**ES6、ES7、ES8、ES9、ES10新特性**

ES全称ECMAScript，ECMAScript是ECMA制定的标准化脚本语言

[**ES6、ES7、ES8、ES9、ES10新特性** 1](#_Toc6086878)

[ES6新特性（2015） 2](#_Toc6086879)

[1. 类（class） 3](#_Toc6086880)

[2.模块化(Module) 4](#_Toc6086881)

[3.箭头（Arrow）函数 5](#_Toc6086882)

[4. 函数参数默认值 5](#_Toc6086883)

[5.模板字符串 6](#_Toc6086884)

[6. 解构赋值 6](#_Toc6086885)

[7.延展操作符(Spread operator) 8](#_Toc6086886)

[8.对象属性简写 9](#_Toc6086887)

[9. Promise 10](#_Toc6086888)

[10.支持let与const 10](#_Toc6086889)

[ES7新特性（2016） 11](#_Toc6086890)

[1.Array.prototype.includes() 11](#_Toc6086891)

[2.指数操作符 12](#_Toc6086892)

[ES8新特性（2017） 12](#_Toc6086893)

[1. async/await 13](#_Toc6086894)

[2.Object.values() 13](#_Toc6086895)

[3.Object.entries() 13](#_Toc6086896)

[4.函数参数列表结尾允许逗号 14](#_Toc6086897)

[5.Object.getOwnPropertyDescriptors() 14](#_Toc6086898)

[6.SharedArrayBuffer对象 15](#_Toc6086899)

[7.Atomics对象 15](#_Toc6086900)

[ES9新特性（2018） 17](#_Toc6086901)

[1.异步迭代 17](#_Toc6086902)

[2.Promise.finally() 18](#_Toc6086903)

[3.Rest/Spread 属性 18](#_Toc6086904)

[4.正则表达式命名捕获组 19](#_Toc6086905)

[5.正则表达式反向断言 20](#_Toc6086906)

[6.正则表达式dotAll模式 21](#_Toc6086907)

[7.正则表达式 Unicode 转义 21](#_Toc6086908)

[8.非转义序列的模板字符串 21](#_Toc6086909)

[ES10新特性（2019） 21](#_Toc6086910)

[1. 新增了String的trimStart()方法和trimEnd()方法 22](#_Toc6086911)

[2.Function.prototype.toString()返回精确字符，包括空格和注释 22](#_Toc6086912)

[3.新的基本数据类型BigInt 22](#_Toc6086913)

## ES6新特性（2015）

ES6的特性比较多，在ES5发布近6年（2009-11至2015-6）之后才将其标准化。在这里列举几个常用的：

* 类
* 模块化
* 箭头函数
* 函数参数默认值
* 模板字符串
* 解构赋值
* 延展操作符
* 对象属性简写
* Promise
* Let与Const

### 类（class）

类让JavaScript的面向对象编程变得更加简单和易于理解

//一.ES5写法：

function Animate(name){

this.name = name;

}

Animate.prototype.getname = function(){

console.log(this.name)

}

var p =new Animate("lity");

p.getname();//lity

//二.ES6,面向对象的写法,calss,

class Person{

//constructor()：构造方法是默认方法，new的时候回自动调用，如果没有显式定义，会自动添加

//1.适合做初始化数据

//2.constructor可以指定返回的对象

constructor(name,age){

　　 this.name = name;

　　 this.age = age;

　　}

getval(){

console.log(`你是${this.name},${this.age}岁`);

}

}

var c1 = new Person("lity",20);

c1.getval();//你是lity,20岁

### 2.模块化(Module)

ES5不支持原生的模块化，在ES6中模块作为重要的组成部分被添加进来。模块的功能主要由export和import组成。每一个模块都有自己单独的作用域，模块之间的相互调用关系是通过export来规定模块对外暴露的接口，通过import来引用其他模块提供的接口。同事还为模块创造了命名空间，防止函数的命名冲突

#### 导出(export)

ES6允许在一个模块中使用export来导出多个变量或函数。

**导出变量**

//test.js

export var name = 'Rainbow'

心得：ES6不仅支持变量的导出，也支持常量的导出。

export const sqrt = Math.sqrt;//导出常量

ES6将一个文件视为一个模块，上面的模块通过export向外输出了一个变量。一个模块也可以同时往外面输出多个变量

//test.js

var name = 'Rainbow';

var age = '24';

export {name, age};

**导出函数**

// 导出函数

export function myModule(someArg) {

return someArg;

}

#### 导入(import)

定义好模块的输出以后就可以在另外一个模块通过import引用

import {myModule} from 'myModule';// main.js

import {name,age} from 'test';// test.js

心得:一条import 语句可以同时导入默认函数和其它变量。

import defaultMethod, { otherMethod } from 'xxx.js';

### 3.箭头（Arrow）函数

这是ES6中最有代表性的特性之一。=>不只是关键字function的简写，它还带来了其它好处，箭头函数与包围它的代码共享同一个this能帮助我们很好的解决this的指向问题。var self = this; 或var that = this是常用的引用外围this的模式。但借助=>，就不需要这种模式了

#### 箭头函数的结构

箭头函数的箭头=>之前是一个空括号、单个的参数名、或用括号括起的多个参数名，而箭头之后可以是一个表达式（作为函数的返回值），或者是用花括号阔气的函数体（需要自行通过return来返回值，否则返回的是undefined）

()=>1

v=>v+1

(a,b)=>a+b

()=>{

alert("foo");

}

e=>{

if (e == 0){

return 0;

}

return 1000/e;

}

心得：不论是箭头函数还是bind，每次被执行都返回的是一个新的函数引用

### 4. 函数参数默认值

ES6支持在定义函数的时候为其设置默认值

function foo(height = 50, color = 'red')

{

// ...

}

不使用默认值：

function foo(height, color)

{

var height = height || 50;

var color = color || 'red';

//...

}

这样写一般没问题，但当参数的布尔值为false时，就会有问题了。比如这样调用foo函数：

foo(0, "")

因为0的布尔值为false，这样height的取值将是50,。同理color的取值为‘red’。

所以说，函数参数默认值不仅能使代码变得简洁，还能规避一些问题

### 5.模板字符串

ES6支持模板字符串，使得字符串的拼接更加的简洁、直观

不使用模板字符串：

var name = 'Your name is ' + first + ' ' + last + '.'

使用模板字符串：

var name = `Your name is ${first} ${last}.`

在ES6中通过${}就可以完成字符串的拼接，只需要将变量放在大括号之中

### 6. 解构赋值

解构赋值语法是JavaScript的一种表达式，可以方便的从数组或者对象中快速提取值赋给定义的变量

#### 获取数组中的值

从数组中获取值并赋值到变量中，变量的顺序与数组中对象顺序对应

var foo = ["one", "two", "three", "four"];

var [one, two, three] = foo;

console.log(one); // "one"

console.log(two); // "two"

console.log(three); // "three"

//如果你要忽略某些值，你可以按照下面的写法获取你想要的值

var [first, , , last] = foo;

console.log(first); // "one"

console.log(last); // "four"

//你也可以这样写

var a, b; //先声明变量

[a, b] = [1, 2];

console.log(a); // 1

console.log(b); // 2

如果没有从数组中的获取到值，你可以为变量设置一个默认值

var a, b;

[a=5, b=7] = [1];

console.log(a); // 1

console.log(b); // 7

通过解构赋值可以方便的交换两个变量的值

var a = 1;

var b = 3;

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

console.log(a); // 3

console.log(b); // 1

获取对象中的值

const student = {

name: 'Ming',

age: '18',

city: 'Chengdu'

};

const { name, age, city } = student;

console.log(name); // "Ming"

console.log(age); // "18"

console.log(city); // "Chengdu"

### 7.延展操作符(Spread operator)

延展操作符... 可以在函数调用/数组构造时，将数组表达式或者string在语法层面展开；还可以在构造对象时，将对象表达式按key-value的方式展开

#### 语法

函数调用：

myFunction(...iterableObj);

数组构造或字符串：

[...iterableObj, '4', ...'hello', 6];

#### 应用场景

在函数调用时使用延展操作符

function sum(x, y, z) {

return x + y + z;

}

const numbers = [1, 2, 3];

//不使用延展操作符

console.log(sum.apply(null, numbers));// apply:方法能劫持另外一个对象的方法，继承另外一个对象的属性.

//使用延展操作符

console.log(sum(...numbers));// 6

构造数组

没有展开语法的时候，只能组合使用push，splice，concat等方法，来将已有数组元素变成新数组的一部分。有了展开语法，构造新数组会变得更简单：

const stuendts = ['Jine','Tom'];

const persons = ['Tony',... stuendts,'Aaron','Anna'];

conslog.log(persions)// ["Tony", "Jine", "Tom", "Aaron", "Anna"]

和参数列表的展开类似，... 在构造字数组时，可以任意位置多次使用。

数组拷贝

var arr = [1, 2, 3];

var arr2 = [...arr]; // 等同于 arr.slice()

arr2.push(4);

console.log(arr2)//[1, 2, 3, 4]

展开语法和Object.assign()行为一致，执行的都是浅拷贝（只遍历一层）

连接多个数组

var arr1 = [0, 1, 2];

var arr2 = [3, 4, 5];

var arr3 = [...arr1, ...arr2];// 将 arr2 中所有元素附加到 arr1 后面并返回

//等同于

var arr4 = arr1.concat(arr2);

### 8.对象属性简写

在ES6中允许我们在设置一个对象的属性的时候不指定属性名

不使用ES6

const name='Ming',age='18',city='Chengdu';

const student = {

name:name,

age:age,

city:city

};

console.log(student);//{name: "Ming", age: "18", city: "Chengdu"}

对象中必须包含属性和值，想的非常冗余

使用ES6

const name='Ming',age='18',city='Chengdu';

const student = {

name,

age,

city

};

console.log(student);//{name: "Ming", age: "18", city: "Chengdu"}

对象直接写变量，简洁

### 9. Promise

Promise是异步编程的一种解决方案

不使用ES6

嵌套两个setTimeout回调函数

setTimeout(function()

{

console.log('Hello'); // 1秒后输出"Hello"

setTimeout(function()

{

console.log('Hi'); // 2秒后输出"Hi"

}, 1000);

}, 1000);

使用ES6

var waitSecond = new Promise(function (resolve, reject) {

setTimeout(resolve, 1000);

});

waitSecond

.then(function () {

console.log("Hello"); // 1秒后输出"Hello"

return waitSecond;

})

.then(function () {

console.log("Hi"); // 2秒后输出"Hi"

});

上面的代码使用两个then来进行异步编程串行化，避免了回调地狱

### 10.支持let与const

在之前JS是没有块级作用域的，const与let填补了这方面的空白，const与let都是块级作用域

使用var定义的变量为函数级作用域：

{

var a = 10;

}

console.log(a); // 输出10

使用let与const定义的变量为块级作用域：

{

let a = 10;

}

console.log(a); //-1 or Error“ReferenceError: a is not defined”

## ES7新特性（2016）

ES2016添加了两个小的特性：

* 数组includes()方法，用来判断一个数组是否包含一个指定的值，根据情况，如果包含则返回true，否则返回false
* a\*\*b指数运算符，它与Math.pow(a,b)相同

### 1.Array.prototype.includes()

includes()函数用来判断一个数组是否包含一个指定的值，如果包含则返回true，否则返回false

includes函数与indexOf函数很相似，如下面两个表达式是等价的：

arr.includes(x)

arr.indexOf(x) >= 0

判断数组中是否包含某个元素：

在ES7之前的做法

使用indexOf()验证数组中是否存在某个元素，这时需要根据返回值是否为-1来判断：

if (arr.indexOf('react') !== -1){

console.log('react存在');

}

使用ES7的includes()

使用includes()验证数组中是否存在某个元素，更加直观簡單：

let arr = ['react', 'angular', 'vue'];

if (arr.includes('react')){

console.log('react存在');

}

### 2.指数操作符

在ES7中引入了指数运算符\*\*，\*\*具有与Math.pow(..)等效的计算结果

不使用指数操作符

使用自定义的递归函数calculateExponent或者Math.pow()进行指数运算：

function calculateExponent(base, exponent){

if (exponent === 1){

return base;

}else{

return base \* calculateExponent(base, exponent - 1);

}

}

console.log(calculateExponent(2, 10)); // 输出1024

console.log(Math.pow(2, 10)); // 输出1024

使用指数操作符

使用指数运算符\*\*，就像+、-等操作符一样使用：

console.log(2\*\*10);// 输出1024

## ES8新特性（2017）

* async/await
* Object.values()
* Object.entries()
* 函数参数列表结尾允许逗号
* Object.getOwnPropertyDescriptors()
* ShareArrayBuffer和Atomics对象，用于从共享内存位置读取和写入

### async/await

await可以和for...of循环一起使用，以串行的方式异步操作。例如：

async function process(array) {

for await (let i of array) {

doSomething(i);

}

}

### 2.Object.values()

Object.values()是一个与Object.keys()类似的新函数，但返回的是Object自身属性的所有值，不包括继承的值。

假设我们要遍历如下对象obj的所有值：

const obj = {a: 1, b: 2, c: 3};

不使用Object.values() :ES7

const vals=Object.keys(obj).map(key=>obj[key]);

console.log(vals);//[1, 2, 3]

使用Object.values() :ES8

const values=Object.values(obj);

console.log(values);//[1, 2, 3]

可以看出Object.values()为我们省去了遍历key，并根据这些key获取value的步骤

### 3.Object.entries()

Object.entries()函数返回一个给定对象自身可枚举属性的键值对的数组

遍历上面的obj对象的所有属性的key和value：

不使用Object.entries() :ES7

Object.keys(obj).forEach(key=>{

    console.log('key:'+key+' value:'+obj[key]);

})

//key:a value:1

//key:b value:2

//key:c value:3

使用Object.entries() :ES8

for(let [key,value] of Object.entries(obj1)){

    console.log(`key: ${key} value:${value}`)

}

//key:a value:1

//key:b value:2

//key:c value:3

### 4.函数参数列表结尾允许逗号

主要作用是方便使用git进行多人协作开发时修改同一个函数减少不必要的行变更

### 5.Object.getOwnPropertyDescriptors()

Object.getOwnPropertyDescriptors()函数用来获取一个对象的所有自身属性的描述符，如果没有任何自身属性，则返回空对象

函数原型：

Object.getOwnPropertyDescriptors(obj)

返回obj对象的所有自身属性的描述符，如果没有任何自身属性，则返回空对象

const obj2 = {

    name: 'Jine',

    get age() { return '18' }

};

Object.getOwnPropertyDescriptors(obj2)

// {

// age: {

// configurable: true,

// enumerable: true,

// get: function age(){}, //the getter function

// set: undefined

// },

// name: {

// configurable: true,

// enumerable: true,

//      value:"Jine",

//      writable:true

// }

// }

### 6.SharedArrayBuffer对象

SharedArrayBuffer对象用来表示一个通用的，固定长度的原始二进制数据缓冲区，类似于ArrayBuffer对象，它们都可以用来在共享内存（shared memory）上创建视图。与ArrayBuffer不同的是，SharedArrayBuffer不能被分离

/\*\*

\*

\* @param {\*} length 所创建的数组缓冲区的大小，以字节(byte)为单位。

\* @returns {SharedArrayBuffer} 一个大小指定的新 SharedArrayBuffer 对象。其内容被初始化为 0。

\*/

new SharedArrayBuffer(length)

### 7.Atomics对象

Atomics对象提供了一组静态方法用来对SharedArrayBuffer对象进行原子操作。

这些原子操作属于Atomics模块。与一般的全局对象不同，Atomics不是构造函数，因此不能使用new操作符调用，也不能将其当做函数直接调用。Atomics的所有属性和方法都是静态的（与Math对象一样）。

多个共享内存的县城能够同时读写同一位置上的数据。原子操作会确保正在读或写的数据的值是符合预期的，即下一个原子操作一定会在上一个原子操作结束后才会开始，其操作过程不会中断。

* Atomics.add()

将指定位置上的数组元素与给定的值相加，并返回相加前该元素的值

* Atomics.and()

将制定位置上的数组元素与给定的值想与，并返回与操作前该元素的值

* Atomics.compareExchange()

如果数组中指定的元素与给定的值相等，则将其更新为新的值，并返回该元素原先的值

* Atomics.exchange()

将数组中指定的元素更新为给定的值，并返回该元素更新前的值

* Atomics.load()

返回数组中指定元素的值

* Atomics.or()

将指定位置上的数组元素与给定的值相或，并返回或操作前该元素的值

* Atomics.store()

将数组中指定的元素设置为给定的值，并返回该值

* Atomics.sub()

将指定位置上的数组元素与给定的值相减，并返回相减前该元素的值

* Atomics.xor()

将指定位置上数组元素与给定的值相异或，并返回异或操作前该元素的值

wait() 和 wake() 方法采用的是 Linux 上的 futexes 模型（fast user-space mutex，快速用户空间互斥量），可以让进程一直等待直到某个特定的条件为真，主要用于实现阻塞。

* Atomics.wait()

检测数组中某个指定位置上的值是否仍然是给定值，是则保持挂起直到被唤醒或超时。返回值为‘ok’、‘not-equal’或‘time-out’。调用时，如果当前线程不允许阻塞，则会抛出异常

* Atomics.wake()

唤醒等待队列中正在数组指定位置上等待的线程。返回值为成功唤醒的线程数量

* Atomics.isLockFree(size)

可以用来检测当前系统是否支持硬件级的原子操作。对于指定大小的数组，如果当前系统支持硬件级的原子操作，则返回true；否则就意味着对于该数组，Atomics对象中的各原子操作都只能用锁来实现。

## ES9新特性（2018）

* 异步迭代
* Promise.finally()
* Rest/Spread属性
* 正则表达式命名捕获组
* 正则表达式反向断言
* 正则表达式dotAll模式
* 正则表达式Unicode转义
* 非转义序列的模板字符串

### 1.异步迭代

在async/await的某些时刻，你可能尝试在同步循环中调用异步函数。例如：

async function process(array) {

for (let i of array) {

await doSomething(i);

}

}

这段代码不会正常运行，下面这段同样也不会：

async function process(array) {

array.forEach(async i => {

await doSomething(i);

});

}

这段代码中，循环本身依旧保持同步，并在内部异步函数之前全部调用完成。

ES2018引入异步迭代器（asynchronous iterators），这就像常规迭代器，除了next()方法返回一个Promise。因此await可以和for...of循环一起使用，以串行的方式运行异步操作。例如：

async function process(array) {

for await (let i of array) {

doSomething(i);

}

}

### 2.Promise.finally()

一个Promise调用链要么成功到达最后一个.then()，要么失败触发.catch()。在某些情况下，你想要在无论Promise运行成功还是失败，运行相同的代码，例如清除、删除对话，关闭数据库连接等。

.finally()允许你指定最终的逻辑：

function doSomething() {

doSomething1()

.then(doSomething2)

.then(doSomething3)

.catch(err => {

console.log(err);

})

.finally(() => {

// finish here!

});

}

### 3.Rest/Spread 属性

ES2015引入了Rest参数和扩展运算符。三个点（…）仅用于数组。Rest参数语法允许我们将一个不定数量的参数表示为一个数组

restParam(1, 2, 3, 4, 5);

function restParam(p1, p2, ...p3) {

// p1 = 1

// p2 = 2

// p3 = [3, 4, 5]

}

展开操作符以相反的方式工作，将数组转换成可传递给函数的单独参数。例如Math.max()返回给定数字中的最大值：

const values = [99, 100, -1, 48, 16];

console.log( Math.max(...values) ); // 100

ES2018为对象解构提供了和数组一样的Rest参数（）和展开操作符，一个简单的例子：

const myObject = {

a: 1,

b: 2,

c: 3

};

const { a, ...x } = myObject;

// a = 1

// x = { b: 2, c: 3 }

或者你可以使用它给参数传递参数：

restParam({

a: 1,

b: 2,

c: 3

});

function restParam({ a, ...x }) {

// a = 1

// x = { b: 2, c: 3 }

}

跟数组一样，Rest参数只能在声明的结尾处使用。此外，它只是用于每个对象的顶层，如果对象中嵌套对象则无法适用。

扩展运算符可以在其他对象内使用，例如：

const obj1 = { a: 1, b: 2, c: 3 };

const obj2 = { ...obj1, z: 26 };

// obj2 is { a: 1, b: 2, c: 3, z: 26 }

可以使用扩展运算符拷贝一个对象，像是这样obj2 = {...obj1}，但是**这只是一个对象的浅拷贝**。另外，如果一个对象A的属性是对象B，那么在克隆后的对象cloneB中，该属性指向对象B

### 4.正则表达式命名捕获组

JavaScript正则表达式可以返回一个匹配的对象——一个包含匹配字符串的类数组，例如：以YYYY-MM-DD的格式解析日期：

const

reDate = /([0-9]{4})-([0-9]{2})-([0-9]{2})/,

match = reDate.exec('2018-04-30'),

year = match[1], // 2018

month = match[2], // 04

day = match[3]; // 30

这样的代码很难读懂，并且改变正则表达式的结构有可能改变匹配对象的索引

ES2018允许命名捕获组使用符号?<name>，在打开捕获括号(后立即命名，示例如下：

const

reDate = /(?<year>[0-9]{4})-(?<month>[0-9]{2})-(?<day>[0-9]{2})/,

match = reDate.exec('2018-04-30'),

year = match.groups.year, // 2018

month = match.groups.month, // 04

day = match.groups.day; // 30

任何匹配失败的命名组都将返回undefined

命名捕获也可以使用在replace()方法中。例如将日期转换为美国的MM-DD-YYYY格式：

const

reDate = /(?<year>[0-9]{4})-(?<month>[0-9]{2})-(?<day>[0-9]{2})/,

d = '2018-04-30',

usDate = d.replace(reDate, '$<month>-$<day>-$<year>');

### 5.正则表达式反向断言

目前JavaScript在正则表达式中支持先行断言（lookahead）。这意味着匹配会发生，但不会有任何捕获，并且断言没有包含在整个匹配字段中。例如从价格中捕获货币符号：

const

reLookahead = /\D(?=\d+)/,

match = reLookahead.exec('$123.89');

console.log(match[0]); // $

ES2018引入以相同方式工作但是匹配前面的反向断言（lookbehind），这样我就可以忽略货币符号，单穿的捕获价格的数字：

const

reLookbehind = /(?<=\D)\d+/,

match = reLookbehind.exec('$123.89');

console.log(match[0]); // 123.89

以上是**肯定反向断言**，非数字\D必须存在。同样的，还存在**否定反向断言**，表示一个值必须不存在，例如：

const

reLookbehindNeg = /(?<!\D)\d+/,

match = reLookbehind.exec('$123.89');

console.log(match[0]); // null

### 6.正则表达式dotAll模式

正则表达式中点.匹配除回车外的任何单字符，标记s改变这种行为，袁旭终止符的出现，例如：

/hello.world/.test('hello\nworld'); // false

/hello.world/s.test('hello\nworld'); // true

### 7.正则表达式 Unicode 转义

到目前为止，在正则表达式中本地访问Unicode字符属性是不被允许的。ES2018添加了Unicode属性转义——形式为\p{...}和\P{...}，在正则表达式中使用标记u（unicode）设置，在\p快儿内，可以以键值对的方式设置需要匹配的属性而非具体内容。例如：

const reGreekSymbol = /\p{Script=Greek}/u;

reGreekSymbol.test('π'); // true

此特性可以避免使用特定Unicode区间来进行内容类型判断，提升可读性和可维护性

### 8.非转义序列的模板字符串

之前， \u开始一个unicode转义， \x开始一个十六进制转义， \后跟一个数字开始一个八进制转义。这使得创建特定的字符串变得不可能，例如Windows文件路径C:\uuu\xxx\111

## ES10新特性（2019）

* 新增了String的trimStart()和trimEnd()方法
* Function.prototype.toString()返回精确字符，包括空格和注释
* 新的基本数据类型BigInt

### 新增了String的trimStart()方法和trimEnd()方法

新增的这两个方法很好理解，分别去除字符串首尾空白字符。

### 2.Function.prototype.toString()返回精确字符，包括空格和注释

function /\* comment \*/ foo /\* another comment \*/() { }

// 之前不会打印注释部分

console.log(foo.toString()); // function foo(){}

// ES2019 会把注释一同打印

console.log(foo.toString()); // function /\* comment \*/ foo /\* another comment \*/ (){}

// 箭头函数

const bar /\* comment \*/ = /\* another comment \*/ () => { };

console.log(bar.toString()); // () => {}

### 3.新的基本数据类型BigInt

现在的基本数据类型（值类型）不止5中（ES6之后是6种），加上BigInt一共有7种数据类型，分别是：String、Number、Boolean、Null、Undefined、Symbol、BigInt