## 内置对象:

Array

Array.spilt(x,x,x)接受三个参数 或者两个参数 从x开始 删除x个参数 添加进去x

Array.slice()从x开始复制到x 复制出一个新数组出来 (深拷贝)

即数组深拷贝可以直接用slice 对象深拷贝可以JSON.parse(JSON.stringify(xxx))

Array.indexof(“xx”)返回值在数组中所在的脚标位置 //如果不存在会返回-1 比较用的是===

Array.length属性可以赋值 如果小于数组现在的length则会删除数组中超过的长度的脚标

Array.isArray(xx)判断xx变量是否是数组

Array.sort(function(pre,cur))返回值如果小于0则第一个数排前面,等于0位置不变,大于0第二个数排前面

Array.concat()可以和一个或者多个数组合并 也可以添加单个元素 返回值是一个新的数组

Array.indexOf(xx)查询参数在数组中所在的角标位置;

Array.filter(function(xx,xx,xx)) 回调函数返回true才会把当前遍历到的元素加入数组 最后返回一个新数组

Array.some 一次遍历返回true 整个函数结果就是true Array.every 所有返回true 函数结果才是true

Array.map 返回一个由每次遍历的返回值构成的新数组

Array.reduce 第一个参数是一个回调函数 第二个参数是 回调函数的第一个参数的初始值

回调函数有第一个参数为每次遍历返回的值 即前一次遍历返回的值

Boolean:

注意值为false的Boolean对象也是true 因为它是一个有值的对象 它和true false不一样 包装类是一个对象 应该当做对象来看待 而不是基础数据类型

Boolean()可以当做函数使用可以把非布尔值转换成布尔值

如果用了new Boolean()则会把false也转为true 应为它代表的是生成一个新的对象

Generator:

使用规范 **function**\* idMaker() {  
 **for** (**let** i=0;i<10;i++){  
 **yield** i;  
 }  
}  
**var** generator=idMaker();

也可以是 function\* xxx(){

yield xxxx;

dosomething....

yield xxxx;

dosomething ...

}

即每次next都是继续往下执行到下一个yield标识符 并且每次都会返回yield指定的值

调用generator生成一个生成器 yield会冻结状态即不让函数继续执行 只有调用next()才会继续执行

generator的函数有

next() 继续执行下一步操作 并且返回一个对象value表示当前变量的值 done表示是否执行完毕 //当函数执行完毕返回的value则是undefined

return(xx)结束函数执行 并且返回一个给定的值

throw()抛出一个错误

RegExp:

str.replace(正则,替换内容) 可以用作字符串replace替换 默认replace是替换匹配到的第一个字符串 但是可以用正则/g进行全局替换

str.match(正则)返回字符串匹配的内容 数组形式 可以用()类似于exec捕捉指定匹配字符串

var re=正则 re.exec()可以捕获正则匹配到的字符串 返回值为第一次匹配到的字符串 可以继续往下匹配 直到返回null则表示没有匹配的内容了

var result=null;

while(resultexec(str)){

console.log(result);

}

string:

“xxx”.charAt(number)//获取第几个字符串

.indexOf(参数)获取给定参数的角标位置

.match(正则)返回字符串匹配的内容 数组形式 可以用()类似于exec捕捉指定匹配字符串

.slice(开始,结束)获取指定范围的字符串

.split(字符串)根据给定字符串分割字符串 返回值是一个新的数组

注意:所有的包装类都是object 千万不要用包装类做Boolean()比较的参数

map:

有set get delete方法 for( let entry of map) 每次遍历会返回一个键值对数组[key,value];对于key的比较是用的全等于===,保存的value只是引用并没有做拷贝处理

大多数情况下我们会用object作为map使用但是object会有原型 但是es5可以使用object.create(null);

object的key只能是字符串或者Symbols 但是map可以用任意值做键可以是对象

对象.size可以获取到键值对的个数

方法有

对象.clear()移除所有的键值对,

对象.delete()根据键移除键值对 并返回一个布尔值代表是否有该键

对象.forEach(function(value,key){})给每个遍历到的回调函数赋值key和value 最好是用for of

对象.has()判断key是否存在

对象.keys()返回所有的key组成的iterator 然后利用展开运算符...可以转换成数组 [...a.keys()]

对象.values()同上

可以使用...iterator对象展开迭代器对象 如果是键值对类型的就会变成二维数组 如果是数组类型的就会变成一维数组

同样二维数组也可以放入map的构造器中var kvArray = [["key1", "value1"], ["key2", "value2"]];var myMap = new Map(kvArray);

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

NaN需要用isNaN来判断 因为NaN不等于NaN代表一个非数字

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

数字类型函数需要记住的函数

parseFloat()将字符串解析成浮点型小数 注意 如果字符串解析默认是整数或者1.00这种 会默认解析成整数 只有1.01有效浮点型字符串才会被解析成浮点型

parseInt()则将字符串解析成整数 注意 无视字符串的小数部分

Number.isInteger()判断数字是否为整数

Number.prototype.toFixed(Number)把数字强制保留几位小数 注意返回的是字符串 而且是在数字类型的原型上的方法 意味着Number类型可以直接调用这个函数

Number(10).toFixed(2)

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

Math:

Math.random()生成一个0-1的伪随机数

Math.ceil(x)向上取整

Math.floor(x)向下取整 返回一个小于x的最大整数

Math.round()返回数字四舍五入后的整数结果

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

Object:

Object.assign(target,...source)从多个对象上复制内容到target上 注意是浅拷贝

Object.create(原型,属性构造器)第二个参数是属性构造器 可选填 例如Object.create(null,{ p: { value: 42, writable: true, enumerable: true, configurable: true }})

Object.freeze(对象);冻结该对象 浅冻结 即如果某个属性只是指向某个对象的内存地址值 即该对象还是可以改变的；

Object.getOwnPrototypeNames(对象)获取对象的所有属性包括不可枚举的 也可以作用在原型上 即返回原型的所有属性

Object.getPrototypeOf(对象)从对象上获取原型

Object.keys()返回一个包含所有key的数组

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

Promise:

基本使用

注意 一个promise只会调用resolve或者reject任意一个回调函数 即要么成功要么失败状态

var pro=new Promise(function(resolve,reject){

if(){

resolve(someValue);

}else{

reject(value);

}

})

pro.then(function(value){dosomething...}).catch(function(value){....})

--------

多个promise一起调用 注意 其中一个promise出现reject所有promise都会停止

Promise.all([...多个promise对象]).then(function(){

}).catch(function(){

})

-------

promise竞速 即只完成一个即可 当完成第一个resolve或者reject的时候promise停止

Promise.race([...多个promise对象]).then(function(){

}).catch(function(){

})

//注意 promise.then都是基于事件驱动的 所以都是异步的 编译器执行到异步代码的时候会插入列队 并不会执行 会等同步代码执行完毕后执行异步代码

-------

Promise.resolve(value) 会返回一个promise对象 promise对象.then(function..) 可以执行这个resolve 并且是异步执行

reject同样可以

而且每次then的返回值都会放入一个promise 所以可以这样写

Promise.resolve(a).then(a=>{a();**return** b}).then(b=>b());

如果需要返回reject 需要这样做

Promise.resolve(a).then(a=>{a();**return** Promise.reject(b)}).catch(b=>b());

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

解构

解构相同的数组可以解构成对应的变量

let [a,b]=[1,2,3]

a则为1b为2

let [a,b]=[1]

a为1b为undefined

如果等号右边是不可遍历的结构则会报错 map和set是具备iterator接口的 是可以遍历的结构

解构赋值允许有默认值

let [a,b=2]=[1] //注意 对应的结构位置上必须严格等于===undefined才会使用默认值

//虽然null == undefined 但是不是严格等于 所以为null不会使用默认值

a为1b为2

---------------------------------------------------------------------------------2.

## 表达式和运算符:

拓展运算符:

[....iterator]用于字面量数组中

function(...iterator) //用于函数中参数

class:

class是语法糖其实就是function(){} 一个构造函数

在里面定义的xx(){}都是给new出来的实例

里面的xx(){}方法其实都是这个实例的原型的函数属性 都会被定义到原型上

es7第一次提案里面有 xx=()=>{} 在函数里定义这样的语法 该函数会被定义到生成的实例的属性上

class xx{

constructor(){}

xx(){}

}

var xx=class{} 也可以这样定义构造器 就和匿名构造函数一样 这种如果需要在内部访问函数的话 需要给class后面加上个名字

delete:

delete xxx.xxx 删除某个对象的某个属性的操作符

in 操作符可以判断某个属性是否在某个对象中

instanceof 操作符用来判断某个对象的原型和某个构造函数的原型是否一样

new 操作符 创建一个新的实例然后通过构造函数来初始化这个实例 如果构造函数bind指向了别的对象 则new创建的新对象会覆盖掉该对象

class xx extends 某个对象 在class中要调用this.xxx时 可以调用父类的函数或者属性 但是一定要先在constructor中声明super();相当于创建super

static 在class中声明的函数前面加上static 就会把该函数或者变量声明到构造函数对象的属性上

普通声明xxx(){}是定义在原型上 xxx=()=>{}是定义在实例的属性上 static xx(){}是定义在构造函数对象上

void 表达式代表该函数不是函数声明 只是一个自执行的表达式 void function(){xxxx}() 和(function(){})()是一样的

continue; 在循环中可以使用表示结束本次循环继续下次循环 break:表示跳出本循环 或者break xxx;跳出指定循环 需要在循环外设定xxx continue也可以指定lable