ES6读书笔记

1. **let和const变量**
2. 不存在变量提升
3. 暂时性死区
4. do表达式：块级作用域没有返回值，在块级作用域之前加上do，使它变为do表达式。

let x = do{

let t = f();

t \* t + 1;

}

此代码中，变量x会得到整个块级作用域的返回值

1. const一旦声明常量，就必须立即初始化，不能留到以后赋值
2. const实际上保证的并不是变量的值不得改动，而是变量只想的那个内存地址不得改动。对于简单类型的数据（数值、字符串、布尔值）而言，值就保存在变量指向的内存地址中，因此等同于常量，但对于符合类型的数据（主要是对象和数组）而言，变量指向的内存地址保存的只是一个指针，const只能保证这个指针是固定的，至于他指向的数据结构是不是可变的，这完全不能控制。

const foo = {};

foo.prop = 123;

foo.prop //123

foo = {}; foo指向另一个对象，就会报错

1. 将对象冻结，Object.freeze
2. **字符串的扩展**
3. javascript允许采用\uxxxx形式表示一个字符，其中xxxx表示字符的Unicode码点
4. es6提供了codePointAt方法，能正确处理四个字节储存的字符，返回字符的码点。
5. es5提供了String.fromCharCode方法，用于从码点返回对应字符，但是这个方法不能识别32位的UTF-16字符。es6提供了String.fromCodePoint方法，可以识别大于0xFFFF的字符，弥补了String.fromCharCode的不足，在作用上，正好与codePointAt方法相反。
6. formCodePoint方法定义在String对象上，二codePointAt方法定义在字符串的实例对象上。
7. 字符串的遍历器接口：es6为字符串添加了遍历器接口，使得字符串可以由for…of循环遍历。除了遍历字符串，这个遍历器最大的优点是可以识别大于0xFFFF的码点，传统的for循环无法识别这样的码点。

for(let codePoint of ‘foo’){console.log(codePonint)} //’f’ ‘o’ ‘o’

1. es5的charAt方法，返回字符串给定位置的字符，该方法不能识别码点大于oxFFFF的字符。es6提供了一个字符串实例的at方法，可以识别
2. es6为字符串实例提供了normalize方法，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。
3. includes(): 返回布尔值，表示是否找到了参数字符串
4. startsWith(): 返回布尔值，表示参数字符串是否在源字符串的头部。
5. endsWith(): 返回布尔值，表示参数字符串是否在源字符串的头部。
6. includes、startsWith、endsWith都支持第二个参数，表示开始搜索的位置，使用第二个参数n时，endsWith时针对前n个字符，而其他两个方法针对从第n个位置到字符串结束之间的字符。
7. repeat方法返回一个新字符串，表示将原字符串重复n次。如果参数为大于0的小数，会向下取整，0到-1之间的数取0。负数或者Infinity会报错。参数是字符串，会先转换成数字。
8. padStart用于头部补全。padEnd()用于尾部补全。分别接受两个参数，第一个参数用来指定字符串的最小长度，第二个参数则是用来补全的字符串，如果原字符串的长度等于或者大于指定的最小长度，则返回原字符串。
9. 如果在模板字符串中需要使用反引号，则在其前面要用反斜杠转义。

var a=`\`hello`\`world

1. es6为原生String对象提供了一个raw方法。String.raw方法往往用来充当模板字符串的处理函数，返回一个反斜线都被转义（即反斜线前面再加一个反斜线）的字符串，对应于替换变量后的模板字符串。String.raw`Hi\n${2+3}!` // “Hi\\n5!” 如果反斜线已经转义，那么String.raw不会做任何处理。
2. **正则的拓展**
3. RegExp构造函数

es5中，RegExp构造函数的参数有两种情况。

(1)参数是字符串，这时第二个参数表示正则表达式修饰符

var regex = new RegExp(‘xyz’, ‘I’);等价于 var regex = /xyz/I;

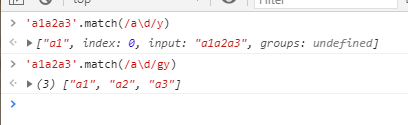
(2)参数是一个正则表达式，这时会返回一个原有正则表达式的拷贝，但此时不允许使用第二个参数添加修饰符，否则会报错。

var regex =new RegExp(/xyz/i);等价于 var regex = /xyz/I;

es6中如果RegExp构造函数第一个参数是正则对象，那么可以使用第二个参数指定修饰符，而且，返回的正则表达式会忽略原有正则表达式的修饰符，只使用指定的修饰符。 new RegExp(/abc/ig, ‘i’).flags // “i”

1. 字符串的正则方法

字符串对象共有4个方法可以使用正则表达式：match()、replace()、search()、split()。es6使这4个方法在语言内部全部调用RegExp的实例方法，从而做到所有与正则相关的方法都定义在RegExp对象上。

1. ES6对正则表达式添加了u 修饰符，含义为“Unicode模式”,用来处理大于\uFFFF的Unicode字符
2. 点（.）字符在正则表达式中的含义是除换行符以外的任意单个字符，对于码点大于0xFFFF的Unicode字符，点字符不能识别，必须加上u修饰符。
3. 除了u 修饰符，es6还未正则表达式添加了y修饰符，叫做粘连修饰符。y修饰符的作用与g修饰符类似，也是全局匹配，后一次匹配都从上一次匹配成功的下一个位置开始，不同之处在于，g修饰符只要剩余位置中存在匹配就行，而y修饰符会确保匹配必须从剩余第一个位置开始，这也就是粘连的含义。**y修饰符的设计本意就是让头部匹配的标志（^）在全局匹配中都有效。在**split方法中使用y修饰符，原字符串必须以分隔符开头，这也意味着，只要匹配成功，数组的第一个成员肯定是空字符串。 ‘##x’.split(/#/y) //[‘’,’’,’x’]。后续的分隔符只有紧跟前面的分隔符才会被识别 。 ‘#x#’.split(/#/y) //[‘’,’x#’] (#代表空格)
4. 单独的一个y修饰符对match方法只能返回第一个匹配，必须与g修饰符联用才能返回所有匹配。
5. 与y修饰符相匹配，es6的正则对象多了sticky属性，表示是否设置了y修饰符。

var y = /hello\d/y; y.sticky // true

1. es6为正则表达式新增了flags属性，会返回正则表达式的修饰符。es5的source属性返回正则表达式的正文 var y = /hello\d/y; y.source // "hello\d" y.flags //y
2. s修饰符：dotAll模式：引入/s修饰符，使得.可以匹配任意单个字符，这称为dotAll模式，即点代表一切字符，所有，正则表达式还引入了一个dotAll属性，返回一个布尔值，表示该正则表达式是否处在dotAll模式下。/foo.bar/s.test('foo\nbar') // true
3. 后行断言：先行断言指的是x只有在y前面才匹配，必须写成 **/x(?=y)/** 的形式.例如：/\d+(?=%)/.exec('100% percent')只匹配百分号之前的数字。先行否定断言指的是x只有不在y前面才匹配，必须写成 **/x(?!y)/** 的形式只匹配不在百分号之前的数字，写成 /\d+(?!%)/。后行断言与先行断言相反，x只有在y后面才匹配，必须写成**/(?<=y)x/** 的形式。 /(?<=\$)\d+/.exec('$500') //[‘500’] 匹配 美元符号后面的数字。后行否定断言：x只有不在y后面才匹配。必须写成 **/(?<!y)x/** 的形式/(?<!\$)\d+/.exec('200 $500') //[‘200’] 。 后行断言：先匹配x再回到左边匹配y的部分，‘先右后左’
4. 具名匹配：“问号 + 尖括号 + 组名”（?<year>）,然后就可以在exec方法返回结果的groups属性上引用该组名。

const RE\_DATE = /(?<year>\d{4})-(?<month>\d{2})-(?<day>\d{2})/

RE\_DATE.exec('2020-03-13').groups // {year: "2020", month: "03", day: "13"}

如果要在正则表达式内部引用某个‘具名组匹配’，可以使用\k<组名>的写法

/^(?<word>[a-z]+)\k<word>$/.test('abcabc') // true

1. **数值的扩展**
2. Number.isFinite()、Number.isNaN()

Number.isFinite()用来检查一个数值是否为有限的

Number.isNaN()用来检查一个值是否为NaN。

这两个方法与传统的全局方法isFinite()和isNaN()的却别在于，传统方法先调用Number()将非数值转为数值，再进行判断，而新方法只对数值有效，对于非数值一律返回false。Number.isNaN()只有对于NaN才返回true，非NaN一律返回false。

1. Number.parseInt()、Number.parseFloat()

es6将全局方法parseInt()和parseFloat()移植到了Number对象上，行为完全不变，这样做的目的是逐渐减少全局性的方法，使得语言逐渐模块化。

1. Number.isInteger()用来判断一个值是否为整数。需要注意的是，在JavaScript内部，整数和浮点数是同样的储存方法，所有3和3.0被视为同一个值。
2. Number.EPSILON：es6在Number对象上新增的一个极小的常量，目的在于为浮点数计算设置一个误差范围，如果这个误差能够小于Number.EPSILON，我们可以认为得到了正确结果。
3. 安全整数和Number.isSafeInteger()

es6引入Number.MAX\_SAFE\_INTEGER和Number.MIN\_SAFE\_INTEGER两个常量，用来表示-2的53次方和2的53次方范围的上下限。

Number.MAX\_SAFE\_INTEGER === Math.pow(2, 53) – 1 // true

Number.MIN\_SAFE\_INTEGER === -Number.MAX\_SAFE\_INTEGER //true

Number.isSafeInteger()则是用来判断一个整数是否落在这个范围之内

1. Math对象的扩展
   1. Math.trunc()：用于去除一个数的小数部分，返回整数部分，对于非数值，会先转换为数值。对于空值和无法截取整数的值，返回NaN。
   2. Math.sign()：判断一个数是正数、负数还是零，对于非数值，会先将其转换为数值。

其返回值有5种情况

1. 参数为整数，返回1
2. 参数为负数，返回-1
3. 参数为0，返回0
4. 参数为-0，返回-0
5. 其他值，返回NaN
   1. Math.cbrt()：用于计算一个数的立方根，对于非数值，会先将其转换为数值。
   2. Math.clz32()：JavaScript的整数使用32位二进制形式表示，Math.clz32()返回一个数的32位无符号整数形式有多个个前导0。
   3. Math.imul()：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
   4. Math.fround()：返回一个数的单精度浮点数形式
   5. Math.hypot()：返回所有参数的平方和的平方根。
   6. es6新增四个对数方法
6. Math.expml(x)返回e的x次方 – 1，即Math.exp(x) – 1
7. Math.log1p(x)方法返回ln(1+x)，即Math.log(1+x)，如果x小于-1，则返回NaN
8. Math.log10(x)返回以10为底的x的对数，如果x小于0，返回NaN。
9. Math.log2(x)返回以2为底的x的对数，如果x小于0，返回NaN。
   1. 双曲函数方法
10. Math.sinh(x)返回x的双曲正弦
11. Math.cosh(x)返回x的双曲余弦
12. Math.tanh(x)返回x的双曲正切
13. Math.asinh(x)返回x的反双曲正弦
14. Math.acosh(x)返回x的反双曲余弦
15. Math.atanh(x)返回x的反双曲正切
    1. Math.signbit()：用来判断一个数的符号位是否已经设置。（还只是个提案）
    2. 指数运算符（\*\*）： 2\*\*2 //4 2\*\*3 //8
16. **数组的扩展**
17. 扩展运算符

扩展运算符是三个点（…），她如同rest参数的逆运算，将一个数组转为用逗号分隔的参数序列 console.log(...[1,2,3]) // 1 2 3 扩展运算符提供了数组合并的新写法。 [1,2,...[3,4]] // [1, 2, 3, 4]。 扩展运算符可以将字符串转为真正的数组。

1. Array.from()

用于将两类对象转为真正的数组：类似数组的对象和可遍历对象，Array.from可以接收第二个参数，作用类似于数组的map方法，用来对每个元素进行处理，将处理后的值放入返回的数组。第三个参数可以用来绑定this。

1. Array.of()

用于将一组值转换为数组，只有当参数不少于2个时，Array()才会返回由参数组成的新数组，参数只有一个时，实际上是指定数组的长度

1. 数组实例的copyWithin()：会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，这个方法会修改当前数组。

它接收3个参数：

target（必选）：从该位置开始替换数据。

start（可选）：从该位置开始读取数据，默认为0，如果为负值，表示倒数。

end（可选）：到该位置前停止读取数据，默认等于数组长度，如果为负值，表示倒数

1. 数组实例的find()和findIndex()

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，所有成员一次执行该回调函数，知道找到第一个返回值为true的成员，然后返回该成员，如果没有找到符合条件的成员，返回undefined。

数组实例的findIndex方法返回第一个符合条件的数组成员的位置，若所有成功都不符合条件，返回-1。

1. 数组实例的fill() ： 使用给定值填充一个数组，可以接收第二个参数和第三个参数，用于指定填充的起始位置和结束位置。
2. 数组实例的entries()、keys()、values() => 对键值对的遍历、对键名的遍历、对键值的遍历。
3. 数组实例的includes() 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。该方法的第二个参数表示搜索的起始位置，默认为0，如果第二个参数是负数，则表示倒数的位置。
4. 数组的空位：forEach()、filter()、every()、some()都会跳过空位。map()会跳过空位，但会保留这个值。join()和toString()会保留空位视为undefined，而undefined和null会被处理成空字符串。

Array.from会将数组的空位转为undefined，这个方法不会忽略空位，扩展运算符（…）也会将空位转为undefined。

copyWithin会连空位一起负值。

fill()会将空位视为正常的数组位置

for…of循环也会遍历空位

entries() keys() values() find() findIndex()会将空位处理成undefined。

1. **对象的扩展**
2. Object.is()：用来比较两个值是否严格相等，与严格相等运算符（===）的行为基本一致。不同之处只有两个：一是+0不等于-0，二是NaN等于自身。
3. Object.assign()：用于将源对象的所有可枚举属性复制到目标对象
4. 属性的遍历
5. for…in 循环遍历自身和继承的可枚举属性
6. Object.keys(obj) 返回一个数组，包含对象自身的所有可枚举属性
7. Object.getOwnPropertyNames(obj) 返回一个数组，包含对象自身的所有属性，但不包含不可枚举属性。
8. Object.getOwnPropertySymbols(obj) 返回一个数组，包含对象自身所有的symbol属性。
9. Reflect.ownKeys(obj) 返回一个数组，包含对象自身的所有属性，不管属性名是symbol还是字符串，也不管是否可枚举。
10. Object.setPrototypeOf() 此方法与\_\_proto\_\_相同，用来设置一个对象的prototype对象，返回参数对象本身，它是es6正式推荐的设置原型对象的方法。
11. Object.getPrototypeOf() 用于读取一个对象的prototype
12. Object.keys() 返回一个数组，成员是参数对象自身的所有可比案例属性的键名。
13. Object.values() 返回一个数组，成员是参数自身的所有可遍历属性的键值。
14. Object.entries() 返回一个数组，成员是参数自身的所有可遍历属性的键值对数组。
15. 扩展运算符（…）
16. Object。getOwnPropertyDescriptors() 返回某个对象属性的描述对象
17. Null传导运算符 （?.） （提案， a?.b?.c?.d）
18. **Symbol**

1、