 false

都是相同长度的字符串且相同字符按相同顺序排列

都是相同对象（意味着每个对象有同一个引用）

都是数字且

都是 +0

都是 -0

都是 NaN

或都是非零而且非 NaN 且为同一个值

**Object.assign()**

浅拷贝

用来将源对象（source）的所有可枚举属性，复制到目标对象（target）。它至少需要两个对象作为参数，第一个参数是目标对象，后面的参数都是源对象。只要有一个参数不是对象，就会抛出TypeError错误。如果目标对象有相同的属性，则会用源对象属性的值覆盖。假如源对象的属性是一个对象的引用，它仅仅会复制其引用值（地址）。

assign()会直接修改目标对象，这种合并的返回值也是目标对象的引用值

如果要拷贝一个新的 则可以这样使用

var obj2 = Object.assign({},obj2),

**ES6 解构赋值**

解构赋值是对赋值运算符的扩展。解构赋值主要分为数组的解构和对象的解构。

**对象的对称赋值**

var obj = {name: "刘德华",age: 20,child:{name:"A"}};

var { name, age } = obj;

console.log(name); //刘德华

console.log(age); //20

var {name,child:{age:age1}} = {name: "刘德华",child:{age:"A"}};

console.log(name); //刘德华

console.log(age1); //A

console.log(age); //报错

给对象对称赋值的时候 结构必须对称且，必须有相同的属性名在前

可以在属性名后面加：别名, 这样使用变量的时候就要用别名 原变量名使用会报错

**数组的对称赋值**

var arr = [11, 22, 33, 44];

// var a = arr[0];

// var b = arr[1];

// var c = arr[2];

var [a, b, c, d] = arr;

console.log(a, b, c, d);//11 22 33 44

**交换位置：如冒泡排序**

var [a, b] = [11, 22];

[b, a] = [a, b];

console.log(a, b);//22 11

**扩展运算符/展开运算符 (解构)**

...

**用于合并数组**

var arr1 = [1, 2, 3];

var arr2 = [4, 5, 6];

var arr3 = [...arr1, ...arr2];

console.log(arr3);// 1,2,3,4,5,6

**用于最大最小值**

var arr = [5, 4, 7, 5, 9, 7, 0, 4, 1, 0, 9, 6, 4, 3];

console.log(Math.max(...arr));//9

**找出最大值下标**

var arr = [5, 4, 7, 5, 9, 7, 0, 4, 1, 6, 4, 3];

console.log(arr.findIndex((item) => item == Math.max(...arr)));//4

**函数形参的解构赋值**

function fn({ name, age }) {

console.log(name);//刘德华

console.log(age);//100

}

fn({ name: "刘德华", age: 100 });

**ES6-Set 值类型**

是一种新的数据结构，它的特点是不能有重复项

var set = new Set([1,2,3,4,5,5,5,5]);

console.log(set.size); // 5

**Set的属性和方法:**

size : 数量

add(value)：添加某个值，返回Set结构本身

delete(value)：删除某个值，返回一个布尔值，表示删除是否成功

has(value)：返回一个布尔值，表示该值是否为Set的成员

clear()：清除所有成员，没有返回值

**ES6-WeakSet:**

WeakSet和Set一样都不存储重复的元素, 用法基本类似，

但有一些不同点, WeakSet的成员只能是对象，而不能是其他类型的值。

不能遍历 没有迭代器 Symbol.iterator

**WeakSet的属性和方法:**

add(value)：添加某个值，返回Set结构本身

delete(value)：删除某个值，返回一个布尔值，表示删除是否成功

has(value)：返回一个布尔值，表示该值是否为Set的成员

**ES6-Map:**

Map 是一个“超对象”，用于存储键值对，是一个数组类型

其 key 和值 可以是任意类型

var map = new Map([

[1, "one"],

[2, "two"],

]);

console.log(map);//Map(2) {1 => 'one', 2 => 'two'}

**Map的属性和方法**

size：返回成员总数。

console.log(map.size);//2

set(key, value)：设置一个键值对。

map.set(3, "three");

console.log(map);//Map(3) {1 => 'one', 2 => 'two', 3 => 'three'}

get(key)：读取一个键。

for (var [k,v] in map){

console.log(k)//键 key

console.log(v)//值 value

}

has(key)：返回一个布尔值，表示某个键是否在Map数据结构中。

console.log(map.has(4));//false

delete(key)：删除某个键。

console.log(map.delete(1));//true

console.log(map);//Map(2) {2 => 'two', 3 => 'three'}

clear()：清除所有成员。

map.clear();

console.log(map);//Map(0) {size: 0}

keys()：返回键名的遍历器。

console.log(map.keys());//MapIterator {2, 3}

values()：返回键值的遍历器。

console.log(map.values());//MapIterator {'two', 'three'}

entries()：返回所有成员的遍历器。

console.log(map.entries());//MapIterator {2 => 'two', 3 => 'three'}

**for in**

for of每次循环取出每一个元素的key(属性名)/或者是数组的下标

更加适合遍历对象

var o = {name: "刘德华",age: 20,};

var a = ["asd", "fgh", "jkl"];

for (var key in o) {

console.log(key);//name

console.log(o[key]);//刘德华

}

for (var key in a) {

console.log(key);//0

console.log(a[key]);//asd

}

**for of**

是ES6新引入的特性，修复了ES5引入的for...in的不足

for...of不能循环普通的对象，需要通过Object.keys()或者.values() .entries()搭配使用

for...of更加强大，能遍历Array、Map、Set、String、arguments、TypeArray

for of每次循环取出每一个元素/属性的值 value

更加适合遍历数组 因为遍历对象的时候会提示没有迭代器 iterator

**Object.entries() 键和值**

for(var [k,v] of Object.entries(o)){

console.log(k,v)//k是对象的属性名，v是对象的值

}

**手动添加迭代器**

var o = {

name:"刘德华",

age:20,

//手动添加迭代器方法

[Symbol.iterator](){

//利用Object.keys()或者.values() 把对象的属性或者值转换为数组

let list = Object.key(this);

let size = list.length;

var index = -1;//因为下面有个index++ 所以初始值要是-1

//方法里返回一个对象

return {

//对象里必须有next方法

next(){

//方法里返回一个对象

index++;

return{

//对象里必须要有value和done

value:list[index],

done:index == true

}

}

}

}

}

for(var key of o){

console.log(key,o[key])

}

已知属性名 用.age更好 可读性好

未知属性名 属性名在变 用[age]

**Symbol 唯一标识符**

Symbol 是ES6引入的一种新的数据类型，来表示独一无二的值

//创建Symbol值 ，不能添加new

var s1=Symbol();

var s2=Symbol();

console.log(s1==s2);//false

Symbol.for() 如果是第一次执行就创建一个Symbol类型

如果是第二次执行，就取上一次的Symbol类型的值

Symbol.for(a)

**ES6-Class:**

ES6提供了更接近传统语言的写法，引入了Class（类）这个概念，作为对象的模板。通过class关键字，可以定义类。

class Point {

constructor(x, y) {

this.x = x;

this.y = y;

}

toString() {

return '('+this.x+', '+this.y+')';

}

}

注意： constructor方法是类的默认方法，通过new命令生成对象实例时，自动调用该方法。

**静态方法**

在类的方法前加static

用的时候不需要实例化（new）一个具体的实例

静态方法的this指向当前的类

**Class的继承**

Class之间可以通过extends关键字，实现继承

class Point {

constructor(x, y) {

this.x = x;

this.y = y;

}

}

class ColorPoint extends Point {

constructor(x, y, color) {

super(x, y);

this.color = color;

}

getData(){

console.log(this.x+','+this.y+','+this.color);

return this.x+','+this.y+','+this.color;

}

}

**工具类**

