1. 声明变量
2. let声明变量
3. const声明常量

(1)声明的常量不允许重新赋值

(2)声明的常量必须初始化(必须有值) -->const a;(报错)

1. 两者的共同点

(1)不存在预解析(不存在变量, 函数提升, 需要先声明后使用)

(2)代码块中不得重复声明变量

(3)存在暂时性死区(声明的变量, 常量之前不能使用)

(4)只在它自己所在的代码块有效 -->{这里面有效}

4>实例 -->

(1)const a = [];

a.push('Hello'); // 可执行

a.length = 0; // 可执行

a = ['Dave']; // 报错

说明：常量a是一个数组，这个数组本身是可写的，但是如果将另一个数组赋值给a，就会报错。如果真的想将对象冻结，应该使用Object.freeze方法

(2)let声明的for循环循环外具体访问循环数返回当前循环数, 而var声明的返回最后一个循环变量 -->

var arr = [];

for (let(var) i = 0; i < 10; i +) { a[i] = function () {console.log(i)} }

a[6](); let声明的i打印出6, var声明的i打印出10

1. 解构赋值(适用于var, let, const)

1>数组的解构赋值

(1)基本用法(可以从数组中提取值, 按照对应位置, 对变量赋值)

var [a, b, c] = [1, 2, 3]

相当于var a = 1, b = 2, c = 3;

(2)默认值

var [a = 默认值, b, c] = [, 4,];

相对于var a = 默认值, b = 4, c = undefined;

(3)总结

##对应关系一一对应(左边索引对应右边索引)

##左边对应的右边的是数组时取右边第一位 -->

var [a, b, c] = [1, [2, 4], 3] 相当于var a = 1, b = 2, c = 3

2>对象解构赋值

(1)基本用法(按照对应属性名, 对属性名赋值)

let{foo, bar} = {foo: “hello”, bar: “hi”}

(2)别名(如果有了别名, 原来的名字就无效了)

let{foo: abc, bar} = {bar: “hi”, foo: “hello”};

打印foo报错, 打印abc值为hello

(3)简写 let{foo, bar} = Math;

(4)默认值

let {foo = “默认值”, bar} = {bar: “hi”} foo打印出默认值, bar打印出hi

3>字符串解构赋值

(1)基本结构(按照对应位置, 对变量赋值)

let= [a, b, c, d, e, length] = “hello”

a = h, b = e, c = l, d = l, e = o, length = undefined

(2)取字符串长度需借助对象赋值 -->

let = {length} = “hello” 打印出length为5

注意：(1)和(2)不可以写在一起, 写在一起会报错, 因为length重新声明了

自己的理解：数组和对象赋值的集合

解构赋值总结：

##有默认值和赋值的取赋值的值

1. 字符串扩展

1>includes(“匹配字符”, [从第几位开始匹配])方法

判断字符串是否包含指定字符串(有返回true, 无返回false);

2>startWith(“特定字符”);

判断字符串是否以特定的字符串开始

3>endWidth(“特定字符”);

判断字符串是否以特定的字符串结束

3>模板字符串

(1)基本格式(` 添加的内容 `)

(2)嵌入变量( ${变量} )

(3)${}里面除了变量还支持简单的运算

(3)模板字符默认都会保留代码的空格如果不需要可以使用trim()方法清除

1. 函数扩张
2. 参数默认值(es5使用 || 设置默认参数)

(1)基本格式

function foo(par = “默认值”) {} foo();

1. 参数结构赋值

(1)基本格式

function foo({username, age}) {} foo({})

(2)简写

function foo({username, age} = {}) {} foo()

1. rest参数(剩余参数)

(1)基本格式

function foo(a, b, ...param) { console.log(param) } foo(1, 2, 3, 4)

打印出一个数组[3, 4]

1. 扩展运算符

(1)function (a, b, c) foo{ console.log(a + b + c) }

let arr = [1, 2, 3] foo(...arr); 打印出6(把数组arr展开分别(成三变量)传进去)

(2)es5用法 foo.apply(null, arr);这样传递

(3)扩展运算符合并数组

let arr1 = [1, 2, 3]; let arr2 = [4, 5, 6]

合并arr1, arr2 -->arr3 = [...arr1, ...arr2];

1. 箭头函数

1>基本使用

let foo = () => console.log(“hello”); foo();

等价于function foo() { console.log(“hello”) }; foo();

2>传递参数

let foo = v => v; foo(111);

等价于function foo(v) { return v; } foo(111);

注意：多个参数括号必须()包裹, 函数执行多行代码必须用{}包裹代码 -->

let foo = (a, b) => { console.log(a); console,log(b) } foo(1, 2)

3>匿名函数

(a) => {} 相当于 function (a) {}

4>注意事项:

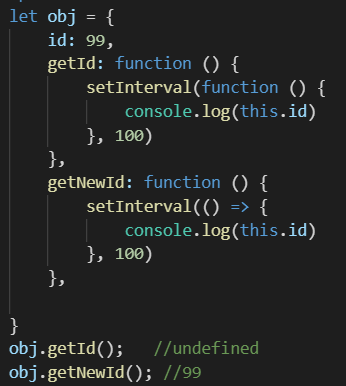
(1)箭头函数this取决于函数定义, 而不是调用

(2)箭头函数不也可new

(3)箭头函数不可以使用arguments获取参数列表, 可以使用rest参数代替...

5>箭头函数this指向(当箭头函数不存在) -->

setInterval里面的函数执行相当于window调用函数



1. Promise异步
2. 概念

Promise 是异步编程的一种解决方案，其实是一个构造函数，自己身上有all、reject、resolve这几个方法，原型上有then、catch等方法。会等自己下面的东西一起执行

2>基本实例



3>

1. Proxy代理

1>概念

Proxy用于修改某些操作的默认行为，等同与在语言层面做出修改，即对编程语言进行编程。

Proxy 可以理解成，在目标对象之前架设一层“拦截”，外界对该对象的访问，都必须先通过这层拦截，因此提供了一种机制，可以对外界的访问进行过滤和改写。Proxy 这个词的原意是代理，用在这里表示由它来“代理”某些操作，可以译为“代理器”。

它实现的就是“方法无法找到时”的行为，Proxy 对象用于定义基本操作的自定义行为(如属性查找，赋值，枚举，函数调用等)。

2>实例

