ECMAScript6

1. babel转码器

2. let、const命令

let：只在所在代码块有效，必须先声明后使用，不能重复声明

const：声明一个只读的常量，不可改变，指向一个固定的地址，其他和let一样

这两个命令声明的全局变量不是顶层对象的属性

3. 变量的解构赋值 ——一定程度上也叫模式匹配，即找到名字相同的匹配项。

（1）用途：交换变量的值，从函数返回多个值，函数参数的定义，提取json数据，函数参数的默认值，遍历Map结构，输入模块的指定方法

（2）数组的解构赋值：let[a,b,c]=[1,2,3] let [head, ...tail] = [1, 2, 3, 4]

（3）对象的解构赋值：let{foo：baz,bar}={foo:'aaa',bar:'bbb'} 注意变量改名的用法

（4）字符串解构赋值：let{a,b,c,d,e}='hello' a=h

（5）函数参数解构赋值：

[[1, 2], [3, 4]].map(([a, b]) => a + b);

4. 字符串的扩展

（1）\uxxxx Unicode表示法也叫转义形式

（2）字符串的遍历器接口 for....of

for (let codePoint of 'foo') {

console.log(codePoint)

}

// "f"

// "o"

// "o"

（3）特殊字符串

U+005C：反斜杠（reverse solidus)

U+000D：回车（carriage return）

U+2028：行分隔符（line separator）

U+2029：段分隔符（paragraph separator）

U+000A：换行符（line feed）

（4）模板字符串

5. 字符串的新增方法

（1）

String.fromCharCode(0x20BB7) //返回码点对应的字符串

String.fromCodePoint(0x20BB7) //返回码点对应的字符串，可识别大于0XFFFFde

String.raw`Hi\n${2+3}!` //显示的是转义后的结果 "Hi\n5!",用于模板字符串的处理

/\* 下面是实例方法 \*/

(2)

let s = '𠮷a'; //比charCodeAt()好的地方：能表示超过4个字节的码点

s.codePointAt(0) // 134071

s.codePointAt(1) // 57271

s.codePointAt(2) // 97

//识别正确的32位的UTF-16字符，知道相应字符的正确位置序号

for (let ch of s) {

console.log(ch.codePointAt(0).toString(16));

}

// 20bb7

// 61

（3）.normalize（）用来将字符的不同表示方法统一为同样的形式，这称为 Unicode 正规化。

（4）

includes(‘目标字符串’，开始查找位置)：返回布尔值，表示是否找到了参数字符串。

startsWith()：返回布尔值，表示参数字符串是否在原字符串的头部。

endsWith()：返回布尔值，表示参数字符串是否在原字符串的尾部。

（5）repeat方法返回一个新字符串，表示将原字符串重复n次。

（6）如果某个字符串不够指定长度，会在头部或尾部补全。padStart()用于头部补全，padEnd()用于尾部补全。

（7）trimStart()消除字符串头部的空格，trimEnd()消除尾部的空格。它们返回的都是新字符串，不会修改原始字符串。

trim()消除两边的空格。

（8）replaceAll() 可以一次性替换所有匹配。replace()只能替换第一个匹配。

（9）at()方法接受一个整数作为参数，返回参数指定位置的字符，支持负索引（即倒数的位置）。

6. 正则的扩展

7. 数值的扩展

（1）ES6 提供了二进制和八进制数值的新的写法，分别用前缀0b（或0B）和0o（或0O）表示。

（2）数值分隔符，较长的数值允许每三位添加一个分隔符（通常是一个逗号），增加数值的可读性。比如，1000可以写作1,000。

（3）Number.isFinite()用来检查一个数值是否为有限的（finite）

Number.isNaN()用来检查一个值是否为NaN

Number.isInteger()用来判断一个数值是否为整数。

（4）Number.parseInt(), Number.parseFloat() 和原来的parseInt()等价

（5）Number.isSafeInteger()则是用来判断一个整数是否落在这个范围之内。ES6 引入了Number.MAX\_SAFE\_INTEGER和

Number.MIN\_SAFE\_INTEGER这两个常量，用来表示这个范围的上下限。

（6）Math对象的扩展

Math.trunc()方法用于去除一个数的小数部分，返回整数部分。

Math.sign方法用来判断一个数到底是正数、负数、还是零。对于非数值，会先将其转换为数值。

Math.cbrt()方法用于计算一个数的立方根。

Math.clz32()方法将参数转为 32 位无符号整数的形式，然后返回这个 32 位值里面有多少个前导 0。

Math.imul方法返回两个数以 32 位带符号整数形式相乘的结果，返回的也是一个 32 位的带符号整数。

Math.fround方法返回一个数的32位单精度浮点数形式。

Math.hypot方法返回所有参数的平方和的平方根。

(7)对数方法

Math.expm1(x)返回 e^x - 1，即Math.exp(x) - 1。

Math.log1p(x)方法返回1 + x的自然对数，即Math.log(1 + x)。如果x小于-1，返回NaN。

Math.log10(x)返回以 10 为底的x的对数。如果x小于 0，则返回 NaN。

Math.log2(x)返回以 2 为底的x的对数。如果x小于 0，则返回 NaN。

(8)双曲函数方法

(9)BigInt数据类型

8. 函数的扩展

(1)函数参数默认值

function log(x, y = 'World') {

console.log(x, y);

}

//参数y的，默认值为word

log('Hello') // Hello World

log('Hello', 'China') // Hello China

log('Hello', '') // Hello

通常情况下，定义了默认值的参数，应该是函数的尾参数。如果非尾部的参数设置默认值，实际上这个参数是没法省略的。

指定了默认值以后，函数的length属性，将返回没有指定默认值的参数个数。如果设置了默认值的参数不是尾参数，那么length属性也不再计入后面的参数了。

函数的name属性，返回该函数的函数名。

(2)函数参数默认值和解构默认赋值结合

function foo({x, y = 5} = {}) {

console.log(x, y);

}

foo() // undefined 5 可以在没有参数的情况下使用

注意：对参数默认赋值和对变量默认赋值的差异

(3)rest 参数（形式为...变量名）本身就是一个数组

(4)箭头函数 var sum = (num1, num2) => num1 + num2; sum为函数名， 箭头前(num1,num2)为参数；箭头后为为代码块部分，可以直接写返回值，若代码块的语句较多使用大括号括起来。

9.数组的扩展

(1)扩展运算符

扩展运算符是三个点（...）。它好比 rest 参数的逆运算，将一个数组转为用逗号分隔的参数序列。

应用：复制数组，合并数组，与解构赋值结合生成新的数组，将字符串转化为数组，实现了Iterator接口对象，Map和Set解构，Generater函数

(2)方法

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）。

Array.of()方法用于将一组值，转换为数组。

数组实例的copyWithin()方法，在当前数组内部，将指定位置的成员复制到其他位置（会覆盖原有成员），然后返回当前数组。也就是说，使用这个方法，会修改当前数组。

Array.prototype.copyWithin(target, start = 0, end = this.length)

//target：开始替换的位置 start：从后面第几个开始读取，再从开始替换位置替换，直到结尾

数组实例的find方法，用于找出第一个符合条件的数组成员。它的参数是一个回调函数，所有数组成员依次执行该回调函数，直到找出第一个返回值为true的成员，然后返回该成员。如果没有符合条件的成员，则返回undefined。

数组实例的

findIndex方法的用法与find方法非常类似，返回第一个符合条件的数组成员的位置，如果所有成员都不符合条件，则返回-1。

fill方法使用给定值，填充一个数组。fill方法还可以接受第二个和第三个参数，用于指定填充的起始位置和结束位置。

可以用

for...of循环进行遍历，唯一的区别是keys()是对键名的遍历、values()是对键值的遍历，entries()是对键值对的遍历。

Array.prototype.includes方法返回一个布尔值，表示某个数组是否包含给定的值，与字符串的includes方法类似。

数组的成员有时还是数组，Array.prototype.flat()用于将嵌套的数组“拉平”，变成一维的数组。该方法返回一个新数组，对原数据没有影响。

at()方法，接受一个整数作为参数，返回对应位置的成员，支持负索引。这个方法不仅可用于数组，也可用于字符串和类型数组（TypedArray）。

10.对象的扩展

(1)属性的简洁表示法

(2)属性名表达式

属性的声明有两种方法，方法一是直接用标识符作为属性名，方法二是用表达式作为属性名，这时要将表达式放在方括号之内。

(3)方法的name属性

函数的name属性，返回函数名。对象方法也是函数，因此也有name属性。

(4)属性的可枚举和遍历

Object.getOwnPropertyDescriptor方法可以获取该属性的描述对象。

属性遍历的五种方法

for ...in循环

Object.keys返回一个数组，包括对象自身的（不含继承的）所有可枚举属性（不含 Symbol 属性）的键名。

Object.getOwnPropertySymbols返回一个数组，包含对象自身的所有 Symbol 属性的键名。

Object.getOwnPropertyNames返回一个数组，包含对象自身的所有属性（不含 Symbol 属性，但是包括不可枚举属性）的键名。

Reflect.ownKeys返回一个数组，包含对象自身的（不含继承的）所有键名，不管键名是 Symbol 或字符串，也不管是否可枚举。

(5)super关键字

(6)对象的扩展运算算符（...）

(7)AggregateError 错误对象

11. 对象的新增方法

(1)Object.is比较两个值是否相等

(2)Object.assign()方法用于对象的合并，将源对象（source）的所有可枚举属性，复制到目标对象

注意，如果目标对象与源对象有同名属性，或多个源对象有同名属性，则后面的属性会覆盖前面的属性。（target）; Object.assign()方法实行的是浅拷贝，而不是深拷贝。也就是说，如果源对象某个属性的值是对象，那么目标对象拷贝得到的是这个对象的引用; Object.assign()可以用来处理数组，但是会把数组视为对象。Object.assign()只能进行值的复制，如果要复制的值是一个取值函数，那么将求值后再复制。

Object.assign()方法用途

为对象添加属性，为对象添加方法，克隆对象，合并多个对象，为属性指定默认值。

(3)Object.getOwnPropertyDescriptor()方法会返回某个对象属性的描述对象（descriptor）

(4)\_\_proto\_\_属性

\_\_proto\_\_属性（前后各两个下划线），用来读取或设置当前对象的原型对象（prototype）

Object.setPrototypeOf方法的作用与\_\_proto\_\_相同，用来设置一个对象的原型对象（prototype），返回参数对象本身。

该方法与Object.setPrototypeOf方法配套，用于读取一个对象的原型对象。

(5)

Object.keys返回一个数组，包括对象自身的（不含继承的）所有可枚举属性（不含 Symbol 属性）的键名。

Object.values方法返回一个数组，成员是参数对象自身的（不含继承的）所有可遍历（enumerable）属性的键值。

Object.entries()方法返回一个数组，成员是参数对象自身的（不含继承的）所有可遍历（enumerable）属性的键值对数组。

Object.fromEntries()方法是Object.entries()的逆操作，用于将一个键值对数组转为对象。

12. 运算符

指数运算符（\*\*)

链接判断符

NULL判断符

逻辑赋值运算符

13. Symbol：一种类似字符串的数据类型，独一无二 不能运算

(1)声明: let sym = Symbol('My symbol');

(2)sym.description

(3)作为属性名的Symbol：不能用 . 的方式来调用

Object.getOwnPropertySymbols()方法，可以获取指定对象的所有 Symbol 属性名。该方法返回一个数组，成员是当前对象的所有用作属性名的 Symbol 值。

(4)Symbol.for()方法重新使用同一个 Symbol 值 会被登记在全局环境中供搜索，Symbol()不会

Symbol.keyFor()方法返回一个已登记的 Symbol 类型值的key。

(5)内置Symbol值

对象的Symbol.hasInstance属性，指向一个内部方法。

对象的Symbol.isConcatSpreadable属性等于一个布尔值，表示该对象用于Array.prototype.concat()时，是否可以展开。

对象的Symbol.species属性，指向一个构造函数。创建衍生对象时，会使用该属性。

对象的Symbol.match属性，指向一个函数。

对象的Symbol.replace属性，指向一个方法，当该对象被String.prototype.replace方法调用时，会返回该方法的返回值。

对象的Symbol.search属性，指向一个方法

对象的Symbol.split属性，指向一个方法，

对象的Symbol.iterator属性，指向该对象的默认遍历器方法。

对象的Symbol.toPrimitive属性，指向一个方法。

对象的Symbol.toStringTag属性，指向一个方法。

对象的Symbol.unscopables属性，指向一个对象。

14. Set和Map数据结构

(1)Set 是不带重复数字的数组

声明：const set = new Set([1, 2, 3, 4, 4]);

(2)Set 结构的实例有以下属性

Set.prototype.constructor：构造函数，默认就是Set函数。

Set.prototype.size：返回Set实例的成员总数。

Set.prototype.add(value)：添加某个值，返回 Set 结构本身。

Set.prototype.delete(value)：删除某个值，返回一个布尔值，表示删除是否成功。

Set.prototype.has(value)：返回一个布尔值，表示该值是否为Set的成员。

Set.prototype.clear()：清除所有成员，没有返回值。

Set.prototype.keys()：返回键名的遍历器

Set.prototype.values()：返回键值的遍历器

Set.prototype.entries()：返回键值对的遍历器

Set.prototype.forEach()：使用回调函数遍历每个成员

(3)WeekSet结构和Set相似， 但是WeekSet的成员只能是对象，创建方式和包含的属性和Set相似

(4)Map数据结构和对象相似，对象结构提供了“字符串—值”的对应，Map 结构提供了“数据类型—值”的对应。（hash结构，常常用于哈希表）

(5)Map 结构的实例有以下属性和操作方法。

size属性返回 Map 结构的成员总数。

set方法设置键名key对应的键值为value，然后返回整个 Map 结构。

get方法读取key对应的键值，如果找不到key，返回undefined。

has方法返回一个布尔值，表示某个键是否在当前 Map 对象之中。

delete方法删除某个键，返回true。如果删除失败，返回false。

clear方法清除所有成员，没有返回值。

Map.prototype.keys()：返回键名的遍历器。

Map.prototype.values()：返回键值的遍历器。

Map.prototype.entries()：返回所有成员的遍历器。

Map.prototype.forEach()：遍历 Map 的所有成员。

(6)Map与其他数据结构的互相转化

Map转数组： Map 转为数组最方便的方法，就是使用扩展运算符（...）。

数组转Map： 将数组传入 Map 构造函数，就可以转为 Map。

Map转对象： 如果所有 Map 的键都是字符串，它可以无损地转为对象。如果有非字符串的键名，那么这个键名会被转成字符串，再作为对象的键名。

对象转Map： 对象转为 Map 可以通过Object.entries()。

Map 转为 JSON

(7)WeekMap只接受对象作为键名

15.Proxy------代理

（1）Proxy

var proxy = new Proxy(target, handler); //target：表示要拦截的目标对象 handler：拦截行为，主要是对拦截对象进行操作

get(target,propKey,receiver) //参数依次为目标对象 属性名和Proxy实例本身

get(target, propKey, receiver)：拦截对象属性的读取，比如proxy.foo和proxy['foo']。

set(target, propKey, value, receiver)：拦截对象属性的设置，比如proxy.foo = v或proxy['foo'] = v，返回一个布尔值。

has(target, propKey)：拦截propKey in proxy的操作，返回一个布尔值。

deleteProperty(target, propKey)：拦截delete proxy[propKey]的操作，返回一个布尔值。

ownKeys(target)：拦截Object.getOwnPropertyNames(proxy)、Object.getOwnPropertySymbols(proxy)、Object.keys(proxy)、for...in循环，返回一个数组。该方法返回目标对象所有自身的属性的属性名，而Object.keys()的返回结果仅包括目标对象自身的可遍历属性。

getOwnPropertyDescriptor(target, propKey)：拦截Object.getOwnPropertyDescriptor(proxy, propKey)，返回属性的描述对象。

defineProperty(target, propKey, propDesc)：拦截Object.defineProperty(proxy, propKey, propDesc）、Object.defineProperties(proxy, propDescs)，返回一个布尔值。

preventExtensions(target)：拦截Object.preventExtensions(proxy)，返回一个布尔值。

getPrototypeOf(target)：拦截Object.getPrototypeOf(proxy)，返回一个对象。

isExtensible(target)：拦截Object.isExtensible(proxy)，返回一个布尔值。

setPrototypeOf(target, proto)：拦截Object.setPrototypeOf(proxy, proto)，返回一个布尔值。如果目标对象是函数，那么还有两种额外操作可以拦截。

apply(target, object, args)：拦截 Proxy 实例作为函数调用的操作，比如proxy(...args)、proxy.call(object, ...args)、proxy.apply(...)。

construct(target, args)：拦截 Proxy 实例作为构造函数调用的操作，比如new proxy(...args)。

16.Reflect

------ 将Object对象的一些明显属于语言内部的方法（比如Object.defineProperty），放到Reflect对象上。

------修改某些Object方法的返回结果，让其变得更合理。

------ 让Object操作都变成函数行为。

------Reflect对象的方法与Proxy对象的方法一一对应，只要是Proxy对象的方法，就能在Reflect对象上找到对应的方法。

17.Promise对象 见JavaScript

18.Iterator（遍历器）和for......of循环

iterator遍历器------提供各种数据结构访问的接口（本质是一个指针对象）

iterator实现过程------创建一个指针对象，指向起始位置，使用next时指向下一位置，返回值{value：当前值，done：是否遍历结束}

function makeIterator(array) { //手写的一个遍历器生成函数

var nextIndex = 0;

return {

next: function() {

return nextIndex < array.length ?

{value: array[nextIndex++], done: false} :

{value: undefined, done: true};

}

};

}

------ Iterator 接口部署在数据结构的Symbol.iterator属性，或者说，一个数据结构只要具有Symbol.iterator属性，就可以认为是“可遍历的”（iterable）。

for......of..循环

for(let v of arr) {

console.log(v); // red green blue

}

19.Generator函数和方法（Generator函数其实是一个遍历器生成器，返回一个遍历器）也可以是一个状态机，保存内部状态

Generator函数形式：

function\* helloWorldGenerator() {

yield 'hello'; //将跟在yield后面的表达式的只作为返回值

yield 'world';

return 'ending';

}

var hw = helloWorldGenerator();

next()------方法测参数作为上一个yield表达式的返回值

Generator与协程

协程------协作的线程和协作的函数，可用单线程和多线程实现；单线程实现：只有有多个线程，其中只有一个线程处于运行状态，占用多内存为代价实现任务的并行。

Generator函数实现协程：函数运行一半遇到yield就停止，等到next()函数的调用再执行

Thunk函数：传名调用的实现函数，JavaScript中用于多参数转为单参数

20.async函数

async函数------是Generator函数的语法糖

async函数和Generator函数比较

------Generator 函数的执行必须靠执行器，所以才有了co模块，而async函数自带执行器。不需要next函数也可以执行

------async和await，比起星号和yield，语义更清楚了

------async函数的返回值是 Promise 对象，这比 Generator 函数的返回值是 Iterator 对象方便多了。你可以用then方法指定下一步的操作。

21.JavaScript中的class类

class Point {

constructor(x, y) {

this.x = x;

this.y = y;

}

toString() {

return '(' + this.x + ', ' + this.y + ')';

}

}