1. const 常量 尽量全大写 要赋初值
2. 保存this：\_this that self
3. 扩展运算符可以将伪数组转换为真数组

const divArr = [...divs];

1. Symbol

let s2 = Symbol('尚硅谷');

        let s3 = Symbol('尚硅谷');

        console.log(s2 === s3); // false

        let s4 = Symbol.for('尚硅谷');

        let s5 = Symbol.for('尚硅谷');

        console.log(s4 === s5); // true

let game = {

            name: '狼人杀',

            [Symbol('say')]: function() {

                console.log('我可以发言');

            },

            [Symbol('zibao')]: function() {

                console.log('我可以自爆');

            }

        }

1. 数据类型

        // USONB you are so niubility

        // u    undefined

        // s    string  symbol

        // o    object

        // n    null  number

        // b    boolean

1. 自定义遍历

const banji = {

            name: '终极一班',

            stus: [

                'xiaoming',

                'xiaoning',

                'xiaotian',

                'knight'

            ],

            // 加上iterator接口

            [Symbol.iterator]() {

                // 索引变量

                let index = 0;

                // 下面也可以写成箭头函数

                let \_this = this;

                return {

                    next: function() {

                        if (index < \_this.stus.length) {

                            const result = {value: \_this.stus[index], done: false};

                            // 下标自增

                            index++;

                            // 返回结果

                            return result;

                        } else {

                            return {value: undefined, done: true};

                        }

                    }

                };

            }

        }

        // 遍历这个对象

        for (let v of banji) {

            console.log(v);  // 报错，因为没有iterator接口

        }

1. 生成器函数

       // 模拟获取  用户数据  订单数据  商品数据

        function getUsers() {

            setTimeout(() => {

                let data = '用户数据';

                // 调用 next 方法，并且将数据传入

                iterator.next(data);

            }, 1000);

        }

        function getOrders() {

            setTimeout(() => {

                let data = '订单数据';

                iterator.next(data);

            }, 1000)

        }

        function getGoods() {

            setTimeout(() => {

                let data = '商品数据';

                iterator.next(data);

            }, 1000)

        }

        function \* gen() {

            let users = yield getUsers();

            console.log(users);

            let orders = yield getOrders();

            console.log(orders);

            let goods = yield getGoods();

            console.log(goods);

        }

        // 调用生成器函数

        let iterator = gen();

        iterator.next();

1. Set()

let s2 = new Set(['大事儿', '小事儿', '好事儿', '坏事儿'])  // 会自动去重

// 添加新的元素

s2.add('喜事儿');

// 删除元素

s2.delete('坏事儿');

// 检测

console.log(s2.has('好事儿'));  // true

for (let v of s2) {

    console.log(v);

}

1. 数组去重 let result = [...new Set(arr)];
2. Map()

let m = new Map();

// 添加元素

        m.set('name', '尚硅谷');

        m.set('change', function() {

            console.log('我们可以改变你!!');

        })

        let key = {

            school: 'atguigu'

        }

        m.set(key, ['北京', '上海', '深圳']);

        // 删除

        m.delete('name');

        // 获取

        console.log(m.get('change'));

        console.log(m.get(key));

        // 遍历

        for (let v of m) {

            console.log(v);

        }

1. class 注意构造函数为：constructor()
2. ES5构造函数继承

        // 手机

        function Phone(brand, price) {

            this.brand = brand;

            this.price = price;

        }

        Phone.prototype.call = function() {

            console.log('我可以打电话');

        }

        // 智能手机

        function SmartPhone(brand, price, color, size) {

            Phone.call(this, brand, price);

            this.color = color;

            this.size = size;

        }

        // 设置子级构造函数的原型

        SmartPhone.prototype = new Phone;

        SmartPhone.prototype.constructor = SmartPhone;

        // 声明子类的方法

        SmartPhone.prototype.photo = function() {

            console.log('我可以拍照');

        }

        SmartPhone.prototype.playGame = function() {

            console.log('我可以玩游戏');

        }

1. Set和Get

        class Phone {

            get price() {

                console.log('价格属性被读取了');

                return 'I love you';

            }

            set price(newVal) {

                console.log('价格属性被修改了');

            }

        }

1. 模块暴露

// 分别暴露     m1.js

export let school = '尚硅谷'

export function teach() {

    console.log('我们可以教给你发开技能');

}

// 统一暴露     m2.js

let school = '尚硅谷'

function findJob() {

    console.log('我们可以帮助你找工作!!');

}

export {school, findJob};

// 默认暴露     m3.js

export default {

    school: 'atguigu',

    change: function() {

        console.log('我们可以改变你!!');

    }

}

// 1. 通用的导入方式

        // 引入 m1.js 模块内容

        import \* as m1 from './js/ms1.js';

        // 引入 m2.js 模块内容

        import \* as m2 from './js/ms2.js';

        // 引入 m3.js 模块内容

        import \* as m3 from './js/ms3.js';

        m3.default.change();

        // 2. 解构赋值形式

        import {school, teach} from './js/m1.js';

        // import {school, findJob} from './js/m2.js';     // 如此写有俩school，不行

        import {school as guigu, findJob} from './js/m2.js';    // 用别名

        console.log(guigu, findJob);

        import {default as m3} from './js/m3.js';

        console.log(m3);

        // 3. 简便形式  针对默认暴露

        import m3 from './js/m3.js';

        console.log(m3);

// 入口文件

// 模块引入

import \* as m1 from './m1.js';

import \* as m2 from './m2.js';

import \* as m3 from './m3.js';

<script src="./5 js/app.js" type="module"></script>

动态 import

    import('./11hello.js').then(module => {

        // console.log(module);

        module.hello();

    })

1. 对象转Map const m = new Map(Object.entries(school));
2. 正则

        let str = '<a href="http://www.atguigu.com">尚硅谷</a>';

        const reg = /<a href="(?<url>.\*)">(?<text>.\*)<\/a>/;

        const result = reg.exec(str);

        console.log(result);

        console.log(result.groups.url);

        console.log(result.groups.text);

1. 私有属性

// 私有属性

            #age;

            #weight;

1. 可选链 const dbHost = config?.db?.host;
2. bigInt

console.log(BigInt(1.2));   // 报错

        // 大数值运算

        let max = Number.MAX\_SAFE\_INTEGER;

        console.log(BigInt(max) + BigInt(1));