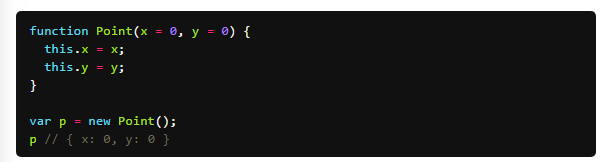
## 一、解构赋值

1、构造函数深入了解



2、typeof判断对象类型



null 不存在

undefined 存在未赋值

3、ES6为函数参数设置默认值

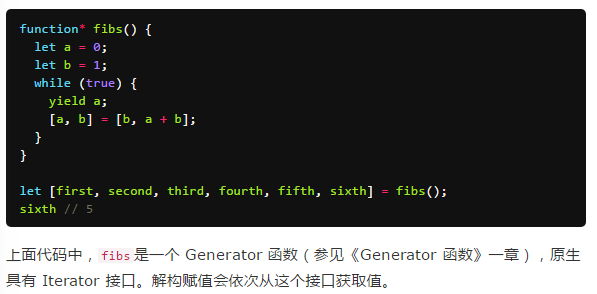


4、ES6 参数在同一个作用域中只能申明一次

5、函数参数默认值是变量，参数默认值是惰性求值



6、es6 yield的理解



7、Iterator 接口为遍历接口？

8、一个数组成员不严格等于undefined，默认值是不会生效的

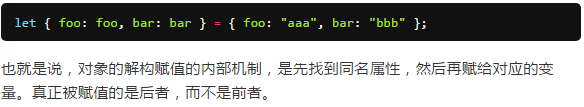


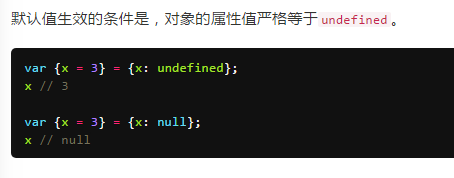
9、对象的解构赋值对应的关系

完整形式：{键名：变量，键名：变量}={属性名：属性值,属性名：属性值}

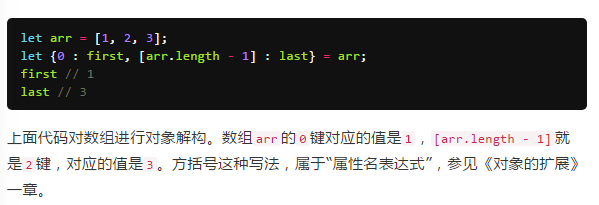
键名对应属性名，属性值赋值给变量（即键值），变量必须与属性名同名，才能模式匹配

简写形式：{变量，变量}={属性名：属性值,属性名：属性值}





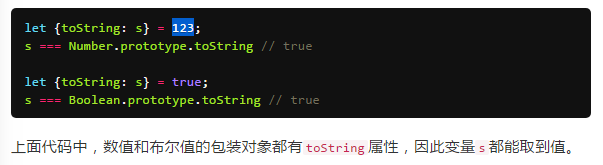
10、数组是特殊的对象？



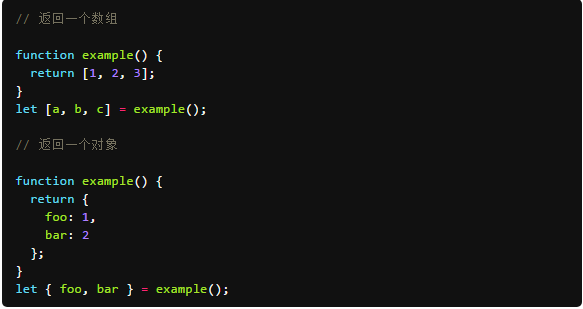
11、对象都有length属性，字符串被转成类似数组的对象，模式length匹配字符串对象的length属性



12、toString() 来把一个布尔值转换成字符串



13、函数只能返回一个值，如果要返回多个值，只能将他们放在数组或者对象里返回



14、遍历Map结构，使用for…of…循环遍历

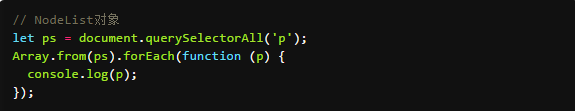


## 二、数组的扩展

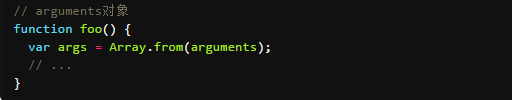
01、Array.from方法用于将两类对象转为真正的数组：类似数组的对象（array-like object）和可遍历（iterable）的对象（包括ES6新增的数据结构Set和Map）

类数组对象（必须有length属性）：

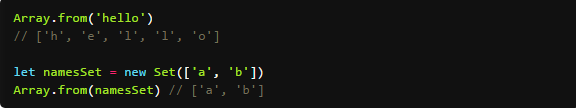
DOM操作返回的NodeList集合



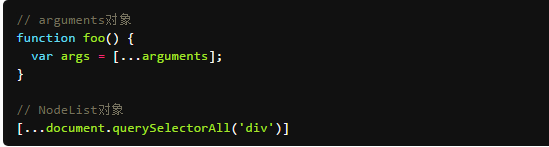
函数内部的arguments对象



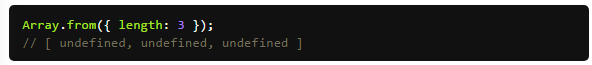
部署了Iterator接口的数据结构：



02、扩展运算符（...）也可以将某些数据结构转为数组



注：任何有length属性的对象，都可以通过Array.from方法转为数组，而此时扩展运算符就无法转换



2、Array.of总是返回参数值组成的数组

3、对象继承方法 call apply protype

4、遍历数组方法 for…of…

Es5写法：

|  |
| --- |
| for( var [key,value] of arr ){  do something;  } |

Es6写法:

|  |
| --- |
| for( let entries of arr.entries() ){  Do something;  } |

5、Set对象（不重复的数组对象）

let set = new Set();

Set内部，两个NaN是相等的，

let set = new Set();

let a = NaN;

let b = NaN;

set.add(a);

set.add(b);

set // Set {NaN}

两个对象总是不相等

let set = new Set();

set.add({});

set.size // 1

set.add({});

set.size // 2

Array.from方法可以将 Set 结构转为数组。

const items = new Set([1, 2, 3, 4, 5]);

const array = Array.from(items);

**（1）**keys()**，**values()**，**entries()

可以省略values方法，直接用for...of循环遍历 Set

let set = new Set(['red', 'green', 'blue']);

for (let x of set) {

console.log(x);

}

// red

// green

// blue

**（2）**forEach()

let set = new Set([1, 2, 3]);

set.forEach((value, key,array) => console.log(value \* 2) )

// 2

// 4

// 6

数组的map和filter方法:

let set = new Set([1, 2, 3]);

set = new Set([...set].map(x => x \* 2));

// 返回Set结构：{2, 4, 6}

let set = new Set([1, 2, 3, 4, 5]);

set = new Set([...set].filter(x => (x % 2) == 0));

// 返回Set结构：{2, 4}

使用 Set 可以很容易地实现并集（Union）、交集（Intersect）和差集（Difference）。

let a = new Set([1, 2, 3]);

let b = new Set([4, 3, 2]);

// 并集

let union = new Set([...a, ...b]);

// Set {1, 2, 3, 4}

// 交集

let intersect = new Set([...a].filter(x => b.has(x)));

// set {2, 3}

// 差集

let difference = new Set([...a].filter(x => !b.has(x)));

// Set {1}

遍历操作中，同步改变原来的 Set 结构

// 方法一

let set = new Set([1, 2, 3]);

set = new Set([...set].map(val => val \* 2));

// set的值是2, 4, 6

// 方法二

let set = new Set([1, 2, 3]);

set = new Set(Array.from(set, val => val \* 2));

// set的值是2, 4, 6

Map数据结构

const map = new Map();

map.set(['a'], 555);

map.get(['a']) // undefined

上面代码的set和get方法，表面是针对同一个键，但实际上这是两个值，内存地址是不一样的，因此get方法无法读取该键，返回undefined。

const a = ['a']; //将对象内存地址赋值给常量

const map = new Map();

map.set(a, 555);

map.get(a) // 555

这样做，引用内存地址是一样的了，就能取到值。

forEach方法还可以接受第二个参数，用来绑定this。

const reporter = {

report: function(key, value) {

console.log("Key: %s, Value: %s", key, value);

}

};

map.forEach(function(value, key, map) {

this.report(key, value);

}, reporter);

上面代码中，forEach方法的回调函数的this，就指向reporter。