1. **定义变量**

let用法

1. let定义变量没有变量的提升（没有预解释）

Console.log(a);// a is not defined

Let a=1;

Console.log(a);//1

Function fn(){

Console.log(b);//b is not defined

Let b=0;

}

Fn();

2.let定义的变量不能重复定义(不管函数还是let定义的变量都不 可以重复)

Let b=12;

Let b=13;//Idenifier ‘b’ has already been declared

Function b(){//Idenifier ‘b’ has already been declared

}

3.虽然不进行预解释，但是代码执行前上来也是先将定义的变量提前过滤一遍，一旦发现不合法的就直接报错了，代码也不会执行了。

Console.log(A);//未定义

Let A=10;

Let A=20;//先报错 ‘A’ has already been declared

块级作用域

{ },是一个私有作用域，块级作用域

for(){ }, if(){ }, { }

Let a=10;

If(a){//私有作用域

Console.log(a);// a is not defined

Let a=1;

Console.log(a);//1

}

var oLis=list.getElementsByTagName("li");

console.log(oLis);

for(let i=0;i<oLis.length;i++){//如果变成var则弹出的最后都是4

oLis[i].onclick=function () {//绑定事件是异步的

alert(i);//0,1,2,3

}

}

Let a=10;

Function fn(){//自己没有a,还是会往上找的

Console.log(a);//10

}

自执行函数简便写法

{

Let a=100;

Console.log(a);//100

}

eval( )将字符串转为json对象

Var user=‘{

Name:”张三”,

Age:12,

Sex:”男”

}’

Var json=eval(‘(’+user+’)’);

const用法

const定义的是静态变量，不能修改值，必须要赋值，不可以重复声明（不管是动态变量let还是静态变量const都不可以重复声明）

const a;//不赋值就报错：Missing initializer in const declaration

const a=20;

a=30;//会报错，不能被修改

//会报错，因为不可以重复声明

Let aa=2;

Const aa=3;

1. **数组赋值**

**解构赋值**

let ary=["a","b"];

let [a,b,c]=ary

console.log(a,b,c);//a,b,undefined

let ary2=['aa','bb','cc','dd'];

let [a,b,c]=ary2;

console.log(a,b,c);//aa,bb,cc

嵌套赋值

let [x,y,[s],[[f]]]=[1,2,[3],[[4]]];

console.log(x,y,s,f);//1,2,3,4

省略赋值

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

let a=ary[0];

let b=ary[ary.length-1];

//x是ary的第1项，y是ary的第5项

let [x,,,,y]=ary;

console.log(x,y);//1,5

不定参数赋值

let [a,...b]=[1,2,3,4,5];

console.log(a,b);//1,[2,3,4,5]

let [,,x,,...y]=[1,2,3,4,5,6,7,8,0];

console.log(x,y);//3,[5,6,7,8,0]

默认值

//默认值

let [f0,f1]=[1];

console.log(f0,f1);//1,undefined

let [f2,f3=2]=[1];

console.log(f2,f3);//1,2

//判断一个位置是否有值，严格按照===判断，如果数组中成员不严格等于undefined，默认值不会生效

let [f=1]=[undefined];

console.log(f);//1

let [f=1]=[2];

console.log(f);//2

let [f=1]=[null];

console.log(f);//null

function fn(){

console.log('哈哈');

}

let [x=fn(),y=0]=[1];//

console.log(x,y);//1,0

//只要变量所占在数组中的位置的值不是undefined，fn()根本不会执行

1. **对象赋值**
2. **字符串**

//此时是将字符串转为类似数据的一个对象

let [x,y,z]='123';

console.log(x,y,z);//1 2 3

let {length}='12345';

console.log(length);//5

字符串方法

//indexof 字符串有没有指定字符，如果有返回索引，没有返回-1

//includes() 字符串中有没有字符，有返回true，没有返回false;

// 第二个参数：开始查找的索引，默认是0

var str="asfdfgzgruh";

console.log(str.includes('g',8));

//startsWith 是不是以某个字符作为开头

//endsWith 。。。。。。。。。。。结尾

//第二个参数：开始查找的索引，默认是0

console.log(str.startsWith('d'));//false

console.log(str.endsWith('h'));//true

console.log(str.startsWith('d',3));//true

//repeat(n),将字符串重复n次返回,原字符串不变

console.log(str.repeat(2),str);

//小数 向下取整

console.log(str.repeat(2.6));//取2次

//负数/infinity(无限值)--->报错

console.log(str.repeat(-2));

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

//0---->-1之间 先进行向下取整，再结果为0

console.log(str.repeat(-0.3));//空字符串 ''

//NaN---> 结果为0

console.log(str.repeat(NaN));//0

//字符串 将字符串作为数字

console.log(str.repeat('2'));//循环2次

模版字符串 ``

let str='I LOVE YOU';

document.body.innerHTML='<h1>'+str+'</h1>';//ES5

//``-----> ${}实现一个变量

let str1=`<h1>${str}</h1>`;//ES6

document.body.innerHTML+=str1;

//换行

console.log("hahahha\nhahhah");//ES5

//ES6 保留换行和空格

let s1=`hahahha

hertvmrgiur

nfrihr

`;

console.log(s1);

//\`转义

console.log(`ssss\`3456`);//ssss`3456

//{可以运算}

let x=1;

let y=2;

console.log(`${x}+${y}=${x+y}`);//1+2=3

//调用函数（带参）

function fn(x){

return x;

}

let s2='hhhh';

console.log(`函数fn输出的结果是${fn(s2)}`);//函数fn输出的结果是hhhh

let s3='return'+'`hello ${wd}`';

let fun=new Function("wd",s3);

console.log(fun('ZFPX'))；//hello ZFPX

1. **数组扩展**

**Array.from(),Array.of()**

**数组实例的copyWithin()**

**..........find()和findIndex()**

**..........fill()**

**..........entries(),keys()**

**..........includes()**

**数组的空位**

{

//一.Array类上的方法

//Array(),将传进来的值变成数组返回

//注意：只传一个参数n,代表返回数组的长度

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

console.log(Array(4));//length:4

//Array.of()

//注意和Array()只有一点区别：只传一个参数n时,返回的是当前参数组成的数组

console.log(Array.of(8,9));//[8,9]

//Array.from()

//参数：数组或者类数组

//返回值：新的数组

//通常：将类数组变成数组

let ary=[5,6,7,8];

let newAry=Array.from(ary);//克隆

console.log(newAry);//[5,6,7,8]

function toArray(){

return Array.from(arguments);

}

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

}

{

//二。Array原型上的方法

//1.copyWithin(target,start,end),原数组会被改变

//target:必填，从该位置开始替换数据

//start:选填，默认值为0，从该位置开始读取数组

//end：选填，默认值为数组长度，到该位置前停止读取数据

let ary=[1,2,3,4,5,6,7,8];

console.log(ary.copyWithin(0,1,5));//[2, 3, 4, 5, 5, 6, 7, 8]

console.log(ary);//替换后的新数组

// 5,2,6

// 1,2,3,4,5,3,4,5

//2.find findIndex

//find:先遍历数组一项一项的执行，一旦函数返回值为true，就停止查找,返回当前项（item），最后都没找到，结果为undefined

//findIndex：返回当前索引(index),若没有，则返回undefined

let ary2=[45,6,2,8,4,5,9,7]

console.log(ary2.find(function(item,index,input){

// return false;//undefined

return item==6;//6

}))

console.log(ary2.findIndex(function(item,index,input){

return item==6;//1

}))

console.log(ary2);

//3.fill(value,start,end)填充数组,原数组会被改变

//value:必填，填充的值

//start：选填，开始填充的位置，默认0

//end: 选填，停止。。。。。，不包括后面

let ary3=[1,2,3,4,5,6];

// console.log(ary3.fill(1));//[1,1,1,1,1]

ary3.fill(1,2,4);

console.log(ary3);//[1,2,1,1,5,6]

//得到有7个1的数组

//console.log(Array(7));//返回的是7个空位，不是undefined

console.log(Array(7).fill(1));

//4.includes()：数组中是否有某项 有:true, 没有：false

//indexof() lastIndexOf() 有:返回索引，没有:返回-1

console.log(ary3.includes(6));//true

//三.数组的空位 ：空位不是undefined

var arr=Array(3);//三个空位

var arr2=[undefined,undefined,undefined];

//用（索引） in arr 来检测是不是空位

console.log(1 in arr);//false

console.log(1 in arr2);//true

var arr3=[1,,,1,,];

console.log(arr3.length);//5

arr3.forEach(function(item){//ES5方法,forEach不支持return

console.log(item);//1 1

})

//ES5中的方法对空位处理都不太一致，大部分都是跳过空位

console.log(arr3.map(function(item){//ES5方法

return item;//[1, empty × 2, 1, empty]

}))

//ES6中的方法会将空位处理为undefined

arr3.find(function(item){

console.log(item);//[1, undefined,undefined, 1, undefined]

})

//四.遍历数组

//for of:遍历当前项

let array=[1,2,3,4,3,5];

for(let item of array){

console.log('当前项'+item);//当前项

}

for(let key in array){

console.log('索引'+key)

}

//keys:遍历索引

for(let key of array.keys()){

console.log(key);//0,1,2,3,4,5

}

//entries()

for (let [index,item] of array.entries()){

console.log(index,item);

}

}

1. **函数扩展**

**1.基本用法**

**2.与解构赋值默认值结合起来使用**

**3.参数默认值的位置**

**4.函数的length**

**5.参数作用域**

**6.参数集合**

**7.扩展运算符**

**8.name属性**

**9.箭头函数**

//一.函数参数问题

//函数的参数可以加默认值,只有不传实参的时候才会走默认值

function fn1(x='x',y){

console.log(x,y);//, 2

}

fn1(1,2);

function fn2({x=0,y=0}={}){

// console.log({x,y});//{x:0,y:0}

return {x,y};//{x: 1, y: 0}

}

console.log(fn2({x:1}));

function fn3({x,y}={x:0,y:0}){

return {x,y};

}

console.log(fn3({x:1}));//{x: 1, y: undefined}

//一般会把默认值放在后面

function fn4(x,y=1,z){

}

fn4(1,2,3);

//参数集合

function fn5(){

//实参的长度

console.log(arguments.length);

}

fn5(1,2,3,4,5)

function fn6(x,y=1,z){

}

fn6();

//没有默认值的情况下：形参的长度

//有默认值，默认值的参数的位置

console.log(fn6.length);//1

}

{

//二.函数作用域问题

let n=10,m=20;

function fn1(x=n,y=m){

let n=1,m=2;

console.log(x,y);//10 20

}

fn1();

let x=10;

function fn2(x=1,y=x){

// let x=20;// 'x' has already been declared

// 因为形参已经有了x,所以let x=20;不能重新被定义

console.log(x,y);//1 1

}

//形参所在的小括号是一个单独的作用域，遇到变量了先找自己的，没有再往上一级找

//

fn2();

}

{

//三.扩展运算符 ...

//可以将数组变成非数组，也可以将非数组变成数组

let ary=[1,2,3];

console.log(...ary);//1 2 3

//将字符串变为数组

let str='12345';

console.log([...str]);

//将类数组变成数组

function fn(){

console.log([...arguments].sort());

}

fn(2,1,4,6,8,5,3);

//应用：求最大值的方法 Math.max(1,2,3,4,43,2,12)

let ary1=[2,3,5,1,5,7,5,43,2,455];

console.log("Math.max("+ary1+")");//Math.max(2,3,5,1,5,7,5,43,2,455) 里面会默认转toString方法

console.log(eval("Math.max("+ary1+")"));//455

//applay

let ary2=Math.max.apply(null,ary1);

console.log(ary2);//455

//concat

let a1=[2,3,4],a2=[1,2,6,7];

console.log(a1.concat(a2));

console.log([...a1,...a2])

}

{

//四.函数的name属性

// function fn(){};

let fn=function(){};

console.log(fn.name);//fn

console.log((function(){}).name);//空 匿名函数

//bind ----->bound 函数名

let obj={};

var fn1=fn.bind(obj);

fn1();

console.log(fn1.name);//bound fn

//new Function------->anonymous 匿名

let f=new Function('n','return n');

console.log(f.name);//anonymous

}

{

//五.箭头函数

function fn(x){

return x;

}

//改为箭头函数：x=>x;

//一般来说:1.点前执行是谁就是谁 2.new 执行函数中执行window 3.点击事件中谁点击就是谁

//箭头函数中this问题

//箭头函数没有this指向，它里面的this是谁，看它的上一级作用域中的this(保证上一级作用域不是箭头函数，如果上一级是箭头函数，还要往上找，直到找到的不是箭头函数)

let obj={

fn:function(){

console.log(this);

let f=()=>{

console.log(this);//this==obj

};

f();

},

fn1:()=>{

console.log(this);

}

}

obj.fn();//this==obj

obj.fn1();//thid==window

document.getElementById('box').onclick=()=>{

console.log(this);//window

}

}

1. **对象扩展**

{

//一.属性的简洁表示

let a='aa',b='bb';

let obj={

a:'a',

[a]:'AA',//[a] 代表变量

[a+b]:'A+B',

fn(){},//简写

// fn:function(){

// }

}

console.log(obj.a);//a

console.log(obj['a']);//a

console.log(obj);//{a: "a", aa: "AA", aabb: "A+B"}

}

{

//二.Object类上的方法

//Object.is

//===

//NaN===NaN false

console.log(Object.is(NaN,NaN));//true

let ary=[1,2,NaN,NaN,1,2];

console.log(ary.includes(NaN));//true

let ary2=[];

ary.forEach((item)=>{

var f=true;

for(var i=0;i<ary2.length;i++){

if(Object.is(item,ary2[i])){

f=false;

}

}

if(f){

ary2.push(item);

}

});

console.log(ary2);//数组去重[1, 2, NaN]

//Object.assign(obj1,obj2) 将obj2的内容合并到obj1上，返回obj1

let obj1={a:'1'},obj2={b:1};

console.log(Object.assign(obj1,obj2));//{a: "1", b: 1}

//Object.getOwnPropertyDescriptor() 数组下面的属性可枚举可修改

let str='123';

console.log(Object.getOwnPropertyDescriptor(str,'length'));//{value: 3, writable: false, enumerable: false, configurable: false}

console.log(Object.getOwnPropertyDescriptor([1,2,3],'length'));//writable: true

//属性的遍历:

//1.for in

//2.Object.keys();返回值是数组，包括对象自己（私有属性）所有可枚举的属性

//3.Object.getOwnPropertyNames();返回值是数组，包括对象自己（私有属性）所有属性

//4.Object.getOwnPropertySymbols(),返回值是个数组,包括对象自身的所有的Symbols属性

}

1. **Set数据结构**

{

//一.Set

//1.使用构造函数方式创建Set实例

//2.参数是个数组，返回值是Set实例，是去重后的

let set1=new Set([1,2,3,NaN,NaN,1,2,3,4,5]);

console.log(set1);

//将Set实例变为数组

console.log([...set1]);//[1, 2, 3, NaN, 4, 5]

console.log(Array.from(set1));// [1, 2, 3, NaN, 4, 5]

//利用Set实例实现数组去重

function fn(ary){

return [...new Set(ary)];

}

console.log(fn([1,2,3,4,5,3,2,2,3,4]));//[1, 2, 3, 4, 5]

}

{

//二.Set原型上的方法

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

console.log(set);

//size 长度

//1.add() :增加，之前没有才可以加上(加在后面)，返回当前实例，可以实现链式写法

console.log(set.add(6).add(8));

//2.has() :判断有没有某一项，返回true/false

console.log(set.has(1));//true

//3.delete() :删除，返回true/false

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

console.log(set.delete(9));//false

//4.clear() :清空 没有返回值 undefined

console.log(set.clear());//undefined

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

let arr2=[2,3,4,5,7,8];

//并集

function bj(arr1,arr2){

return [...new Set([...arr1,...arr2])];

}

console.log(bj(arr1,arr2));//[1, 2, 3, 4, 5, 7, 8]

//交集

function jj(arr1,arr2){

return arr1.filter((item)=>{

return arr2.includes(item);

})

}

console.log(jj(arr1,arr2));//[2, 3, 4, 5]

//差集: 并集过滤掉交集

function cj(arr1,arr2){

return bj(arr1,arr2).filter((item)=>{

return !jj(arr1,arr2).includes(item);

})

}

console.log(cj(arr1,arr2));//[1,7,8]

}

1. **Map数据结构**

{

// 二.Map

//1.使用构造函数方式创建一个实例

//2.参数是数组，数组每一项都是一个数组，这个数组有两项（key,value）

//3.key可以是任意数据类型

var map1=new Map([[1,'a'],['a','A'],[{name:'珠峰'},'珠峰'],[/\d+/,'正则']]);

console.log(map1);

//Map原型上的方法：

//1.get(key)

console.log(map1.get('a'));//A

//2.set(key,value)

console.log(map1.set(2,'js'));

//3.delete,has,clear

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

//将数组变成Map

var map=new Map();

//1,[1]

//2,[1,2]

//3,[1,2,3]

ary.forEach((item,index)=>{

map.set(index+1,ary.slice(0,index+1))

})

console.log(map);

//4.forEach(),keys(),values(),entries();既能遍历value，又能遍历key

map.forEach((val,key,map){

//val：值 key:键 map:原Map实例

})

for(var key of map.keys()){

// key:键

}

for(var val of map.values()){

// val:值

}

for(var [key,val] of map.entries()){

//val:值,

//key:键

}

}

1. **Promise**
2. **Class**
3. **Module**
4. **迭代器**