# 箭头函数(=>)

作用:简化函数定义

常用:

1. function (x,z,y) {方法体} 相当于 (x,z,y) => {方法体}

注: ()中表示的是参数，{}中表示的是方法体

1. () => {方法体}

注:若无参数，则()中为空

1. x => x\*x // 相当于 (x) => {return x\*x}

注:若方法体中只包含一个表达式，那么可以省略{}，且自动补上return

# 扩展运算符(…)

作用：可以将一个数组转为用逗号隔开的参数序列

常用；

1. 将一个数组push入另一个数组中

如: let a1 = [1,2,3]

let a2 = [4,5]

a1.push(…a2) // a1 = [1,2,3,4,5]

1. 复制数组

注意：数组之间不可以直接赋值，数组名只是引用

如: let a1 = [1,2,3]

let a2 = [..a1] // a2 = [1,2,3]

1. 合并数组

如: let a1 = [1,2,3]

let a2 = [3,4,5]

let a3 = [7,8]

let a4 = […a1,…a2,…a3] // a4 = [1,2,3,4,5,6,7,8]Set

# ForEach

作用: 遍历，循环，不会改变原有的函数

常用：

1. let a1 = [1,3,4,5]

a1.forEach(console.log) // 等同于 for( let k = 0;k<a1.length;k++){ console.log(a1[k]) }

1. 回调函数

如: [].forEach(function(value,index,array){

})

注: 第一个参数value是遍历的数组内容

第二个参数index是对应的索引

第三个参数array是数组本身

只跟参数所在的位置有关，跟变量名无关，并非同时都要写上这三个

# map

作用:映射，将原数组映射成新的数组，不会改变原有的函数，用法与forEach方法类似

常用:

1. 回调函数

let arr = [1,2,3]

arr.map((value,index,array) => {方法体})

注: 第一个参数value是遍历的数组内容

第二个参数index是对应的索引

第三个参数array是数组本身

只跟参数所在的位置有关，跟变量名无关，并非同时都要写上这三个

1. map()方法创建一个新的数组，其结果是该数组中的每个元素都调用一次callback后返回的结果。

如：let arr = [1,2,3]

let b = arr.map(item => item\*item) // b = [1,4,9]

# Set数据结构

含义：本身是一个构造函数，用来生成 Set 数据结构，类似于数组，但是成员值是唯一的。

常用：数组去重

例如： let arr1 = [1,2,3,4,4]

let arr2 = […new Set(arr1)] // arr2 = [1,2,3,4]

注：1.set内部判断两个数组是否相同，类似于“===”

2.两个对象总是不相等的。

3.操作方法：add(value), delete(value), has(value), clear()

# Map数据结构

含义： Map 数据结构。它类似于对象，也是键值对的集合，但是“键”的范围不限于字符串，各种类型的值（包括对象）都可以当作键。也就是说，Object 结构提供了“字符串—值”的对应，Map 结构提供了“值—值”的对应。

常用：

1. 利用set方法

let m = new Map()

m.set(1,’hei’)

m.set(2,’hello’)

m.get(1) // ’hei’

注：对同一个值进行set方法会进行覆盖

2. 利用构造函数（只要实现了Iterator 接口，都可以作为参数）

let map = new Map([

[‘name’,’zhangsan’],[‘title’,’author’]

])

注：构造函数参数格式 [[],[],[]]