**ES6学习总结**

**一、let和const命令**

1、let命令只在该命令所在的代码块内有效，在该代码块外使用会报错。

2、在for循环内声明循环变量i，如果用var，则循环中只有一个全局变量i，所有的i都指向它。而用let声明，则每次循环都会重新声明，各自拥有局部变量i。而且for循环设置循环变量的那部分是一个父作用域，而循环体内部是一个单独的子作用域。

3、let声明变量不存在变量提升，区别于var

4、在一个块级作用域内，只要有let或者const存在，则它们声明的变量在声明之前都属于变量的暂时性死区，使用会报错。

5、let不允许在相同作用域内，重复声明同一个变量。不同作用域可以重复声明。

6、let声明会新增一个块级作用域，块级作用域之间

7、ES6的块级作用域必须要有大括号，没有大括号，引擎认为不存在块级作用域

8、let只能出现在当前作用域的顶层（不理解）

// 第一种写法，报错if (true) let x = 1;

// 第二种写法，不报错if (true) {let x = 1;}

1. const是声明变量指向的内存地址保存的数据不变，对于基本数据类型就相当于声明常量，对于复合类型数据（对象和数组），变量指向的内存地址保存的是一个指向实际数据的指针，所以只能保证指向数据的指针不变，但不能保证实际数据不变。

如果真的想将对象冻结，应该使用Object.freeze()方法。

1. let和const声明的对象不属于顶层对象。
2. **变量的解构赋值**
3. 数组的解构赋值

分为完全解构和不完全解构，当左右值格式一一对应则属于完全解构，不完全对应（右边的值多与左边）属于不完全解构。当右边的值少于左边时属于解构不成功，变量的值等于undefined

1. 对象的解构赋值

数组的解构需要顺序对应，而对象没有顺序，需要属性名和变量名相同才能解构成功。否则为undefined。对象的变量名和属性名不一致的话，需要写成：

let { foo: baz } = { foo: 'aaa', bar: 'bbb' };

baz // "aaa"

foo是匹配模式，匹配解构对象中的属性，而baz是变量。

对象的解构赋值可以取到继承的属性

1. 解构的默认值

数组和对象都可以给定解构的默认值。前提是被解构的数组或对象的值严格等于undefined

1. 数组可以对对象进行解构

let arr = [1, 2, 3];let {0 : first, [arr.length - 1] : last} = arr;

first // 1last // 3

1. 字符串也可以进行解构，此时字符串被转换成一个类似数组的对象
2. 数值和布尔值进行解构时都会转换为对象
3. 变量的解构赋值的用途

a、交换变量的值

let x = 1;let y = 2;

[x, y] = [y, x];

b、接收从函数返回的数组或对象

c、函数参数的定义

d、提取JSON数据，与对象解构类似

e、函数参数的默认值

f、遍历Map结构

g、输入模块的指定方法  
 const { SourceMapConsumer, SourceNode } = require("source-map");

**三、字符串的扩展**

1、for of 循环

for (let codePoint of 'foo') {

console.log(codePoint)}

// "f"// "o"// "o"

1. 模版字符串

使用一对反引号 `` 标示字符串。可以当作普通字符串，也可以定义多行字符串且所有的缩紧和格式会被保留，在模版字符串中插入变量需要将变量名写在${}中。在{}中可以放入任意的javaScript表达式。

1. **字符串的新增方法**

1、String.fromCodePoint() // 将unicode码点转换为字符

2、includes()：返回布尔值，表示是否找到了参数字符串。

3、startsWith()：返回布尔值，表示参数字符串是否在原字符串的头部。

4、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、padStart()和padEnd() 用来补全字符串

7、trimStart()和trimEnd() 用来消除字符串头和尾的空格

1. **数值的扩展**
2. 二进制用 0b做前缀，八进制用0o做前缀
3. Number.isFinite() 用来检查一个数值是否为有限的
4. Number.isNaN() 用来检查一个数值是否为NaN（Not a Number）这两个方法对传统的全局方法的区别是，全局方法会将非数值转换为数值再进行判断，而新方法只对数值有效。
5. Number.parseInt()和Number.parseFloat()将非数值类型转换为数值类型。行为与之前的全局方法完全相同。
6. Number.isInteger() 用来判断是否是一个整数
7. **函数的扩展**
8. 添加函数参数的默认值，函数参数是默认声明的，不能再次声明。如果参数默认值是一个表达式的话，每次调用都要重新计算并赋值。
9. 函数默认值可以和解构默认值一起使用，但是解构参数不能省略。

function f({ body = '', method = 'GET', headers = {} } = {}) {

console.log(method) }

1. 默认参数的位置如果不是尾参数不能省略。
2. 函数的length属性用来获取函数预期获得的参数个数，即全部参数 - 指定默认值的参数。默认参数后面的参数不参与计算。

(function (a, b = 1, c) {}).length // 1

1. 函数使用参数默认值时会形成一个单独的作用域。
2. rest参数（形式为 ...变量名）用于获取函数多余参数，而且rest参数后面不能再有参数，rest参数必须是最后一个参数。而函数的length属性也不计算rest参数。
3. 函数的name属性返回函数名，es5对于匿名函数赋值的变量读取name为空，而es6能正常读取。
4. 箭头函数
   1. 函数体内的指针是定义时所在的对象，而不是使用时所在的对象。
   2. 不可以用作构造函数，即不能用new命令
   3. 函数体内不存在arguments对象，可以用rest参数替代。

d) 定义时所在作用域和运行时所在作用域

1. **数组的扩展**
2. 为扩展运算符 ... 可以将数组拆分成参数列表  
   2、Array.from() 将类数组或者可遍历的对象转换为真正的数组
3. Array.of() 返回由参数组成的数组
4. 数组实例copyWithin() 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，此操作是修改数组。
5. 数组实例的find()和findIndex()方法，找出符合条件的成员和位置。
6. 数组实例的fill()方法用来填充一个数组
7. 数组实例的entries()、keys()、values()方法，分别对键值对
8. 数组实例的include()
9. 数组实例的flat()和flatMap()
10. sort()函数，根据传入的排序规则进行排序，若不传入，默认将数组元素转换为字符串按照unicode排序。
11. **对象的扩展**
12. 属性的简写，属性名和属性值相同时，可以只写一个。
13. 对象中的方法可以调用name属性，但是对于取值函数{getter}和存储函数{setter}
14. super关键字指向当前对象的原型对象，只能用在对象的方法中（只有简写的对象方法才能被javaScript认为是对象方法）
15. 扩展运算符配合对象解构不能拷贝继承的属性
16. ?. 运算符，链式调用，判断左侧的对象是否为null或者undefined，如果是则不继续往下计算，直接返回undefined
17. Null运算符 ?? 只有运算符左侧为null或者undefined时才会返回右侧的值
18. **对象的新增方法**
19. Object.is() 方法用来比较两个值是否严格相等，与===的区别是，===认为NaN不等于NaN，+0等于-0。而Object.is() 认为NaN与自身相等，+0不等于-0
20. Object.assign() 方法用于对象的的合并，将源对象的可枚举属性复制到目标对象。同名的属性会被后面源对象的属性覆盖。注意：此方法为浅拷贝，且不能合并继承值
21. Object.getOwnPropertyDescriptors() 返回对象自身属性的描述对象
22. Object.setPropertyOf() 设置原型对象
23. Object.getPropertyOf() 读取原型对象
24. Object.keys() 返回对象自身的所有可遍历的属性的键名
25. Object.values() 返回对象自身的所有可遍历的属性的值
26. Object.entries() 返回对象自身的所有可遍历的属性的键值对
27. Object.fromEntries() 是上一个方法的逆操作，将键值对转换为对象
28. **Promise对象**
29. Promise建立后会立刻执行
30. 回调函数将在该脚本所有同步任务执行完毕后才会执行
31. 调用resolve和reject并不会终结promise的运行，如果return的话，后面的语句就不会运行了。
32. .then() 可以运行Promise执行成功的回调
33. .catch() 可以捕捉到以上以上所有Promise对象的抛错
34. .finally() 中代码无论Promise的状态如何都可以运行。通常用来关闭流，
35. .all() 方法接收一个数组，数组的每一项必须是Promise实例，不是的话会先利用Promise.resolve() 转换。最终返回的Promise实例由传入的多个Promise实例的状态决定。只有所有传入的Promise实例状态都为fulfilled时，返回的Promise实例才是fulfilled。当有一个是rejected时，返回的Promise实例就是第一个被rejected的实例返回值。如果传入的Promise实例自己定义了catch()函数，则最终的Promise的catch()中不会捕捉到。
36. .race() 方法接收一个数组，与all相同，但是当有一个Promise实例状态改变时，最终的Promise实例的状态就会相应改变。
37. .allSettled() 函数接收一个数组，与all相同。只有当所有的Promise实例都结束的时候，最终Promise实例才会结束，而且总是返回fulfilled状态，返回值是所有Promise操作的结果组成的数组。all做不到这一点。
38. .any() 函数接收一个数组，与all相同。当有一个Promise实例变为fulfilled时，最终Promise实例就会变成fulfilled状态。如果所有的Promise实例都变为rejected时，最终Promise实例才会变成rejected。
39. resolve() 返回一个状态为fulfilled的Promise对象
40. reject() 返回一个状态为rejected的Promise对象
41. try() 方法包装函数，不论是同步还是异步都可以用Promise来处理，统一处理错误。
42. **async函数**
43. async函数返回的是一个Promise对象，async函数内部的返回值会成为then方法的回调函数的参数。
44. async返回的Promise对象要等到函数内部所有await后面的Promise对象的状态改变之后才会改变，只有遇到return和抛错（包括await后面的Promise抛错）才会终止。
45. 顶层await，在一个模块在输出时加入await命令，引用该模块的模块则会等待异步操作完成后才会运行下面的代码。
46. **Class的基本用法**
47. 类内部的所有方法都是不可枚举的。
48. **Class的继承**
49. 继承的关键字extends
50. 子类在构造函数constructor中必须调用super（指向父类super()则为调用父类的构造函数）方法，否则会报错。
51. 若子类的constructor没有显示定义，则可以不给出super()，因为super也默认隐式调用了。
52. 子类中的this关键字必须在super()方法之后使用，因为子类的构建依靠于父类的构建。
53. 父类的静态方法也会被子类继承
54. Object.getPrototypeOf()方法可以从子类上来获取父类。
55. super()函数只能用在子类构造函数里面
56. 什么是原型对象，和父类有啥区别
57. super关键字用在静态方法内是指向的是父类而不是父类的原型对象。
58. 在子类的静态方法中使用super调用父类的方法时，方法内的this指向的是当前的子类，而不是子类的实例。
59. **Module语法**
60. 在ES6模块中，在顶层this指向undefined，不应该在顶层使用undefined
61. export除了输出变量，还可以输出函数活着类（class）
62. 可以用as关键字重命名export和import都可以
63. 通过模块引入，可以取到模块内部实时的值
64. export可以出现在任何模块的任意位置，只要是模块的顶层作用域就可以，若出现在模块的块级作用域，则会报错。
65. 不允许在加载模块中修改接口，但是如果接口是一个对象则可以修改对象的属性，但是并不建议这么做。加载的模块认为是只读属性就可以。
66. 如果仅仅import某个模块，而不是引用其接口，则会运行这个模块。
67. 可以用import \* from 来引入所有的方法
68. 可以通过export default 来输出一个默认接口，这时引入模块可以指定任意的名字，但是不能用{}
69. export和import的复合写法

export { foo, bar } from 'my\_module';

这时该模块只是一个转发模块，不能使用这两个方法。

1. Import()函数是动态加载模块。当运行到那一行时才加载模块。而且该方法返回一个Promise对象，是异步方法。在then中解构接口。
2. **最新提案**
3. 管道运算符 |>将左边执行的结果向右边传送，但最多只能传送一个参数。
4. 数值分隔符 \_ 在数字的中间加入，可以增加可读性
5. Math.signbit() 判断一个值的符号为是否设置了。若没设置则返回false。表示正值，若返回true则认为是负数。
6. **装饰器**
7. @ + 函数名 放到类或者类方法的前面。