for of 和 for in区别 参数本身是数组形式

一、块级作用域 1、let a=1; 用于for(){} 3、常量 const 不能改变 4、顶层对象的属性 window global(node) 没有class let const 5、

二、解构，把数组元素和对象属性解构为全局变量 1、数组解构 var[变量1，变量2]=数组 2、对象解构 （1）解构为属性同名变量 var{属性名1，属性名2}=对象 （2）另外命名的变量

var{属性：变量名}=对象 （3）默认值 （4）模式 3、字符串解构 （1）var[变量]="串" （2）

三、字符串扩展（主要是为了显示更长字节的unicode） 1、Unicode表示法 \u编码 超出两个字节\u{} 2、codePointAt(字符) 返回超过两字节的编码(十进制) 3、String.fromCodePoint(编码) 比charCode()好 4、遍历器 for of 5、charAt(索引) 识别三四字节at(索引) 6、正规化 .normalize()参数 7、是否包含子串 .includes(子串) .startsWith(子串).endsWith(子串) 8、重复 .repeat(次数) 9、填充 .padStart(总长，填充串) .padEnd() 10、模板字符串 反引号`串${变量或函数} ` 保留空格和换行，可以嵌套

11、标签模板（作为函数参数） 函数名`` 模板中的字符串和${}会分隔成参数 字符串组成一个数组 12、String.raw``

四、正则扩展 1、构造函数 new RegExp(模式或串，修饰符) 2、字符串对象的方法 匹配.match 替换.replace() 查找.search() 分隔.split（） 3、u修饰符 4、y修饰符 5、sticky 属性 是否设置了y 6、修饰符 .flags 7、s修饰符 8、后行断言 9、Unicode 属性类 10、具名组匹配 11、提案 串.matchAll（）

五、数值扩展 1、二进制0b 1010 八进制 0o 767 字符串转十进制 Number('0b1o01') 2、有限的 Number.isFinity() 是非数字 .isNaN("") 3、字符串转整数Number.parseInt("100") 字符串转为浮点数.parseFloat("") 4、是否整数 Number.isIteger(值) 5、极小常量Number.EPSILON 最小误差 6、安全整数 -2^53到2^53 超过无法精确表示 Number.MAX\_SAFE\_INTEGER和Number.MIN\_SAFE\_INTEGER 是否安全整数Number.isSafeInteger(数) 7、数字 Math静态方法 去除小数.trunc() 判断正负 .sign() 正负+0-0非数 立方根.cbrt() 二进制的零个数.clz32() 二进制乘法.imul() 多于会溢出 .fround() 平方和的平方根.hypot()

六、函数扩展 1、参数默认值 y="OK" 2、解构默认值({x=0,y}={}) 解构为变量 在函数体使用 3、参数默认值位置 4、函数长度 .length不包括默认值 5、作用域 6、rest参数 （参数数组） 函数(...rest变量){for of rest变量 } 7、name属性 8、箭头函数 （1）命名var 函数名=(参数)=>语句 （2）匿名 (参数)=>语句 （3）this指向定义时的对象（固定） （4）封装回调不变this (5)嵌套箭头函数 管道机制 λ 演算 9、提案:函数绑定运算符 对象::函数 绑定this 10、尾调用优化 11、尾递归 12、递归修改 柯里化 13、参数尾逗号

七、数组扩展 1、扩展运算符 ...[] 数组展开:把数组转变成参数组 函数(...组) 2、应用 （1）数组复制（克隆） 组2=[...组1] （2）合并 [...组,...组] （3）与解构赋值结合 （4）字符串转数组[...string] (5)可遍历

对象转数组 [...nodelist] map set 3、转成数组 Array.from(可遍历对象,回调) 4、一组值转数组Array.of(1,2,3,4) 5、数组对象的实例方法

（1）内部元素复制 .copyWithin（位置，复制起点，复制终点） （2）每个元素上应用 .find(回调（值，索引，原组）) .findIndex(回调) （3）填充.fill(值，开始，结束) （4） 实体迭代器 .entities() 值迭代器.values() 键迭代器.keys() (5)是否包含.includes(值) 6、数组空位

八、对象扩展 1、对象中写外部变量属性 var变量名=值 {变量名} （2）fuction(x,y){return{x,y }} 2、方法简写 {方法名（参数）{体} } 代替{方法名:function(){}} 3、赋值器 取值器 把属性命名为\_ 4、属性名表达式 {[表达式]:值; } 5、方法的name属性 6、是否相等 Object.is(值1，值2) 7、对象合并 Object.assign() (1）浅拷贝 （2）属性覆盖 （3）数组对象的合并 属性名 0 1 2 （4）取值函数 8、用途（1)添加属性 (2)添加方法 （3）对象克隆（浅）（4）合并对象 （5）指定默认值 9、可枚举性和遍历 for in stringify assign 返回属性名数组Object.keys(对象) Object.getOwnPropertyNames(obj) Object.getOwnPropertySymbols(obj) Reflect.ownKeys(obj) 返回所有属性的描述对象Object.getOwnPropertyDescriptors（对象） 10、\_\_proto\_\_是用来设置prototype对象的 设置原型Object.setPrototypeOf()，读取原型Object.getPrototypeOf() Object.create()

11、super指向方法（必须用简写法）所在对象的原型对象 12、属性名数组Object.keys()，属性值数组Object.values()，键值对数组Object.entries() 13、扩展运算符，{...对象} 14、解构赋值

九、Symbol类型

十、Set和Map 1、创建 new Set(数组) 2、赋值 .add(值) 3、遍历 for(变量 of s) 4、去除数组重复元素 [...new Set(array)] 5、实例属性 构造器.constructor Set 个数.size 6、实例方法 添加.add(值) 删除.delete(值) 有无.has(值) 清除.clear() 7、转为数组 Array.from(set) 8、遍历 Set键名键值一样 keys()，values()，entries() （2）.forEach(回调) （3）间接调用filter和map new Set([...set].filter(回调)) 交集并集差 集 .has() 8、WeakSet 成员只能是对象，外部不引用自动回收 .add() .delete() .has 没有size无法遍历 9、Map 键值对，键可以是任何类型 创建 new Map() new Map参数可以是数组或者Set、Map (1) 加入 .set(键,值) (2) 获取.get(键) (3)成员数.size (4) 是否存在.has（键）(5)删除.delete(键) 清除.clear() 10、遍历方法 返回遍历器对象 所有键.keys() 所有值.values() 所有实体 .entries() .forEach(回调) 11、转为数组 [...map] [...map.values] 12、map和filter() 13、数组转Map Map转对象 对象转Map Map 转JSON JSON转Map

十一、代理(拦截器，符号重载） 1、生成代理对象拦截对某个对象的访问 配置对象 var proxy= new Proxy(被代理对象，{被拦截的行为}) （2）在配置对象中重写函数（所有参数为形参，实参在使用时自动传入）形参用于函数体 2、拦截所有对象{} 作为原型 3、拦截读取操作 .和[] get:function(使用的目标对象target，属性property) 2、拦截设置赋值 set(目标，属性名，属性值) 3、拦截 存在判断 in has（目标，键） 4、拦截属性删除 delete(目标，键) 5、拦截属性名遍历（键遍历）ownKeys(目标) 6、拦截属性描述获取 getOwnPropertyDescriptor(目标，键) 7、拦截属性定义 defineProperty(目标，键，描述对象) 9、拦截原型获取 getPrototypeOf(目标) 10、拦截函数调用() .call() .apply() apply(目标，this对象，参数数组) 11、拦截实例化 constructor(目标，参数组) 12、拦截原型设置 setPrototypeOf() 14、阻止继承 15、可否继承 16、

十二、反射 1、把部分Object方法移动到Reflect 全部都是静态方法 Reflect. 1、对象属性值.get(target, name, receiver) 2、 设置对象属性 set(target, name, value, receiver) 3、是否有属性.has(target, name) 4、删除属性 .deleteProperty(target, name) 5、实例化.construct(target, args) 6、获取原形.getPrototypeOf(target) 7、设置原型 .setPrototypeOf(target, prototype) 8、绑定this并执行函数.apply(target, thisArg, args) 12、定义属性.defineProperty(target, name, desc) 13、获取属性描述 .getOwnPropertyDescriptor(target, name)14、可否继承 .isExtensible(target) 15、阻止继承 .preventExtensions(target)16、属性名遍历 .ownKeys(target)

十三、承诺对象Promise 1、状态 进行中pending 已成功fulfilled 已失败rejected 已定型resolved 2、实例 new Promise(function(resolve,reject)) resolve(value)和reject(error)是函数，作用是改变状态 2、promise.then(function(value){},function(error){}) 状态改变后调用回调 3、异步加载图片 4、Promise Ajax 5、then返回了Promise实例 链式调用.then().then() 6、.catch（function(error){}） 避免用then()第二个参数 7、.finally（function()） 8、多个Promise .all() .race

Promise包装 用匿名回调函数包裹异步过程，如文件IO，网络io，实现异步过程的顺序执行，有效地实现函数结果的 顺序传递

十四、迭代器(遍历器)和for of 1、遍历器是数据结构的接口 set array 对象 2、指针对象指向开头，指针.next()=>返回对象 属性:值value 已到尾部done:false 3、默认遍历器接口 [Symbol.iterator]:function(){return {}} 该属性值是一个函数，函数返回了一个遍历器对象 4、原生遍历器 数组 Map Set String 参数对象 节点列表 for of 5、类数组对象部署遍历器 直接引用数组的 A.原型[Symbol.iterator]=[][Symbol.iterator] 7、调用 （1）解构赋值自动调用（2）扩展运算符（3）生成器（4）数组参数 8、字符串 9、用生成器构建[Symbol.iterator]方法 10、遍历器对象 .return() .throw() 11、 for of 12、生成遍历器对象的方法 .keys() .values() .entities() 13、类数组对象转为数组 Array.from 14、普通对象实现接口 15、for in和for of

十五、生成器 异步函数，状态机、分步执行 1、定义function\* 函数名（）{yield 语句;yield 语句2;return 语句3; } 2、执行后返回一个指针对象，遍历器对象（语句值.value，是否结束.done），遇到yield就停止,返回yield后面语句的值（但是不执行） 3、 .next() 执行下一条语句 4、不用yield 5、实现遍历器接口 对象[Symbol.iterator] = function\* (){yield ;} 6、语句赋值（注入） (1).next(参数) 设置上一条语句返回值 (2)var a=yield 语句 下一句调用或return 7、for of 解构 8、throw() 9、return（） 10、yield\* 11、作为对象属性 12、this 13、状态机 while和yield的组合

十六、生成器异步应用 1、传统方法 回调 事件监听 发布订阅 Promise 2、 回调地狱 3、协程 var a=yield 语句 4、需要暂停的地方都加入yield .next() .next() 5、数据交换.next(参数) 6、错误处理 内部部署错误代码 try catch(error) 抛出错误（触发错误） .throw() 7、自动执行 Thunk函数 （1）求值策略 参数 传值 传名（2）临时函数作为参数传入fn{return 参数} 8、co模块

十七、异步函数 async acync function(){ 变量=await 语句 } 1、可以自动执行 2、返回值是Promise对象

十八、类 1、定义类 2、构造器 constructor(){ this.属性 return 对象（默认返回this）} 3、实例化 4、类表达式 5、没有提升 6、私有方法和属性 （2）提案 7、类方法this指向实例

8、name属性 9、getter setter 10、生成器 11、静态方法 12静态属性 实例属性 （2）提案

十九、类的继承

装饰函数 1、类的修饰 修饰器 @修饰函数 class{} 2、外部定义修饰函数 function 函数名(类){} 3、方法/属性修饰 @readonly 方法（）{} 4、定义 readonly(对象，属性，描述)

5、 6、混入 Mixin 7、Trait

二十、模块 1、严格模式 2、输出变量/函数/类 export{变量1，变量2，变量3} 3、加载 import {变量} from "路径/js" 4、别名 变量 as 5、整体加载 import \* as 6、export default 8.模块继承 9、跨模块常量 10import()

二十一、风格 1、let取代 var 不变量 用const 函数用const 2、静态字符串用'' 动态用反引 3、提取数组成员和对象属性 作为变量或者参数 用解构 4、多行对象属性，用逗号结尾

5、动态对象属性名 6、扩展运算符拷贝数组 类数组对象转为数组 7、立即执行函数写成箭头函数 map() 回调 事件处理都可以用箭头 8、Map 代替Object方便遍历 9、 class取代需要prototype的操作 10、ESLINT

二十二、规格 数组空位 === map()