一．Let:

1. let定义变量没有预解析（不存在变量提升）
2. let不能重复定义（指的是在同个{}中不能重复let定义）
3. for(){}循环中，for循环（let i=0）中i的定义是父级定义，{let i =1}花括号中的i定义是另一个定义。

二．Const:(特性与let一样)

1. const定义的变量不能修改
2. const定义常量时必须赋值，不能后面赋值，且不能修改该 赋值

三．解构赋值：

数组：

Let [a,b,c]=[1,12,3]

Json:

Let {name,age,job}={name:manting,age:18,job:aaa}

四．‘’字符串模板

优点：可以随意换行，${name}可以表示变量

五．关于字符串

1.字符串查找：

includes（） str.indexOf(要找的单词) 返回索引（位置），没找到返回-1

str.includes(要找的单词) 返回true，没找到false

2.字符串是否以谁开头：str.startsWith(检测的东西) 返回true，没找到false

3.字符串是否以谁结尾：str.endsWith(检测的东西，可用于检测文件后缀.png)返回true/false

4.重复字符串：str.repeat(3重复的次数)

5.字符串填充：往前填充：str.padStart(字符串总长度，填充的东西)

往后填充：Str.padEnd(字符串总长度，填充的东西)

Str. padStart (str.length+padStr.length，padStr)

六．函数

1.函数默认参数

Function show(a=”name”,b=”age”){}其中name，age就是当函数没有传参时的默认参数

2.函数的参数默认已经被定义，不能重复let,const声明

function show(a=18){

let a=15//错误，因为默认参数里面a已经被定义

console.log(a)

}

Show()

七．扩展运算符，reset运算符,剩余运算符

…展开数组[1,2,3,4,5] …[1,2,3,4,5] ----🡪 1,2,3,4,5

可以收缩数组a=1,2,3,4,5 …a----🡪 [1,2,3,4,5]

剩余参数：必须放到 最后

八．箭头函数

()=>{

语句

return

}

1. let show =(a,b)=>{return a+b}
2. this指向问题，当使用箭头函数时，this指向当前定义函数所在的对象，不再试运行时所在的函数对象
3. 箭头函数里面没有arguments,用’…’
4. 箭头函数不能当构造函数

九．数组

1.arr.fotEach( //没有返回值，代替for循环使用

function(var,index,arr){

console.log(this,val,index,arr)

},123

)

没有返回值，代替for循环使用

2.arr.map(循环的回调函数，this指向谁 ){}// 返回是一个新的数组（有返回值）

正常情况下，需要配合return，返回是一个新的数组

若是没有return,相当于使用forEach

重新整理数据结构：

[{title:’aa’}] 🡪[{t:’a’}]

Eg: let arr=[

{title:"aaaa",hot:true},

{title:"bbb",hot:false},

{title:"ccc",hot:false},

{title:"ddd",hot:true}

]

let newarr=arr.map((item)=>{

let jisons=[]

jisons.tts=item.title

jisons.hots=item.hot

return jisons

})

console.log(newarr)

1. arr.filter(循环的回调函数，this指向谁){} //过滤效果，返回一个数组（有返回值）

//回调函数里面返回结果为true的时候，该item就留下

过滤一些不合格的“元素”，返回值如果为true就留下

Eh: let arr=[

{title:"aaaa",hot:true},

{title:"bbb",hot:false},

{title:"ccc",hot:false},

{title:"ddd",hot:true}

]

let newarr=arr.filter((item)=>{

return item.hot==true

})

console.log(newarr)

1. arr.some(循环的回调函数，this指向谁){}//返回true和false

类似查找，数值里面某一个元素符合条件，返回true

let arr=["apple","banana","orange"]

function findarry(arr,item){

return arr.some((val,index,arr)=>{

return val== item

})

}

console.log(findarry(arr,"banana"))//查找这个arr数组里面是否有banana,有就返回true

1. arr.every(循环的回调函数，this指向谁){} //返回true和false

数值里面所有元素都符合条件，才会返回true

let arr1=[1,3,5,7,9,10]

let arr2=[1,3,5,7,9]

function aeeevery(arr){

return arr.every((item,index,arr)=>{//every要循环每一项，必须所有项回调函数的返回值都为true才会返回true

return item%2 ==1 //判断这个数组是否全部的数字都为奇数（就是余数为1）

})

}

console.log(aeeevery(arr1))

console.log(aeeevery(arr2))

--------------------------------------------------区分线--------------------------------------------------------------

1. arr.reduce(prev,cur,index,arr)//求数组的和,阶乘,（从左往右）
2. // arr.reduce()求数组的和
3. let arr=[1,2,3,4,5,6,7,8,9,10]
4. let aa=arr.reduce((prev,cur,index,arr)=>{//（之前的数，现在当前的值，当前的所引，当前的数组）
5. console.log(prev)
6. return prev+cur
7. })
8. console.log(aa)

执行顺序：

1. prev：0 cur：1 return 1
2. prev：1 cur：2 return 1+2
3. prev：3 cur：3 return 3+3
4. prev：6 cur：4 return 6+4

…

let arr=[2,2,4]

let aa=arr.reduce((prev,cur,index,arr)=>{//（之前的数，现在当前的值，当前的所引，当前的数组）

console.log(prev,cur)

console.log()

return Math.pow(prev,cur)

})

console.log(aa)

执行顺序：

1. prev：0 cur：2 return 2
2. prev：2 cur：2 return 2\*2
3. prev：4 cur：4 return 256(4\*4\*4\*4)

7. arr.reduceRight()//求数组的和,阶乘,（从右往左）

let arr=[2,2,3]

let aa=arr.reduceRight((prev,cur,index,arr)=>{//（之前的cur数，现在当前的值，当前的所引，当前的数组）

console.log(Math.pow(prev,cur))

return Math.pow(prev,cur)

})

console.log(aa)

执行顺序：

1. prev：0 cur：3 return 3
2. prev：3 cur：2 return 3\*3
3. prev：9 cur：2 return 9\*9

8.for of for (let aa of arr){}

for..of适用遍历数/数组对象/字符串/map/set等拥有迭代器对象的集合.但是不能遍历对象,因为没有迭代器对象

arr.keys()数组的下标

arr.entries()数组的某一项

let arr=["apple","banana","orange"]

for(let val of arr){//arr循环每一项的值

console.log(val)

}

for(let val of arr.keys()){//arr.keys()循环每一项索引的值

console.log(val)

}

for(let val of arr.entries()){//arr.entries()循环每一项索引和值

console.log(val)

}

补充：for in

遍历对象 通常用for in来遍历对象的键名

var myObject={

　　a:1,

　　b:2,

　　c:3

}

for (let val in myObject) {

console.log(val);

}

for of遍历的只是数组内的元素，而不包括数组的原型属性method和索引name

9.Array.from()//返回一个数组（Array.from()适用于所有具备length属性的数据类型的转换为数组）

window.onload=function(){

let ali=document.querySelectorAll("ul li")

let allt=Array.from(ali)

console.log(allt)

}

10.Array.of():把一组值转换为数组

let arr=Array.of('apple','banaba','orange')

console.log(arr)

11.arr.find()//找出数组里面第一个符合的值返回,如果没有就返回undefined

let arr=[2,10,15,19,30,42,48]

let sd=arr.find(function(val,index,arr){

return val>15//找出数组里面第一个符合的值返回,如果没有就返回undefined

})

console.log(sd)

12. arr. findIndex()//找出数组里面第一个符合的值返回该值的索引,如果没有就返回-1

let arr=[2,10,15,19,30,42,48]

let sd=arr.findIndex(function(val,index,arr){

return val>115//找出数组里面第一个符合的值返回,如果没有就返回undefined

})

console.log(sd)

13.arr.fill()填充

arr.fill(填充的东西，开始位置（包含），结束位置（不包含）)

let arr=new Array(10)

arr.fill('填充的内容','1','4')

console.log(arr)

14.arr.includes()//includes() 方法用来判断一个数组是否包含一个指定的值，如果是返回 true，否则false

let site = ['runoob', 'google', 'taobao'];

console.log(site.includes('runoob'))

补充：indexOf() 方法可返回某个指定的字符串值在字符串中首次出现的索引

let site = ['runoob', 'google', 'taobao','google'];

console.log(site.indexOf('google'))

十．对象json

1.jison定义对象

let name="aa"

let age="18"

let jison={

name,

age,

show(){

return this.name

},

nones(){

return this.age

}

}

console.log(jison.show())

2.Object.is(a,b)比较两个东西是否相等,返回true,false

console.log(Object.is(+0,-0))

console.log(Object.is(+0,-0))

3.Object.assign()用来：1)复制对象，2)合并数组，对象

Object.assign(目标对象，放进目标对象的参数1，放进目标对象的参数2，放进目标对象的参数3)

目标对象可以是数组也可以是对象，放进目标对象的参数里面的属性名相同的话，放进目标对象的参数位置在后面的会覆盖前面的值

let json={a:1}

let json2={b:2,a:2}

let json3={c:3}

let ss={}//ss=[]

let obj=Object.assign(ss,json,json2,json3)

console.log(obj)

console.log(ss)

4. Object.keys()，返回一个数组，成员是参数对象自身的（不含继承的）所有可遍历（enumerable）属性的键名

Object.values()，返回一个数组，成员是参数对象自身的（不含继承的）所有可遍历（enumerable）属性的键值

Object.entries()，返回一个数组，成员是参数对象自身的（不含继承的）所有可遍历（enumerable）属性的键值对数组。

1. 对象身上的拓展…

let json={a:1,b:2};

let json2= {...json}

console.log(json2)

十一.promise

1. new Promise((resolve,reject)=>{

// resolve调用成功

// reject失败的时候调用

}).then(res=>{//执行回调成功

}).catch(err=>{ //捕获错误

})

Promise. resolve:将现有的东西，转成一个Promise对象，resolve状态，成功状态

new Promise(resolve=>{ resolve(‘aaa’)})

Promise. reject:将现有的东西，转成一个Promise对象，reject状态，成功状态

new Promise(reject =>{ reject (‘aaa’)})

等价于：new Promise((resolve ,reject) =>{

resolve(‘aaa’);

reject(‘bbb’);

})

2. Promise.all([p1,p2,p3]):把Promise打包，扔到一个数组里面，打包完之后还是一个Promise对象。

Promise.all必须确保所有的Promise对象，都是resolve状态，才会执行then

let p1=Promise.resolve('aaa');

let p2=Promise.resolve('bbb');

let p3=Promise.resolve("ccx");

Promise.all([p1,p2,p3]).then(res=>{

// console.log(res)

let [res1,res2,res3]=res

console.log(res1,res2,res3)

}).catch(err=>{

console.log(err)

})

3.Promise.race([p1,p2,p3])只要参数中有一个率先得到结果就开始执行

let p1=Promise.reject('aaa');

let p2=Promise.resolve('bbb');

let p3=Promise.resolve("ccx");

Promise.race([p1,p2,p3]).then(res=>{

console.log(res)

}).catch(err=>{

console.log(err)//执行，以为p1先获取到结果

})

十二。模块化

Js不支持模块化

Ruby require

Puthon import

在ES6之前，社区制定一套模块规范

Commonjs 主要服务端nodejs require(‘http’)

AMD requires,CURIJS

CMD seaJs

ES6出来，统一服务端和客户端模块规范

imputer{xx} ddd

Math.pow()

Math.abs()

1. 如何让定义模块

export a

export let a=12

export {a as aa} 引用时使用aa

1. 如何使用

<script type="module"> import './js/mkh.js'</script>

Import 特点：

1. import可以是相对路径，也可以是绝对路径
2. import只会导入一次（无论引入多少次）
3. import {a as aa} from ‘./js/mkh.js’
4. import \* as mod from './js/mkh.js';
5. 有提升效果，import会自动提升到顶部，首先执行
6. 导出去模块内容，如果里面有定时器或者其他更改到值，引用的页面页会跟着改变

补充:Import()类似node里面require，可以动态引入，默认Import语法不能写到if里面

Import的返回值是个Promise对象，所以可以使用Promise的一切属性

import('./js/mkh.js').then(res=>{

console.log(res)

})

优点： 1.按需加载

2．可以写在if中

3.路径页可以动态

3．async await:异步加载

// 异步加载、

async function mani(){

const mod1=await import('./js/mkh.js')

const mod2=await import('./js/copy.js')

console.log(mod1,mod2)

const [m1,m2]=await Promise.all([

import('./js/mkh.js'),

import('./js/copy.js')

])

console.log(m1,m2)

}

mani()

4.使用严格模式’use strict’

十三。类和继承

1. 面向对象 ，类

函数摸拟

ES5语法：

function prop(name,age){

this.name=name

this.age=age

}

prop.prototype.show=function(){

return `显示为:${this.name},${this.age}`

}

let p=new prop('hmt','18')

console.log(p.show())

ES6语法：使用class类定义的方法没有预提升功能

class prop{//const prop=class {这样也可以函数定义，但是不推荐

constructor(name,age){

// console.log(`构造函数执行了，${name},${age}`);

this.name =name

this.age=age

}

showname(){return `名字为：${name}`}

showageg(){return `年龄为：${age}`}

}

let P1=new prop('hmt','18')

console.log(P1)

ES6调用变量作为方法名称使用【】

let aaa='strive'

const prop=class {

constructor(age){

this.age=age

}

showageg(){return `年龄为：${this.age}`}

aaa(){ return '我是aaa'}

[aaa](){//[]中括号可以引用到外面的变量，相当于是strive（）{}

return '我是变量定义的'

}

}

let P1=new prop('hmt','19')

console.log(P1.aaa())

// console.log(P1.strive())

console.log(P1[aaa]())

注意：

1. ES6里面没有提升功能，在ES5，用函数模拟可以默认函数提升
2. ES6里面this比之前轻松多了

补充矫正this:

1. fn.call(this指向谁，args1,args2…)
2. fn.apply(this指向谁，args1,args2…)
3. fn.bind(this指向谁，args1,args2…)

class pers{

constructor(){

this.name='hmts'

this.showname=this.showname.bind(this)//在这里，把constructor里面的this绑定到this.showname()

}

showname(){

console.log('this:',this)//这里面通过上面的this绑定，导致这里的this不会为空

return `名字为：${this.name}`

}

}

let p1=new pers()

let {showname}=p1

console.log(showname())

取值函数（getter）和存值函数（setter）

class pers{

constructor(){

}

get aaa(){

return `aaa的值：`

}

set aaa(val){

return `aaa的值：${val}`

// console.log(`${val}`)

}

}

let p1=new pers()

p1.aaa='123'

console.log(p1.aaa)

静态方法：类身上得对象 //定义的方法名称前面加上static,直接调用pers.queshow()

class pers{

constructor(){

}

aaa(){

return `我是定义的aaa`

}

static queshow(){//子类定义

return '我是静态的，可以直接通过定义的对象名称调用我'

}

}

let p1=new pers()

console.log(p1.aaa())

console.log(pers.queshow())////父类定义

继承

Es6之前

// ES6之前继承

// 父类

function person(name){

this.name=name

}

person.prototype.myname=function(){

return `my name is ${this.name}`

}

// 子类

function student(name,skill){

person.call(this,name)//名字继承

this.skill=skill

}

student.prototype=new person()//方法继承

let stu1=new student('manting','go home')

console.log(stu1.name)

console.log(stu1.myname())

Es6继承

// 父类

class persons{

constructor(name){

this.name=name

}

myname(){

console.log(`父类的名字${this.name}`)

return `my name is ${this.name}`

}

}

//继承父类

class student extends persons{

constructor(name,skill){//这里继承父类persons的参数name

super(name)//super继承父类定义的name

this.skill=skill//父类没有定义的skill，子类自己定义skill

}

myname(){

super.myname()//super调用父级的myname在子类的myname里面

// console.log(`子类的我想随便写的方法`)

return `子类的我想随便写的方法`

}

showskill(){//子类里面自己的函数

return `my skill is ${this.skill}`

}

}

let stu2=new student('manting','go home')

//console.log(stu2.skill)//通过子类调用父级的属性

console.log(stu2.myname())//通过子类调用父级的方法，因为子类super调用父级的myname在子类的myname里面，所以这里可以调用到父类的myname（）

// console.log(stu2.showskill())//调用子类自己的方法

拖拽

// 拖拽

class drag{

constructor(id){

this.odiv=document.querySelector(id)

this.disX=0

this.disY=0

this.init();

}

init(){

this.odiv.onmousedown = function(ev){

// console.log("1")

this.disX=ev.clientX-this.odiv.offsetLeft;

this.disY=ev.clientY-this.odiv.offsetTop;

document.onmousemove=this.fnMove.bind(this)//这里把this绑定到fnMove里面

document.onmouseup=this.fnUp.bind(this)

// console.log(this.disX,this.disY)

}.bind(this)//这里把this绑定到init（）里面

}

fnMove(ev){

this.odiv.style.left=ev.clientX-this.disX+'px'

this.odiv.style.top=ev.clientY-this.disY+'px'

}

fnUp(){

document.onmousemove=null

document.onmouseup=null

}

}

// class tre extends drag{}//直接继承

class tre extends drag{//div2限制范围

fnMove(ev){

super.fnMove(ev)

if(this.odiv.offsetLeft <= 0){

this.odiv.style.left='0px'

console.log(this.odiv.style.top)

}

}

}

new drag('#div1')

new tre('#div2')

--------------------------------------------------------------------------------------------

十四Symbol

1.注意：

1. Symbol不能new
2. Symbol返回的是一个唯一的值，作为key定义一些唯一的东西
3. 用typeo检测出来的数据类型是symbol(是一个单独的数据类型)
4. 当成一个key使用，当使用for in循环，是出不来的

//Symbol('aaa')可以当成一个key使用

let sym= Symbol('aaa')

let json={

name:'libai',

age:'18',

[sym]:'tang'

}

console.log(json[sym])

for(let key in json){

console.log(key)

}

1. 箭头函数()=>{}

generator函数

生成器：

异步加载（解决异步，深度嵌套）

for of也可以自动遍历，只不过return会被遍历

// 异步加载

function \* show(){

yield 'welcome'

yield 'to'

return '牧马人'

}

let g1 = show()

// generator的调用

console.log(g1.next())//{value: "welcome", done: false}

console.log(g1.next())//{value: "to", done: false}

console.log(g1.next())//{value: "牧马人", done: true}

//因为已经结束所以下面再调用就没有value值

console.log(g1.next())//{value: undefined, done: true}

//for of也可以自动遍历，只不过return会被遍历

for(let val of g1){

console.log(val)

}

1.解构赋值（配合结构）

let [a,b,c]=show()

console.log(a,b,c)// welcome to undefined

let [a,...b]=show()

console.log(a,b)// welcome ["to"]

2.扩展运算符‘…’

console.log(...show())//自动遍历出每一个值welcome to

3.Array.from()

console.log(Array.from(show()))// ["welcome", "to"]

十五。Async---- await

十六。数据结构

**1.Set**

1）定义new Set([1,2,3,4,4])

 const setarr=new Set([1,2,3,4,4])

        setarr.clear()

2）增加元素setarr.add('a')//增加元素

3）删除元素

setarr.delete(4)

4）setarr.values()

for(let item of setarr){//默认试setarr.values()

   console.log(item)

}

5）for…..of…

 for(let item of setarr){} 默认试setarr.values()

for(let item of setarr){//默认试setarr.values()

        //     console.log(item)

        // }

for(let item of setarr.keys()){}

for(let item of setarr.keys()){

 console.log(item)

}

for(let item of setarr.values()){}

for(let item of setarr.values()){

console.log(item)

}

Map 循环遍历返回每个处理后的值

let set=new Set([1,2,3,4,4])

 console.log([...set])

 set=new Set([...set].map(val=>val\*2))

 console.log(set)

filter筛选出满足判断条件的元素返回

let set=new Set([1,2,3,4,5,6])

console.log([...set])

 set=new Set([...set].filter(val=>val%==0))

  console.log(set)

6）往set里面增加json数据 set.add()

let set=new Set()

 let json={

    a:1,

b:2,

  c:3

 }

let json2={

   d:1,

e:2,

   f:3

 }

set.add(json).add(json2)

console.log(set)

2. WeakSet

定义：

const ws = new WeakSet()

WeakSet 的成员只能是对象，而不能是其他类型的值

* **WeakSet.prototype.add(value)**：向 WeakSet 实例添加一个新成员。
* **WeakSet.prototype.delete(value)**：清除 WeakSet 实例的指定成员。
* **WeakSet.prototype.has(value)**：返回一个布尔值，表示某个值是否在

3. Map （类似于json，但json的键（key）只能是字符串，map的key可以是任意类型）

1)定义

Var Maps=new Map()

2）增加内容map.set()

map.set('key','value')

map.set(json,'qq1')//键名可以是对象

map.set('qq2',json)//键值可以是对象

3）map.get() //get方法用来获取一个Map对象中指定元素

map.get(json)

4)map.delete(key)// delete删除一项

map.delete(json)

5）map.has(key)//判断有没有某个键（某一项），返回true/false

map.has(json)

6）map.clear()//清空

map.clear()

7)map. forEach()

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

    console.log(val,key)

})

1. WeakMap()：key值只能是对象
2. 定义：new WeakMap()
3. WeakMaps.set(json,'aaa')//WeakMap {{…} => "aaa"},WeakMapde的key值只能是对象。WeakMaps.set('aaa'，json)会报错

总结：：：：

Set 里面是数组，不重复，没有key,没有get方法

Map 对json功能增强，key可以是任意类型值

十七数字（数值）变化

1.

isInteger判断是不是整数

Number.isInteger(a)

isFinite判断是不是数字

console.log(Number.isFinite(a))//isFinite判断是不是数字

Number.parseInt( )

console.log(Number.parseInt(a))//转成数字

Number.parseFloat( )

console.log(Number.parseFloat(a))//解析一个字符串，并返回一个浮点数

2.安全整数

安全整数：-(2\*\*53-1)到(2\*\*53-1)

Number.MAX\_SAFE\_INTEGER //最大的安全整数Number.MIN\_SAFE\_INTEGER //最小的安全整数

console.log(2\*\*53-1)

console.log(Number.MAX\_SAFE\_INTEGER)//最大的安全整数

console.log(-(2\*\*53-1))

console.log(Number.MIN\_SAFE\_INTEGER)//最小的安全整数

3.Math.trunc(5.5)//截取，保留整数部分

 console.log(Math.trunc(5.5))//5

4. Math.sign()判断是证书还是负数-1/1/0/-0

console.log(Math.sign(-5))

5. Math.cbrt()计算一个数的立方根

console.log(Math.cbrt(27))

十八ES9

1命名捕获

语法：(?<名字> )

let str='2019-06-02'  let reg=/(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/

let str='2019-06-02'

let reg=/(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/

console.log(str.match(reg).groups)

let {year,month,day}=str.match(reg).groups//到对应得值赋值分配

console.log({year,month,day})

2反向引用命名捕获

反向引用

\1 \2 $1 $2

语法：\k<名字> let regs=/^(?<strives>welcome)-\k<strives>$/ welcome- welcome

let regs=/^(?<strives>welcome)-\k<strives>$/

let sage3='welcome-welcome'

console.log(regs.test(sage3))

'welcome-welcome-welcome'

// 通过\1   \2   $1   $2进行反向引用

let regsss=/^(?<strives>welcome)-\k<strives>-\1$/

console.log(regsss.test(sage4))//true

替换：$<名字>

使用改名字对应的

let time='2019-06-02'

let regsr=/(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/

console.log(str.match(reg).groups)

//02/06/2019

var times=time.replace(regsr,'$<day>/$<month>/$<year>')// $<名字>

console.log(times)

3.dotAll模式 s

\w匹配大 小写字母数字和下划线

之前’.’在正则里表示匹配任意东西，但是不包括\n，当存在要匹配\n的时候，在后面加上/s就可以匹配

Let res=/\w+/gims

4标签函数

使用：fn`aaa`

  function fsd(args){

            return args

        }

        console.log(fsd`welcome`)

function fsdss(args){

            return args[0].toUpperCase()//第一项就是传入的参数

        }

        console.log(fsdss`welcome`)

十九。Proxy代理(拦截处理)

1. 作用：扩展（增强）对象一些功能

Vue

Vue.config.keyCodes.enter=65

Proxy作用：比如vue中拦截

预警，上报，扩展功能，统计，增强对象等等

1. 用法

Let obj={

Name:’strive’

}

定义：

let obj=new Proxy(target,handler)

let obj=new Proxy(被代理的对象，对代理对象进行操作)

{

Set(){}//设置的时候干的事情

Get(){}//获取干的事情

deleteProperty(){}//删除

has(){}//有没有这个东西

apply(){}//调用函数处理

}

3.get（）使用

实现一个，访问一个对象身上属性，偶人不存在的时候给了undefined,希望如果不存在错误（警告）信息。（通过^-^来提示错误警告）

        let obj = {

            name: "strive"

        }

        let newobjs=new Proxy(obj,{

            get(target,property){//获取

                if(property in target){

                    return target[property]

                }else{

               // throw new ReferenceError(`${property}属性不在此对象上`)

                    console.warn(`${property}属性不在此对象上`)

                    return '^-^'

                }

            }

        })

        console.log(newobjs.name)

        console.log(newobjs.age)

3.DOM.div()

DOM.a()

DOM.ul()

//============================== DOM.div()=============================

const DOM=new Proxy({},{

    get(target,property){

        console.log(target,property)//property就是DOM.div里面的div

        return function(attr={},...children){

            // console.log(attr)

            // console.log(children)

            const el=document.createElement(property)//创建节点

            for(let key of Object.keys(attr)){

                el.setAttribute(key,attr[key])

                console.log()

            }

            for(let child of children){

                if(typeof child == 'string'){

                    child = document.createTextNode(child)//创建节点文本内容

                }

                el.appendChild(child)

            }

            return el

        }

    }

})

let odiv = DOM.div(

    {id:"div1",class:"aaa"},'我是第二个参数','呵呵',

    DOM.a({href:"https://www.baidu.com/"},"百度一下")

)

console.log(odiv)

document.body.appendChild(odiv)

4.set（）使用

设置一个年龄，保证是整数，且范围不能超过200

// ================set=======================

let obj=new Proxy({},{

    set(target,prop,value){//target指被定义的对象，prop指obj.a中a的名称，value指prop的值

        // console.log(target,prop,value)

        if(prop=='age'){

            if(!Number.isInteger(value)){

                throw new TypeError(`年龄必须为整数`)

            }

            if(value>200){

                throw new RangeError(`年龄不能超过200`)

            }

        }

        target[prop]=value

    }

})

obj.a = 123;

obj.name = 'Strive';

console.log(obj)

obj.age=12.5//报错

5.deleteProperty（）：删除，拦截

let obj = {a:1,b:2}

let newJson=new Proxy(obj,{

    deleteProperty(target,property){

        console.log(`您要删除${property}属性`)

        delete target[property]

    }

})

delete newJson.a

console.log(newJson)

6.has()判读有没有

let obj = {a:1,b:2}

let newJson=new Proxy(obj,{

    has(target,property){

        console.log(`判断是否存在调用has方法`)

        return property in target

    }

})

console.log('a' in newJson)//true

console.log(newJson)

7. apply(){}//调用函数处理拦截

function fn(){

    return '我是函数'

}

let newfn=new Proxy(fn,{

    apply(){

        return '函数？'

    }

})

console.log(newfn())//拦截调用fn函数，直接返回内容

二十。Reflect.apply(调用的函数，)

1. Reflect.xxx()语言内部方法

Object.defineProperty

放到Reflect对象身上

通过Reflect对象身上直接拿到语言内部东西

 'assogn' in Object  --> Reflect.has(Object,'assign')delete json.a  --> Reflect. deleteProperty(json,'a')