## [es6](http://es.xiecheng.live/es6/generator.html" \l "%E5%9F%BA%E6%9C%AC%E8%AF%AD%E6%B3%95)

## const/let

相同点：

1. 没有变量提升
2. 是一个块级作用域
3. 不能重复声明
4. 不会污染全局变量
5. 暂时性死区 在代码块内，使用let/const命令声明变量之前，该变量都是不可用的

注意：

const声明的变量无法被修改

声明的常量是数组、对象时可以改变数组、对象中的值

原因：const声明的常量储存的是数组、对象的地址，数组、对象是存储在堆内存中的

Object.freeze(obj1); 冻结函数，浅层次冻结，使obj1中当前这一层的属性无法修改

Object.keys():返回一个当前对象属性所组成的数组

## 模板字符串

允许嵌入表达式的字符串字面量

插入变量时使用${} `我的名字是${name}`

## 函数

### 箭头函数

一旦使用箭头函数,当前不存在作用域链,里面的this需要沿着作用域链往上找

注意

1.不能用箭头函数作为事件的回调函数

2.不能用箭头函数定义对象中的方法

3.箭头函数中无法使用arguments

4.箭头函数不能在构造函数中使用

5.箭头函数不能定义原型下的方法

const sum1 = function(x, y) {

            return x + y;

        }

//相当于

const sum1 = (x, y) => {

            return x + y;

        }

// 如果只有一行代码可以更简洁,如果返回的是对象需要加括号

const sum1 = (x, y) => x + y;

// 无参数的情况

        const fn = () => {

                // 代码

            }

// 只有一个参数的情况

        const fn1 = x => {

            // 代码

        }

### 默认值/剩余参数

参数有默认值：调用时没有传实参，使用该形参默认值，反之使用实参的值

const sum = (x, y = 3) => {

            console.log(x + y);

        }

        sum(6);

参数默认值可以为一个函数

        const f = (x, y = getVal(5)) => {

            console.log(x + y);

        }

        const getVal = (m) => {

            return m + 5;

        }

剩余参数 ...keys keys是一个存储实参的数组 解决arguments的问题

function pick(obj, ...keys)

扩展运算符... 能将数组转换为逗号分隔的参数序列

将一个数组分割，并将各个项作为分离的参数传给函数

// 1.数组合并

        const a = [1,2,3];

        const c = [4,5,6];

        const ac = [...a,...c];

        console.log(ac);

        // 2.数组的克隆   深拷贝

        const arr1 = [2,3,421];

        const arr2 = [...arr1];

        const arr3 = arr1;

        console.log(arr1 === arr2,arr1 === arr3);

        // 3.将伪数组转换为真正的数组

        let lis = document.querySelectorAll('li');

        const lit = [...lis];

        console.log(lit);

## 解构赋值

 // 解构赋值 等号两边的结构完全相等，属性名也必须一样

        const obj = {

                name: '胡歌',

                age: 43,

                teacher: {

                    name: 'xiecheng',

                    age: 56

                }

            }

// const name = obj.name;

// const age = obj.age;

// obj.name / obj.age 与obj.teacher.name obj.teacher.age属性名一样 解决方法如下

        const {

            name: allName, //起别名

            age,

            teacher: {

                name,

                id

            }

        } = obj;

        console.log(allName, age, name, id);

使用的情况

1.1数组作为参数

1.2使用对象做参数

2.1使用数组做函数返回值

2.2使用对象做函数返回值

3.两个值互换

4.解构赋值在json中的使用

5.Ajax请求

## 扩展对象的功能

// es6直接写入变量和函数，作为对象的属性和方法

        const name = '胡歌';

        const age = 89;

        const f = () => {

            console.log('hhhh');

        }

        const obj = {

            name,

            age,

            f

        }

对象的方法

Object.is(a,b) === 判断ab是否严格相等

Object.is(a,b)解决===在面对NaN时为false的特殊情况

+0 === -0 为 true ，Object.is(+0, -0) 为false

Object.assign(target,obj1,obj2...) 对象的合并 返回合并之后的新对象

## 数组的扩展方法

将伪数组转换为真正的数组

1. Array.from()

//from() 还可以接受第二 个参数，用来对每个元素进行处理

var lists = document.querySelectorAll('li');

const liContexts = Array.from(lists, els => els.textContent);

1. 给扩展运算符加上[]
2. Array.of()将任意数据类型转换为数组

Array.of(3, 12, [54, 12, 3], {id: 12 })

copywithin(a,b)

将索引为b及之后的所有元素复制，然后将复制的元素从索引为a开始覆盖，之后返回当前数组

 console.log([1, 2, 3, 4, 5, 6, 7, 8, 9].copyWithin(1, 3));

        //  [1, 4, 5, 6, 7, 8, 9,8,9]

find() 查找满足条件的第一个元素

n表示数组成员，返回的是要满足的条件

const num = [1, 2, 3, 12, 45, -6, -23].find(n => n < 0);

includes()返回一个布尔值，表示某个数组是否包含给定的值 包含返回true,不包含返回false(在之前进行判断使用indexOf(),包含返回当前索引,不包含返回-1)

findIndex()查找满足条件的第一个元素的索引

// 5.entries() keys() values() 返回一个遍历器Array Iterator 可以使用for.. .of循环进行

        const arr = ['a', 'b', 34, [1, 2]];

        console.log(arr.values());

        //keys()对键名遍历

        for (let index of arr.keys()) {

            console.log(index);

        }

        // vaLues()对值遍历

        for (let ele of arr.values()) {

            console.log(ele);

        }

        // entries()对键值对遍历

        for (let [index, ele] of arr.entries()) {

            console.log(index, ele);

        }

// 数组for..in 与for ..of区别

        // for ..of 中i是value

        const arr5 = [1, 23, 45, 's'];

        for (let i of arr5) {

            console.log(i);

        }

        // for ..in 中i是index

        for (let i in arr5) {

            console.log(i);

        }

## Symbol

// 原始数据类型Symbol，表示独一无二的值

        const name = Symbol('name');

        const name2 = Symbol('name');

        // name name2 的内存地址不同

        console.log(name === name2); //false

            // 如果用Symbol定义的对象中的变量，取值或修改时一定要用[ 变量名]

        console.log(grade[stu1])

        console.log(grade[stu2])

            // 遍历是得不到Symbol定义的对象中的属性名的

        for (let i in grade) {

            console.log(i);

        }

最大用途：用来定义对象的私有变量 用于对象的属性名，就能保证不会出现同名的属性

比如在一个班级中，可能会有同学名字相同的情况，这时候使用对象来描述学生信息的时候，如果直接使用学生姓名作为key会有有问题。

获取Symbol声明的属性名（作为对象的key）

const m = Object.getOwnPropertySymbols(grade);

获取所有的属性名

const n = Reflect.ownKeys(grade);

## set/map

### set

集合 表示无重复值的有序列表

添加元素：set.add()

删除元素：set.delete() 删除全部的元素set.clear()

校验某个值是否在set中set.has()

查看集合长度 set.size

        // 初始化的参数必须是可遍历的，可以是数组或者自定义遍历的数据结构。

        // 遍历

        set.forEach((key, val) => {

                // key 和 val 是相等的

                console.log(key);

                console.log(val);

            })

        // 将set转换为数组  扩展运算符

        const b1 = new Set([1, 2, 3, 4, 3, 3]);

        const b2 = [...b1];

        console.log(b2)

### map

Map类型是键值对的有序列表，键和值是任意类型 大部分操作与let相同

        // 定义时初始化

        let m = new Map([

            ['name', 'hhh'],

            ['age', 32]

        ])

        let map = new Map();

        // 添加键值对

        map.set('name', '张三');

        map.set('age', 23);

        // 根据key查value

        console.log(map.get('name'));

        map.set(['a', [1, 2, 3]], 'hello')

        console.log(map);

## 迭代器Interator

Interator 一种新的遍历机制 迭代就是遍历

两个核心

1.迭代器是一个接口，能快捷的访问数据，通过Symbol.iterator来创建迭代器，通过迭代器的next( )获取迭代之后的结果

2.迭代器是用于遍历数据结构的指针(数据库的游标)

        // 使用迭代

        const items = ['a', 'b', 'c'];

        // 创建一个新的迭代器

        const item = items[Symbol.iterator]();

{value: 'a',done: false} done如果为faLse表示遍历继续如果为true表示遍历完成

        console.log(item.next());

## 生成器Generator

generator函数可以通过yield关键字，将函数挂起，为了改变执行流提供了可能，同时为了做异步编程提供了方案

它与普通函数的区别

Generator 函数的定义不能使用箭头函数

1. function后面函数名之前有个\*

2. 只能在函数内部使用yield表达式，让函数挂起

函数的调用func()不会进入到func函数中，只是返回一个遍历器对象,可以调用next()

return 方法可以让 Generator 遍历终止，有点类似 for 循环的 break

当然 return 也可以传入参数，作为返回的 value 值

使用场景:为不具备Interator接口的对象提供了遍历操作

一些内置类型都是内置的可迭代类型并且有默认的迭代行为, 比如 Array or Map, 另一些类型则不是 (比如Object) 。

为了变成可迭代对象，这个对象（或者它原型链上的某个对象）必须有一个名字是 Symbol.iterator 的属性,它等于用生成器返回的迭代器

总结: generator 函数是分段执行的，yield语句是暂停执行而next()恢复执行

## Promise

Promise 承诺

相当于一个容器，保存着未来才会结束的事件(异步操作)的一个结果

各种异步操作都可以用同样的方法进行处理axios

特点:

1. 对象的状态不受外接影响处理异步操作
2. 三个状态pending( 进行) fulfilled(成功) rejected(失败)
3. 一旦状态改变，就不会再变，任何时候都可以得到这个结果

创建一个promise对象，new Promise()实参是一个回调函数，回调函数的两个参数是resolve和reject函数

resolve 和 reject 都可以传入任意类型的值作为实参，表示 Promise 对象成功（Fulfilled）和失败（Rejected）的值

then()方法

then()第一个参数是resolve回调函数，第二个参数是可选的是reject状态

then()返回新的promise实例，可以采用链式编程

.then(null,err=>{}) 相当于.catch(err=>{})

promise对象.then(f1(res),f2(err)) 即promise对象.then(f1(res)).catch(f2(err))

Promise的内置方法

1.resolve() 能将现有的任何对象转换成promise对象

let p = Promise.resolve('foo');

2.all()

let promise1 = new Promise((resolve, reject) => {});

let promise2 = new Promise((resolve, reject) => {});

let promise3 = new Promise((resolve, reject) => {});

let p4 = Promise.all([promise1, promise2, promise3]);

p4.then(() => {

// 三个都成功

}).catch(() => {

// 只要有一个失败，就失败

})

all()方法应用：一些游戏类的素材比较多， 等待图片、fLash、 静态资源文件都加载完成，才进行页面的初始化

3. race([c,d]) c,d为Promise对象

Promise.race([a(),b()]).then().catch() a,b为一个返回Promise对象的函数

在b执行前a执行完成，执行.then()，否则执行.catch()

## async

作用:使得异步操作更加方便

基本操作 async它会返回一个Promise对象 then catch

async 是Generator的一个语法糖(更易于理解阅读)

在我们处理异步的时候，比起回调函数，Promise的then方法会显得较为简洁和清晰，但是在处理多个彼此之间相互依赖的请求的时候，就会显的有些繁琐。这时候，用async/await更加优雅。

在async函数中使用await，那么await这里的代码就会变成同步的了， 意思就是说只有等await后面的Promise执行完成得到结果才会继续下去，await就是等待

await后面的Promise对象不必写then，因为await的作用之一就是获取后面Promise对象成功状态传递出来的参数。

如果async函数中有多个await那么then函数会等待所有的await指令运行完,才去执行

### generator async promise的作用

1.解决回调地域

2.使得异步操作显得更加方便

## class

 class Grand {

            // 实例化new Grand()的时候立即调用

            constructor(name, age) {

                    this.name = name;

                    this.age = age;

                }

                // 方法之间不需要','

            sayName() {

                console.log(`my name is ${this.name}`);

            }

        }

//通过Object.assign()方法一次性向类中添加多个方法

        Object.assign(Grand.prototype, {

            eat() {

                console.log('i can eat food');

            },

            run() {

                console.log('i can run fast');

            }

        })

类的继承 extends 继承属性和方法

super(name,age)会去调用父类中的constructor相当于借用构造函数

Animal.call(this,name,age);

## 模块化

模块化是指将-一个大的程序文件,拆分成许多小的文件，然后将小文件组合起来。

模块化的优势有以下几点:

1)防止命名冲突

2)代码复用

3)高维护性

// es6模块功能主要有两个命令构成: export和import

// export用于规定模块的对外接口,import用于输入其它模块提供的功能

// export 语句用于从模块中导出实时绑定的函数、对象或原始值，以便其他程序可以通过 import 语句使用它们。

//一个模块就是独立的文件

// 导出单个特性

export const name = '张三';

export const age = 12;

export const show = () => {

        console.log('hello');

    }

    //可以写成这样

    // const name = '张三';

    // const age = 12;

    // const show = () => {

    //     console.log('hello');

    // }

    // 导出列表

    // export {name,age,show};

// 不能这样抛出

//  const say = ()=>{

//      console.log('I can say');

//  }

//  export say;

// 默认导出,只能有一个

export default {

    name: '胡歌',

    say() {

        console.log('i can say')

    }

}

在导出多个值时，必须使用相应对象的相同名称。但是，可以使用任何名称导入默认导出

import m,{name,age,show} from './index.js';

(m为默认导出的别名)

抛出所有

import \* as all from './index.js';

## babel es6->es5

[babel安装与使用](https://blog.csdn.net/qq_35606400/article/details/107935280?ops_request_misc=%257B%2522request%255Fid%2522%253A%2522161382949516780265490614%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=161382949516780265490614&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_v2~rank_v29-1-107935280.pc_search_result_before_js&utm_term=babel+%E7%9A%84%E5%AE%89%E8%A3%85)

npx babel src/text.js -o dist/text.js