1. async函数简介： Async函数是一个基于Promise的generator语法糖，以同步流程表达异步操作。

1. async函数语法：

async function 函数名(){

await 异步操作1;

await 异步操作2;

}

1. async函数特点：

1不需要像Generator去调用next方法，遇到await等待，当前的异步操作完成就往下执行

2返回的总是Promise对象，可以用then方法进行下一步操作

3async取代Generator函数的星号\*，await取代Generator的yield

4当给resolve()回调函数传参时会作为对应await语句的返回值

5语意上更为明确，使用简单

1. async用法注意:

async是用来修饰function函数的(放在function关键字前面)，被async修饰过的函数，返回值是一个promise对象(不管这个函数内部是否有return返回值)；这个promise对象的状态是resolved（成功状态）；除非在async函数内部出现异常，此时的promise状态是rejected；async函数返回的promise状态是自动的，我们不能像promise一样手动调用resolve和reject方法。

    <script>

        // async function demo() {

        //     return '成功...'

        // }

        // console.log(demo()); //返回的是promise对象

        // let prm = demo();

        // prm.then((d) => {

        //     console.log(d, 111)

        // }).catch((err) => {

        //     console.log(err, 333);

        // });

        //await必须与async结合使用，但async可以单独使用

        async function fn() {

            await '123';

        }

        // fn();

        async function test() {

            console.log('开始...');

            await 12;

            console.log('执行到这里');

            await 'Hi';

            console.log('完成...');

        }

        test();

    </script>

1. await用法注意：

        1、await要与async一起使用

         2、await通常等待的是Promise结果，当Promise结果为resolve()时await可以正常处理，但当Promise结果为reject()时await不能正常处理，我们可以使用这种方法来处理Promise的异常：resolve(['是否正常',结果])

         3、await后面也可以跟同步操作，但很少这样使用

    <script>

        //await特点：

        //1)await要与async一起使用

        //2)await通常等待的是Promise结果，当Promise结果为resolve()时await可以正常处理，但当Promise结果为reject()时await不能正常处理，我们使用这种方法来处理Promise的异常：resolve(['是否正常',结果])

        //3)await后面也可以跟同步操作，但很少这样使用

        function demo() {

            let p = new Promise((resolve, reject) => {

                setTimeout(() => {

                    console.log('ok');

                    //**\*\*\*注意异步操作完成后要手动调用resolve()方法，否则程序一直处于等待状态**

                    //  resolve();

                    //     if (true) { //正确

                    if (false) { //正确

                        //  resolve(33);

                        resolve([null, 33]);

                    } else { //错误

                        // reject('error message...');

                        resolve(['error message', '']);

                    }

                }, 2000);

            });

            return p;

        }

        async function fn() {

            console.log(1);

            //await能处理promise中resolve()，但不能处理reject()

            //异常处理方法一:

            // let d = await demo().catch((err) => {

            //     console.log(err, 9999);

            // });

            // let d2 = await demo().catch((err) => {

            //     console.log(err);

            // })

            //异常处理方法二：

            let d = await demo();

            if (d[0] === null) { //正常

                console.log('正确结果为：' + d[1]);

            } else { //异常

                console.log('失败原因：' + d[0]);

            }

            console.log(d, 2);

        }

        fn();

    </script>

1. async应用：async+promise+ajax解决异步回调地狱问题

   <script>

        //发送请求：Promise+ajax

        function sendAjaxs({

            url,

            type = 'get',

            data = {},

            dataType = 'json'

        } = {}) {

            let p = new Promise((resolve, reject) => {

                $.ajax({

                    url,

                    type,

                    data,

                    dataType,

                    success(d) {

                        // console.log(d, 111);

                        resolve([null, d]);

                    },

                    error(e) {

                        resolve([e, '']);

                    }

                });

            });

            return p;

        }

        //获取所有商品分类数据

        async function getCateData() {

            //获取所有商品一级分类数据

            // let data = await sendAjaxs({

            //     url: 'http://106.13.114.114:5000/api/firstCategory'

            // });

            let [err, datas] = await sendAjaxs({

                url: 'http://106.13.114.114:5000/api/firstCategory'

            });

            // console.log(datas);

            //根据某个一级分类id获取所有商品二级分类数据

            [err, datas] = await sendAjaxs({

                url: 'http://106.13.114.114:5000/api/secondCategory',

                data: {

                    firstId: datas['list'][0][0].firstId

                }

            });

            // console.log(err, datas);

            // 根据某个二级分类id获取所有商品三级分类数据

            [err, datas] = await sendAjaxs({

                url: 'http://106.13.114.114:5000/api/thiredCategory',

                data: {

                    secondId: datas['list'][0].secondId

                }

            });

            console.log(err, datas);

        }

        getCateData();

    </script>

1. 面向过程与面向对象？

1面向过程(中午吃蛋炒饭)： 第一步买食材-->第二步洗食材-->第三步开始做饭-->第四步吃饭

2面向对象(中午吃蛋炒饭) ：第一步打开app-->第二步选好商家并下单-->第三步外卖小哥送餐-->第四步吃饭

面向对象编程相对面向过程可以降低系统耦合度、提高代码重用度。

不同编程语言在实现面向对象编程时采用的方式不太一样， 像java、php等后端语言都采用的是“类”实现的，而javascript采用的“原型”

1. Es6中的类（class）：
2. 什么是类？

具有相同属性及方法(行为)的一类事务的集合。（比如：电脑、人类）

1. 什么是对象？

对象就是类中存在的某个具体的个体。(类实例化后称为对象)

1. 类的定义语法：

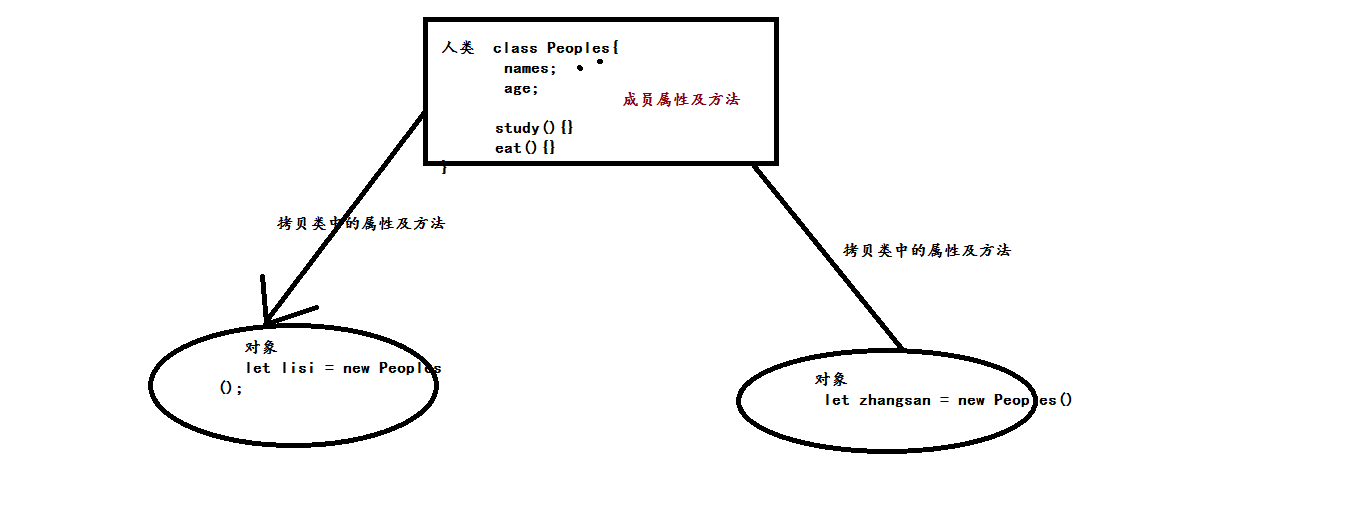
class 类名{

//属性

//方法

}

1. 成员属性和方法：在每创建一个对象时都会将类中的属性及方法拷贝一份，每个对象中都具有类中基本属性和方法，这样属性和方法称为成员属性和方法，成员属性和方法属于对象，而不属于类。



1. 静态属性和方法：

静态属性和方法是归类所有，全局独一份，在静态方法中只能访问静态属性，当然在成员方法中只能访问成员属性，静态方法直接通过类名来访问，不需要实例化。

**注意**：静态方法中this指向类本身，而成员方法中的this指向当前对象。

    <script>

        class Peoples {

            //属性

            // name;

            // age;

            //静态属性：

            static types = '人类';

            //方法

            //构造器方法：在类实例化时会被自动调用

            constructor(name, ages) {

                this.names = name

                this.age = ages;

                // console.log(name, ages, 'Hello...');

            }

            //静态方法

            static gettype() {

                console.log(this.age, 666);

                console.log(this.types, 999);

            }

            study() {

                //成员方法中不能访问静态属性及静态方法

                console.log(this.types);

                console.log(`${this.names}  ${this.age} 正在学习`);

                //  this.eat();

            }

            eat() {

                console.log(`${this.age} ${this.names} 正在吃饭`);

            }

        }

        //对象(类实例化后为对象)

        let lisi = new Peoples('李四', 20);

        lisi.names = 'lisi';

        lisi.study();

        //注意：静态方法不能通过对象来访问，直接通过类名来访问静态方法

        //lisi.gettype(); // lisi.gettype is not a function

        Peoples.gettype();

        //对象

        let zhangsan = new Peoples('张三', 19);

        zhangsan.eat();

    </script>

1. 类的继承：extends
2. 类的继承注意事项：

1)关于构造器方法：

      (1)、当子类中没有定义构造方法时默认访问父类的构造方法

            (2)、当子类中定义了构造方法时要调用super()方法，通过super()方法可以把父类中的this继承下来的同时也可以给父类中的属性设置属性值

2)方法重构/重写？

         (1)为什么要使用方法重构/重写？

            当父类中的方法在子类中不能满足需求时要对父类中的方法进行重构/重写.

           (2)怎样进行方法重构/重写？

             在子类定义和父类同名的方法即可。

    <script>

        //定义类[人类]

        class Peoples {

            static types = '人类';

            constructor(name, ages) {

                this.names = name;

                this.age = ages;

                console.log('哈哈...');

            }

            study() {

                console.log(`${this.names} ${this.age}正在学习`);

            }

            eat() {

                console.log(`${this.names} ${this.age}正在吃饭`);

            }

        }

        //定义类[学生类]:类的继承

        class Student extends Peoples {

            //注意：

            //1)、当子类中没有定义构造方法时默认访问父类的构造方法

            //2)、当子类中定义了构造方法时要调用super()方法，通过super()方法可以把父类中的this继承下来的同时也可以给父类中的属性设置属性值

            constructor(xuehao, name, age) {

                //  console.log(xuehao, name, age, 88888);

                super(name, age);

                this.xuehao = xuehao;

            }

            //为什么要使用方法重构/重写？

            //当父类中的方法在子类中不能满足需求时要对父类中的方法进行重构/重写.

            //怎样进行方法重构/重写？

            //在子类定义和父类同名的方法即可。

            study() {

                console.log(`${this.age}岁的${this.names}正在认真地学习！！！`);

            }

            static homework() {

                console.log(this.types, 8889);

            }

            //访问父类中的方法

            play() {

                this.eat();

            }

        }

        //定义类[工人]:

        class Worker extends Peoples {

        }

        //访问静态方法：

        Student.homework();

        let xiaoming = new Student('xm321', '小明', 18);

        xiaoming.study();

        console.log(xiaoming.age, 999);

        xiaoming.play();

    </script>

1. 类的应用： 将所有商品一级分类名称展示到div标签中

    <script>

        //需求：将所有商品一级分类名称展示到div标签中

        //使用技术：promise+async+ajax

        class ShowData {

            constructor(url, type = 'get', data = {}, dataType = 'json') {

                this.url = url;

                this.type = type;

                this.data = data;

                this.dataType = dataType;

            }

            //方法:获取数据

            sendAjax() {

                let p = new Promise((resolve, reject) => {

                    $.ajax({

                        url: this.url,

                        type: this.type,

                        data: this.data,

                        dataType: this.dataType,

                        success(d) {

                            // console.log(d);

                            resolve([null, d]);

                        },

                        error(err) {

                            // console.log(err);

                            resolve([err, '']);

                        }

                    });

                });

                return p;

            }

            //方法：展示数据

            async render() {

                let [err, data] = await this.sendAjax();

                let arr = data.list;

                let divs = '';

                arr.forEach(item => {

                    //console.log(item);

                    item.forEach(rt => {

                        divs += `<li>${rt.firstName}</li>`;

                    });

                });

                $("#content").html(`<ul>${divs}</ul>`);

            }

        }

        let obj = new ShowData('http://106.13.114.114:5000/api/firstCategory');

        obj.render();

    </script>

1. 类的私有属性和方法: 就是在类的外部通过对象直接访问、使用的属性及方法。

    <script>

        // let \_tel = Symbol('电话');

        // let \_email = Symbol('邮箱');

        // class demo {

        //     constructor(tels, emails) {

        //         this[\_tel] = tels;

        //         this[\_email] = emails;

        //     }

        //     getdata() {

        //         console.log(this[\_tel], this[\_email]);

        //     }

        // }

        // let obj = new demo(110, '110@163.com');

        // //因为Symbol的值是唯一，因此再创建一个Symbol则无法访问到属性值

        // curTel = Symbol();

        // console.log(obj[curTel])

        //**问题：一旦知道了\_tel这个变量名则可以正确访问对应的属性值**

        // console.log(obj[\_tel]);

        var mydemo = (function() {

            let \_tel = Symbol('电话');

            let \_email = Symbol('邮箱');

            class demo {

                constructor(tels, emails) {

                    // this.#address = add;

                    this[\_tel] = tels;

                    this[\_email] = emails;

                }

                getdata() {

                    console.log(this[\_tel], this[\_email]);

                }

            }

            return demo;

        })();

        // console.log(mydemo);

        let myobj = new mydemo('13078901234', ['demo@163.com');](mailto:'demo@163.com');)

**//注意：即使知道了变量名\_tel则也无法访问对应的属性值**

        // console.log(myobj[\_tel]);  // \_tel is not defined

        myobj.getdata();

    </script>

1. Set与map: set与map是es6新增的数据结构，它们对应ES5中的Array和Object数据结构。
2. 什么是set?

Set类似于数组，但与数组不同的是set中的成员没有重复的。

1. Set的使用:

Set原型提供了这些方法：add()、delete()、has()、clear()、 size

    <script>

        //set这种数据类型类似于数组，但与数组最大的区别就是set中的值是唯一的

        let arr = [1, 3, 5, 3, 5, 9];

        let curset = new Set(arr);

        //for..of

        for (var x of curset) {

            console.log(x, 'set');

        }

        //添加值

        curset.add(20);

        //删除

        curset.delete(3);

        //判断是否存在某个值

        console.log(curset.has(9));

        console.log(curset.size);

        //set用途：可以实现数组去重

        // function uniqueval(arrs) {

        //     let sets = new Set(arrs);

        //     return Array.from(sets); //再转换成数组

        // }

        var uniqueval2 = arrs => Array.from(new Set(arrs)); //再转换成数组

        console.log(curset);

        //将set转换成数组

        let arr2 = Array.from(curset);

        console.log(arr2);

    </script>

1. 什么是map?

map和object对象不一样，map可以使用forEach遍历，object不可以，这一点map的遍历形式与set、array是一致。map的key可以允许es已知的数据类型，object对象的key就是一个字符串（还有symbol）；所有综合考虑来说map 的功能相比object更强大。但是现在的开发场景下map的作用很小，因为我们请求数据都是JSON格式，JSON.stringify和JSON.parse已经与object可以有效的转换了，map暂无提供，不太方便；map目前没有提供解构赋值的做法，也不太方便。

1. map的使用：

Map原型提供了一些方法： set()、get()、delete()、has()、clear()、size

    <script>

        let obj = {};

        //  obj .3 = 'ok';  //错误的

        let curmap = new Map();

        curmap.set(3, 'yesok');

        let curobj = new Object;

        curmap.set(curobj, 'haha..')

        console.log(curmap);

        console.log(curmap.get(curobj));

    </script>