```
break,continue,return区别
                                                                                 continue
                                                                                  return
                                                 stack是有结构的,每个区块按照一定次序存放,可以明确知道每个区块的大小;
                                                                                                 栈 (stack)
                                                stack创建的时候,大小是确定的,数据超过这个大小,就发生stack overflow错误
                                                                         每个线程分配一个stack,线程独占
                                                                                                              堆和栈
                                                                               内存一般储存引用数据类型
                                                                       JavaScript不允许直接访问堆内存中的位置
                                                                   heap的大小是不确定的,需要的话可以不断增加。
                                                                           每个进程分配一个heap, 线程公用
                                                                                                          i++和++i
                                               i++是先进性赋值或比较后, 自加
                                                                      如果自加语句和赋值,比较等形成一个语句,就有区别了
                                                                               转换为数字是0
                                                                       变量赋值为null可以被垃圾回收
                                                                                                    null和undefined
                                                                    是一个特殊对象,可以作为参数传递
                                                                          转换为数字是NaM
                                                                                         undefined
                                                                      只代表一个变量未被赋值
                      或者子函数在声明之外的地方被调用,子函数所在的父函数的作用域不会被释放
                                                                      闭包是指一个有权访问另一个函数作用域中的变量的函数
                                                      function foo(){
                                                       var b = 14
                                                       function fo(){
                                                        console.log(a, b)
            这里的子函数 fo 内存就存在外部作用域的引用 a, b, 所以这就会产生闭包
                                                                       函数的内部存在外部作用域的引用就会导致闭包
                       ● 假如闭包中的变量存储在栈内存中,那么栈的回收 会把处于栈顶的变量自动回收。所
                        以闭包中的变量如果处于栈中那么变量被销毁后,闭包中的变量就没有了。所以闭包引
                                                                                              闭包变量存储的位置
                                                                        闭包中的变量存储的位置是堆内存。
                        用的变量是出于堆内存中的。
                          每一个函数(包括全局和自定义函数),在词法解析阶段,都会有自己的词法作用域。当我们调
                          用一个函数的时候,若该环境没有被js回收机制回收时,则我们仍可以通过其来引用它原始的
                                                                                                       原理
                                                                              闭包的实现原理,根本上来说是作用域链
                          作用域链。
                                                                                 手动延长某些局部变量的寿命
                                                                                        函数作为参数
                                                                                       所有的回调函数
                                                                                                    应用场景
                                                                                         自执行函数
                                                                                         防抖,节流
                             我们在闭包中引用的变量,IS的回收机制不会主动的进行释放,当达到一定量后,会引起内存
解决方式:将闭包返回的那个函数设置为null
                             但其实闭包的内存泄露并不是因为闭包自身的机制,而是来源于某些浏览器针对DOM和BOM
                                                                                 会引起内存泄漏
                             对象使用的是引用计数回收、当两个对象相互引用的时候、自然就发生了计数永不为零、而永
                             远占用空间的情况。
                                                                                                      优缺点
                                                         • 保护函数的私有变量不受外部的干扰。形成不销毁的栈内存。
                                                    ● 保存,把一些函数内的值保存下来。闭包可以实现方法和属性的私有化
                        <u>indow</u>.onload必须等到页面内包括图片的所有元素加载完毕后才能执行
                                onload只执行最后一个而onready可以执行多个
                                                                             window.onload和$(document).ready()区别
                                                           $(document).ready()
            $(document).ready()是当DOM结构完成后就执行,不必等到全部元素加载完毕。
                            HTTP协议只能由客户端主动发起通信
                             websocket协议可以让服务器主动给客户端推送消息。双向通信。实现客户端与服务
                             HTTP的缺点:
                                                                                是基于HTTP协议的websocket协议
                             单向请求的特点,注定了如果服务器有连续的状态变化,客户端要获知就非常麻烦。客
                             户端只能通过轮询(每隔一段时间发送一次请求)的方式查询最新消息。这种方式效率
                             低,浪费资源
                                                                          建立在TCP协议上,比较容易实现
                                                                                  与HTTP协议兼容
                                                                      数据格式轻量, 性能开销小, 通信效率高
                                                                             可以发送文本和二进制数据
                                                                    没有同源限制,客户端和服务器可以任意通信
                                        eg: ws://example.com:80/some/path
                                                       与服务器连接: var ws = new WebSocket('ws://localhost:8080');
                                                                                                         websocket
                                                                ws.onopen连接成功后的回调,发送数据ws.send()
                                                      ws.onmessage收到数据后的回调,不需要连接的话关闭连接ws.close
                                                                            ws.onclose连接关闭后的回调
                                                                            Connection: Upgrade
                                                                            Upgrade: websocket
                                                                                             请求头变化
                                                sec-websocket-key: 客户端随机生成的字符串。
                                                sec-websocket-accetp: 服务点加密sec-websocket-key后生成的字符串
                                                       Sec-WebSocket-Protocol: 客户端和服务端商量用哪个协议
                 下一个定时器,在一定时间间隔下发送一个空包给客户端,然后客户端反馈一个同样的
                 空包回来,服务器如果在一定时间内收不到客户端发送过来的反馈包,那就只有认定说
                 掉线了。
                                                                    如果是超时断线,可以发送<mark>心跳包</mark>保活    断线问题
```

```
可以遍历对象,数组。但更适合遍历对象
                                                      for..in..取得是key/index
                                   可以遍历对象,数组
                                                      for.. in .. 中的index是string类型 ,不是num类型
                                                      for.. in .. 会把原型连上加的方法打印出来
                                                                                       可以使用hasOwnProperty来避免遍历原型
                                                      遍历顺序可能是乱序的
for..in..和for..of..
                                                     遍历可迭代对象
                                                     可以简单, 正确的遍历数组
                                   能遍历可迭代对象
                                                     不遍历原型
                                                     取的是value
                                                     break,continue,return都能正常使用
                                 箭头函数只有函数表达式, 没有函数声明式
                                 箭头函数没有constructor,没有原型链(有__proto__,但是没有prototype)
                                 箭头函数没有arguments属性
                                 箭头函数的this指向最近一层包裹它的普通函数的this。这个this有需要的属性值就
                                 用,没有拉倒,也不往上继续找。
                                 如果没有普通函数包裹,那就指向全局this(在浏览器中全局this是window,在node中
箭头函数和普通函数的区别
                                 全局this是个空对象{})。
                                                                     let obj = {
    school: "sxau",
                                                                     func: (a,b) => {
                                                                       console.log(this.school,a,b);
                                                                     let func = obj.func;
                                 call,apply,bind无法改变箭头函数中的this指向
                                                                     func.call(obj,1,2); // undefined 1 2
                                                              因为构造函数需要用new创建一个空对象,将空对象.__proto__ =构造函
                                 箭头函数为什么不能作为构造函数?
                  concat
                  slice()
数组浅拷贝
                 Array.from()
                        Student.prototype = new Person()
                                   父类型所有内容子类共享,子类可以修改掉父类的属性和方法
           原型链继承
                                   每新增一个子类就要都要操作一下它的原型链
                                   没有办法在子类自己的原型链上添加方法
                                   没有哦办法给父类传参
                                                   在子类型的构造函数内部执行
                          Person.call(this, name, age)
                                    没办法继承父类型原型链上的内容
                          缺点:
           构造函数继承
                                    每个子类实例都拷贝了一份父类方法和属性
                                    创建子类时,可以给父类传参
                                    可以时间多继承,可以call多个父类
                       Person.call(this,name,age)
                       Student.prototype = new Person()
                       Student.prototype.constructor = Student
                                 调用了两次父类构造函数,生成了两份实例
                          Person.call(this,name,age)
           混合继承优化
                          Student.prototype = Object.create(Person.prototype
                          Student.prototype.constructor = Student
           ES6继承
                              typeof 'blubber' // string
                              返回消协类型字符串
                              能判断出: number、string、 undefined、 boolean、 function
                              无法区分(都返回object): object、array、null、new创建出来的
                                                                          function Dog(name) {
    this.name = name
                                                                          let dog = new Dog('ww')
                                                                          dog instanceof Dog
                                 判断前者是否为后者的实例,比较适合判断引用类型
                                 eg: 234 instanceof Number // true
                    instanceof
                                           1.不能区分null和undefined
判断数据类型
                                 弊端:
                                                                        比如: "ggg" instanceof String // false
                                           2.基本类型的,不是实例的判断不了
                                           3.arr function obj instanceof Object 都返回true
                                  eg: let num = 233; num.constructor === Number // true
                    constructor
                                            1.不能区分null和undefined(null,undefined没有constructor属性)
                                            2.这种判断不安全 因为constructor指向可变
                                               eg: Object.prototype.toString.call(null) // "[object Null]
                                               优点: 能区分所有类型
                    Object.prototype.toString.call()
                                                         非原生实例检测不出构造函数名
                              eg: Array.isArray([1, 2, 3]); // true
                  https://juejin.cn/post/6844904056159223816
                                 Promise.then
                  微任务有哪些
eventloop
                                                   以监听一个DOM对象上发生的子节点删除、属性修改、文本内容修改等。
                                                                                                                 发。这里MO的回调是放在microtask(微任务)中执行的。
                                 mutationObsever
                                process.nextTick
                  如果在微任务执行期间微任务队列加入了新的微任务,会将新的微任务加入队列尾部
                  之后也会被执行。
                             require/module.exports
                             node的模块方案
                             同步加载
                             输出的是值的拷贝
                             运行时加载
                         require/define
                         异步模块加载
                         依赖前置
JS模块化
                        是 RequireJS 在的推广和普及过程中被创造出来。
                        就近依赖, 实现按需加载
                       异步加载
              CMD
                        是 SeaJS 在的推广和普及过程中被创造出来。
                            import/export
                            输出的是值的引用
              ES6模块化
                            编译时加载
                            前面几种都是前端社区实现的,ES6才是真正的官方出品
```