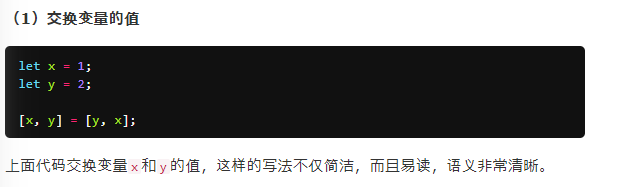
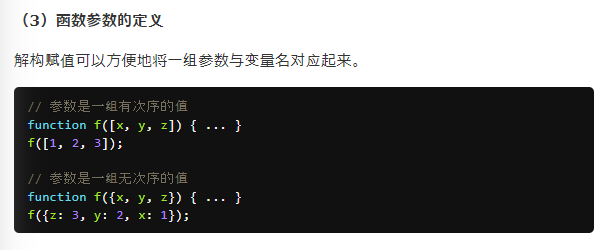
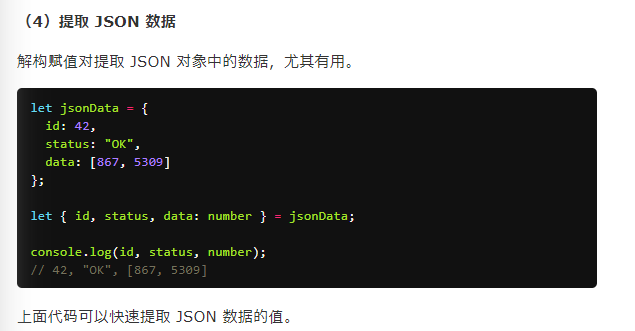
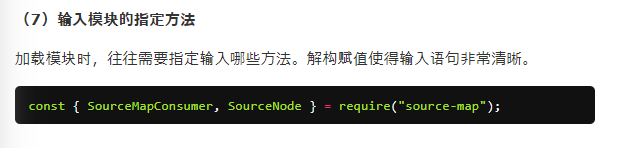
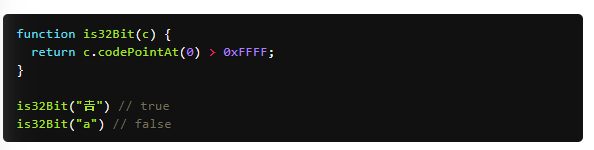
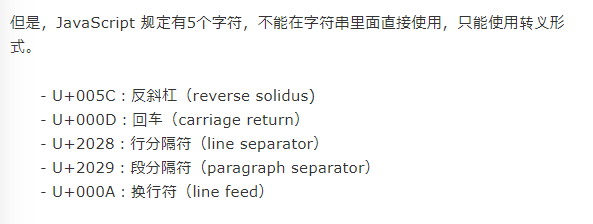
顶层对象

1. 浏览器里面，顶层对象是window，但 Node 和 Web Worker 没有window。
2. 浏览器和 Web Worker 里面，self也指向顶层对象，但是 Node 没有self。
3. Node 里面，顶层对象是global，但其他环境都不支持。
4. 综上原因 ：ES2020 在语言标准的层面，引入globalThis作为顶层对象。也就是说，任何环境下，globalThis都是存在的，都可以从它拿到顶层对象，指向全局环境下的this。

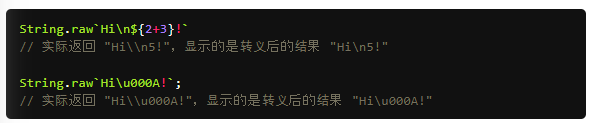
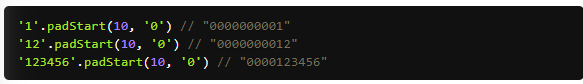
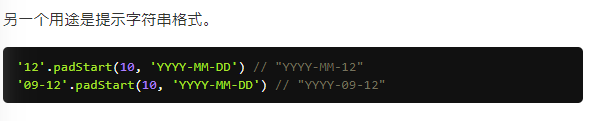
变量的解构赋值

1. 用途：
   1. 
   2. 
   3. 
   4. 
   5. 
   6. 
   7. 

字符串的拓展

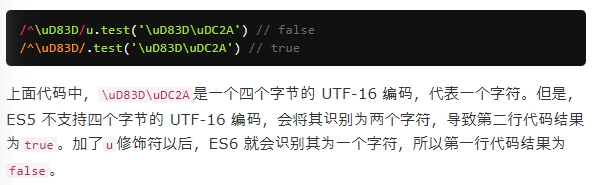
1. ES6 加强了对 Unicode 的支持，允许采用\uxxxx形式表示一个字符，其中xxxx表示字符的 Unicode 码点。
   1. 此方法无法表示码点大于\uFFFF的字符，表示大于ffff的需要拆成两个字节，或者用大括号括起来表示。如
      1. \u20BB7 = \u{20BB7}
      2. \ u20BB7 =\uD842\uDFB7
2. 字符串的for of遍历器，能遍历码点。自动识别是四个字节还是两个字节
   1. For of是用来判断一个字符是四个字节还是两个字节最简单的方法
      1. 
3. 
   1. 但是json.parse()解析时，允许u+2028 u+2029 直接使用，不需要进行转义，所以可能会报错，为了解决这个问题：
      1. ES2019 允许 JavaScript 字符串直接输入 U+2028（行分隔符）和 U+2029（段分隔符）

字符串的新增方法

1. String.fromCodePoint 因为fromcharcode不能返回大于16bits的字符，所以加入了这个方法
2. String.raw()该方法返回一个斜杠都被转义（即斜杠前面再加一个斜杠）的字符串，往往用于模板字符串的处理方法
   1. 该方法可以使用标签函数
   2. 
3. String.prototype.codePointAt() 能处理四个字节的码点，弥补charCodeAt ()的不足
   1. codePointAt()方法返回的是码点的十进制值，如果想要十六进制的值，可以使用toString(16)方法转换一下。
4. 实例方法：includes(), startsWith(), endsWith()
   1. 这三种方法都能提供第二个参数n
      1. Includes startsWitdh 表示从n的位置开始判断
      2. endsWith表示针对字符串中的前n个字符进行判断
   2. 这三种方法都返回布尔值
5. 实例方法：repeat()
   1. 表示将字符串重复n次，创建新的字符串返回。
      1. 如果是小数，会向下取整
      2. 如果是负数或者nan会报错
6. 实例方法： padStart() padEnd()
   1. 接受两个参数
      1. 需要补全到的长度
      2. 需要用来补全的字符串
   2. 如果第二个参数省略，那么默认用空格补全
   3. 用途：
      1. 为数值补全指定位数
         1. 
      2. 提示指定格式
         1. 
7. 实例方法：trimStart trimEnd
   1. 用法与trim()类似，对空格，换行，制表符，都有效
8. 实例方法：matchAll()
   1. 返回正则表达式的所有匹配的结果，match()的拓展

正则表达式的扩展

u修饰符

* 1. 加上u修饰符时，含义为“Unicode 模式”，用来正确处理大于\uFFFF的 Unicode 字符。也就是说，会正确处理四个字节的 UTF-16 编码。
  2. 
  3. 一旦加上u修饰符，很多表达式的效果会相应改变。
  4. 实例属性 ： regExp.unicode
     1. 能正确判断该正则表达式是否加了u修饰符，加了返回true。

y修饰符

* 1. 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可，而y修饰符确保匹配必须从剩余的第一个位置开始，这也就是“粘连”的涵义
  2. 实例属性：regExp.sticky
     1. 能证券判断该正则表达式是否加了y修饰符，加了返回true

1. 实例属性 ： regExp.flags
   1. 返回所有存在的修饰符
2. 实例属性 ： regExp.source
   1. 返回正则表达式本身内容

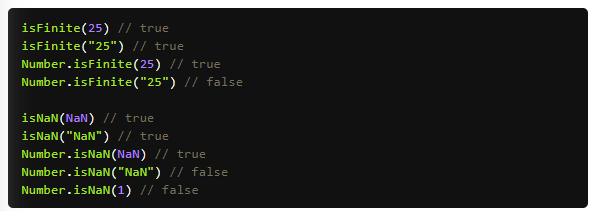
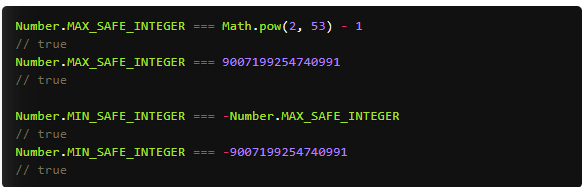
s修饰符

* 1. 正则表达式中. 代表一切字符，但是不能匹配换行符(/n)，回车符（/r）,行分隔符（line separator），段分隔符（paragraph separator）
  2. 所以es2018出了s修饰符，让.也能匹配这些字符
  3. 实例属性 ： regExp.dotAll
     1. 返回是否存在s修饰符

实例属性 ： string.mathAll()

* 1. 可以一次性取出所有匹配。不过，它返回的是一个遍历器（Iterator），而不是数组。
  2. 可以通过for of 或者…运算符或者Array.from()变成数组

数组的扩展

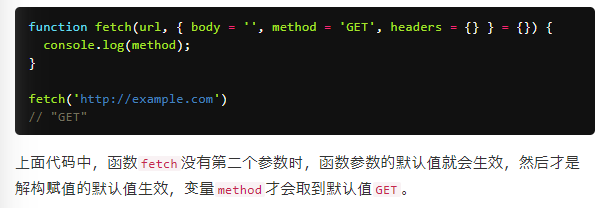
1. 0b表示二进制 0o表示八进制 0x表示十六进制
   1. 如果需要转成十进制只需要用Number(0b111)
2. Number.isfinity(num) 判断是否是有限数值，如果类型不是数值，一律false
3. Number.isNaN(num) 判断是否是NaN，如果类型不是NaN，一律false
   1. 上面两种方法与全局的同名方法的区别是直接对数值进行判断
   2. 传统的全局方法会将参数隐性转换为数值，即调用一次Number（）然后再判断。
   3. 
4. Number.parseInt === parseInt // true 完全一样，规范化
5. Number.parseFloat === parseFloat // true 完全一样，规范化
6. Number.isInteger 判断是否为整数，小数点后如果是0也是返回True，
   1. 不是数值，一律flase
   2. 精度需求很高的，位数很多的数，不建议使用
7. Number .isSafeInteger()
   1. Javascript中的整数范围为-2的53次方到2的53次方
   2. 
   3. 此函数就是用来判断一个数是否在这个范围内

ES2020引入了新的数据运算符BigInt

* 1. 。BigInt 只用来表示整数，没有位数的限制，任何位数的整数都可以精确表示。
  2. 为了与number取别，bigint必须使用n结尾
  3. 

函数的扩展

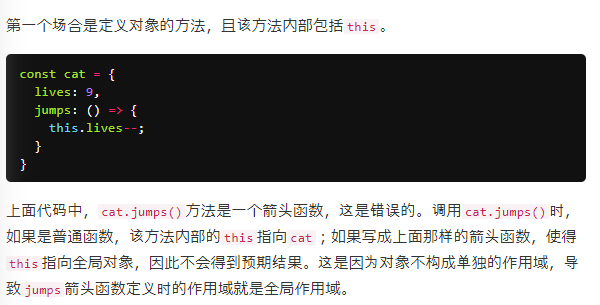
函数参数可以指定默认值

* 1. 配合解构赋值使用，尤其好用
     1. 
     2. 第二个参数的默认值生效时，就会解构空对象赋值给定义的参数对象，又因为第二个参数是空对象，所以没有值能够赋值给参数对象，就会调用定义参数中的默认值
  2. 利用默认值可以给参数创造必须赋值的条件
     1. 
     2. 函数的默认值不是在函数定义的时候执行的，还是在函数调用的时候执行。

函数的rest参数

* 1. Function(…value) …value就是数组参数
     1. Rest参数是真正的数组，能使用数组的方法，而arguments是伪数组
  2. Rest参数后面不能有其他参数，否则会报错

箭头函数

* 1. 箭头函数没有自己的this，所以箭头的this跟它所在的普通函数的this是同一个
  2. 即，箭头函数的this,指向定义时所在的对象，而不是运行时所在的对象
  3. 除了没有this.箭头函数也没有arguments, super , new target
     1. 想用arguments,可以用rest函数替代
  4. 箭头函数不适用的场所
     1. 直接在对象内定义，如
        1. 
     2. 需要利用动态this的时候
        1. 
     3. 复杂的函数体的时候，用普通函数的可读性高