0201

=====================================

模块化:

js不支持模块化

ES6出来了，就统一了服务端和客户端模块规范：

模块化：

需要放到服务器环境

1. .如何定义模块？

export 东西

export { a as aaa, b as bbb, c as ccc };

1. .如何使用？

import

import ‘./modules/1.js’

import {a as aa,b as bb,c} from ‘./modules/1.js’

import \* as modTwo from "./modules/1.js";

使用模块

<script type=”module”></script>

import:

1. .import 科可以是相对路径也可以是绝对路径
2. .import模块只会导入一次，无论你引入多少次
3. .import ‘./modules/1.js’ 如果这么用，相当于引入文件
4. .有提升效果，import会自动提升到顶部，首先执行
5. .导出去的模块内容，如果里面有定时器更改了，外面也会改动。不像common规范会有缓存。

\*

import() 类似node里面require，可以动态引入 ,默认import

语法不能写到if之类里面。

这个import()返回值，是个promise对象

优点:

1.按需加载

2.可以写到if中

1. 路径也可以动态

Promise.all([])

Promise.all([import("./modules/1.js"), import("./modules/2.js")]).then(

([mod1, mod2]) => {

console.log(mod1, mod2);

}

);

====================================

ES2017加了 async await:

================================

Es6里面自动加严格模式 ‘use strict’

0202

=========================

程序中类

ES6

面向对象，类

属性

方法

没有es6的时候是拿函数模拟的

人：Person

属性：name

展示名字：showName

Person.prototype.showName

ES5之前：

function Person (){

this.name = ‘aaa’;

}

Person.prototype.showName = function(){}

ES6中变形：

class Person{

constructor(){

this.name = ‘aaa’;

}

showName(){}

}

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

const Person = class{}

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

let a = ‘strive’;

let b = ‘method’;

Class Person{

[a+b](){}

}

let aaa = "suibian";

let bbb = "method";

class Person {

constructor(name, age) {

//构造方法（函数），调用new，自动执行

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

this.name = name;

this.age = age;

}

showName() {

return `名字是：${this.name}`;

}

showAge() {

return `年纪是：${this.age}`;

}

[aaa + bbb]() {

return "随便了。。。。。。";

}

}

let p1 = new Person("joyce", 18);

console.log(p1.suibianmethod());

console.log(p1[aaa + bbb]());

注意：

1. ES6里面class没有提升功能，ES5用函数模拟可以，默认函数提升

1. ES6里面的this比之前轻松多了

矫正this:

1. fn.call(this指向谁，arg1,arg2.....)
2. fn.apply(this指向谁，[arg1,arg2......])
3. fn.bind(this)

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

class里面取值函数(getter),存值函数(setter)

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

静态方法：就是类身上的方法。

class Person {

constructor() {}

showName() {

return `这是showName方法`;

}

static aaa() {

return "这是静态方法";

}

}

let p1 = new Person();

console.log(p1.showName());

console.log(Person.aaa());

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

父类

子类

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

继承：

之前

Person

Student

现在：extends

class Student extends Person {}

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

拖拽demo

Es6 类

----------

数据类型：

number、string、boolean、Object、undefined、function

用typeof检测出来数据类型：symbol

new Number(12)

new String()

new Array()

symbol 使用情况一般

定义：

let syml = Symbol('aaa')

注意：

1. Symbol 不能new
2. Symbol()返回是一个唯一值

坊间传说，做一个key，定义一些唯一或者私有一些东西

1. symbol是一个单独的数据类型，就叫symbol，基本类型
2. 如果symbol作为key，用for in循环，出不来

json ->for in

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

箭头函数

（）=>{}

generator函数

生成器

解决异步，深度嵌套的问题

语法：

function \* show(){}

定义：

function \* gen(){

yield 'welcome';

yield 'to';

return 'mmr';

}

调用：

let g1 = gen();

console.log(g1.next());//generator2.html:18 {value: "welcome", done: false}这里的done false的意思是后面还有东西没有执行完

console.log(g1.next())//generator2.html:19 {value: "to", done: false}

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

上述调用，手动调用，麻烦

for ... of 自动遍历generator

return的东西，它不会遍历

generator不仅可以配合for...of...

还可以：

1. 解构赋值：

function \* gen(){

yield 'welcome';

yield 'to';

yield 'lalalla';

return 'mmr';

}

// let [a,b] = gen()

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

console.log(a,b); // welcome (2) ["to", "lalalla"]

1. 扩展运算符

‘...’

function \* gen(){

yield 'welcome';

yield 'to';

yield 'lalalla';

return 'mmr';

}

console.log(...gen(),'...gen()') //welcome to lalalla ...gen()

1. Array.from()

function \* gen(){

yield 'welcome';

yield 'to';

yield 'lalalla';

return 'mmr';

}

console.log(Array.from(gen()),'Array.from(gen())') //["welcome", "to", "lalalla"] "Array.from(gen())"

generator结合axios 数据请求：

主要是解决异步的问题

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

异步：不连续，上一个操作没有执行完，下一个操作昭阳开始

同步：连续执行，上一个操作没有执行完，下一个操作没法开始

关于异步，解决方案：

1. 回调函数
2. 事件监听
3. 发布\订阅
4. Promise对象

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

Es2017，规定了async

nodeJs

读取文件 fs.readFile

1. Promise
2. Generator
3. async

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

async function fn(){ //表示异步，这个函数里面有异步的任务

let result = await xxx //表示后面的结果需要等待

}

async特点:(例子是async2.js 需要在终端中输入node async2.js去看结果)

1. await只能放到async函数中
2. 相比generator语义化更强
3. await后面可以是一个promise对象，也可以数字，字符串、布尔
4. async函数返回是一个promise对象
5. 只要await语句后面promise状态变成了reject，那么整个async函数会中断执行。 (async4.html)

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

如何解决async函数中抛出错误，影响后续代码：

1、用try

try{

await Promise.reject('.出现问题了')

}catch(e){}

1. promise本身catch

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

个人建议大家：

有await的地方都加上try

try{

let f1 = await readFile("data/a.txt");

let f2 = await readFile("data/b.txt");

let f3 = await readFile("data/c.txt");

}catch(e){}

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

Promiss.all([ ])

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

数据结构

数组

json ，二叉树。。。

set数据结构：

类似数组，但是里面不能有重复值

let arr = [‘a’,’b’,’c’]；

let arr = new Array();

set用法：

let setArr = new Set([‘a’,’b’]);

setArr.add('a')

往setArr里面去添加一项

setArr.delete('b')

删除一项

console.log(setArr.has('b'),"setArr.has('b')")

判断setArr里面有没有此值

console.log(setArr.size,"setArr.size")

这是属性，去查看个数

setArr.clear()

清空

for...of....

let setArr = new Set(["a", "b", "c", "d"]);

for (let item of setArr) { //默认是values

console.log(item);

/\* a

b

c

d \*/

}

循环：

1. for(let item of setArr ){} //默认的是values
2. for(let item of setArr.keys() ){}
3. for(let item of setArr.values() ){}
4. for(let [k,v] of setArr.entries() ){}
5. let setArr = new Set(["a", "b", "c", "d"]);

setArr.forEach((value,index)=>{

console.log(value,index)

})

let setArr= new Set().add('a').add('b').add('c')

数组去重：

var arr = [1,1,'true','true',true,true,15,15,false,false, undefined,undefined, null,null, NaN, NaN,'NaN', 0, 0, 'a', 'a',{},{}]; console.log([...new Set(arr)],[...new Set(arr)].length); // [1, "true", true, 15, false, undefined, null, NaN, "NaN", 0, "a", {…}, {…}] 13

set数据结构变成数组：

[...set]

想让set使用数组的map循环和filter：

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

let arr =[{},{}];

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

new Set([]) 存储数组，这种写法对

new Weakset({}) 存储json,这种写法不靠谱

Weakset没有size，没有clear，有add(),has(),delete()

let wset=new WeakSet()

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

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

wset.add(json).add(json2)

console.log(wset.size)//undefined

确认，初始往里面添加东西，是不行的。最好用add添加

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

总结：new Set()

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

map:

类似于json，但是json的键（key）只能是字符串

map的key可以是任意类型

使用：

let map=new Map()

map.set(key,value) //设置一个值

获取一个值

let map=new Map()

let json={

a:1,b:2

}

map.set('a','aaa')

map.set(json,'aaa')

map.set('aaaa',json)

console.log(map)

console.log(map.get(json))

console.log(map.get('aaaa'))

map.delete(key) 删除一项

map.has(key) 删除一项

map.clear()清空

循环：

for(let [key,value] of map){}

for(let key of map.keys()){}

for(let values of map.values()){}

for(let [k,v] of map.entries()){}

map.forEach()

map.forEach((value,key)=>

{

console.log(value,key)

})

Weakmap（）：key只能是对象

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

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

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

2021-2-28

数字（数值）变化

NaN isNaN

二进制:(binary)

let a = 0b010101