```
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..
                                                                                                                                                                                                            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
                                                                                                    break,continue,return区别
                                                                 跳出本次循环, 进入下次循环
                                                                                                                                                                                         箭头函数只有函数表达式, 没有函数声明式
                                                                                          return
                                                                                                                                                                                         箭头函数没有constructor,没有原型链(有__proto__,但是没有prototype)
                                                                     内存一般储存基础数据和函数类型,后进先出的原则。
                                                                                                                                                                                         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
                                                                                                          栈(stack)
                                                                                                                                                                                         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
                                                                                每个线程分配一个stack,线程独占
                                                                                                                                                            箭头函数和普通函数的区别
                                                                                                                                                                                         全局this是个空对象{})。
                                                                                                                        堆和栈
                                                                                       内存一般储存引用数据类型
                                                                                                                                                                                                                          let obj = {
    school: "sxau",
                                                                                                                                                                                                                           func: (a,b) => {
                                                                              JavaScript不允许直接访问堆内存中的位置
                                                                                                                                                                                                                            console.log(this.school,a,b):
                                                        heap是没有结构的,数据可以任意存放。因此,stack的寻址速度要快于heap。
                                                                                                                                                                                                                           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
                                                                                  每个进程分配一个heap,线程公用
                                                                                                                                                                                                                    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
                                                                                                                    i++和++i
                                                                                                                                                                           concat
                                                                             如果自加语句和赋值, 比较等形成一个语句, 就有区别了
                                                                                                                                                                           slice()
                                                                                                                                                           数组浅拷贝
                                                                                                                                                                           Array.from()
                                                                                       转换为数字是0
                                                                             变量赋值为null可以被垃圾回收
                                                                                                                                                                                  Student.prototype = new Person()
                                                                                                              null和undefined
                                                                           是一个特殊对象,可以作为参数传递
                                                                                                                                                                                           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
                                                                             只代表一个变量未被赋值
                                                                                                                                                                                           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
                                                           var a = 0
                                                                                                                                                                                             没办法继承父类型原型链上的内容
                                                           function foo(){
                                                            var b = 14
                                                                                                                                                                     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
                                                             function fo(){
                                                                                                                                                                                             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
                                                                                                              形成原因
                                                                              函数的内部存在外部作用域的引用就会导致闭包。
                                                                                                                                                                                 Person.call(this,name,age)
                          用的变量是出于堆内存中的
                                                                               闭包中的变量存储的位置是堆内存。
                             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
                                                                                                                                                                                          调用了两次父类构造函数,生成了两份实例
                             用一个函数的时候,若该环境没有被is回收机制回收时,则我们仍可以通过其来引用它原始的
                                                                                     闭包的实现原理, 根本上来说是作用域链
                                                                                                                                                                                   Person.call(this,name,age)
                                                                                                                          闭包
                                                                                                                                                                     混合继承优化
                                                                                                                                                                                   Student.prototype = Object.create(Person.prototype)
                                                                                         手动延长某些局部变量的寿命
                                                                                                                                                                                   Student.prototype.constructor = Student
                                                                                                 函数作为参数
                                                                                                                                                                                extends
                                                                                                              应用场景
                                                                                                                                        js基础
                                                                                               所有的回调函数
                                                                                                                                                                     ES6继承
                                                                                                  防抖,节流
                                                                                                  自执行函数
                                                                                                                                                                                       typeof 'blubber' // string
                                我们在闭包中引用的变量,JS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
                                                                                                                                                                                       返回消协类型字符串
                               泄漏
                                                                                                                                                                                       能判断出: number、string、 undefined、 boolean、 function
                                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
                                                                                                                                                                                       无法区分(都返回object): object、array、null、new创建出来的
                               远占用空间的情况
                                                                                                               优缺点
                                                                                                                                                                                                                                 function Dog(name){
                                                              • 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。
                                                                                                                                                                                                                                    this.name = name
                                                                                                      优点:
                                                         ● 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
                                                                                                                                                                                                                                 let dog = new Dog('ww'
                                                                                                              参考链接
                                                                                                                                                                                                                                 dog instanceof Dog
                                                                                                                                                                                          判断后者是否在前者的原型链上,比较适合判断引用类型
                                                                                                                                                                                         eg: 234 instanceof Number // true
                            window.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
                                                                                                                                                                             instanceof
                                                                                                                                                                                                   1.不能区分null和undefined
                                                                                        window.onload和$(document).ready()区别
                                                                                                                                                            判断数据类型
                                                                                                                                                                                                   2.基本类型的,不是实例的判断不了
                                                                                                                                                                                                                              比如: "ggg" instanceof String // false
                                                        jquery的语法
                                                                      $(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
                                                                                                                                                                                                   3.arr function obj instanceof Object 都返回true
                                                                                                                                                                                          eg: let num = 233; num.constructor === Number // true
                                                1、A 页面打开 B页面, A页面可以从window.open返回值, 拿到B页面的window
                                                                                                                                                                                                   1.不能区分null和undefined(null,undefined没有constructor属性)
                                               2、A页面发送 message 给B页面
                                                                                                                                                                                                    2.这种判断不安全 因为constructor指向可变
                                                3、B页面接收到A页面发来的 message , 并从中获取到A页面的window
                                                                                                                                                                                                       eg: Object.prototype.toString.call(null) // "[object Null]"
                                                                                                                                                                                                       优点: 能区分所有类型
                                               4、此时A页面有B页面的window,B页面也有A页面的window,只要双方都监听
                                                                                                                                                                             Object.prototype.toString.call()
                                                                                                       原理
                                                message 事件,就可以双向通信了
                                                                                                                 postmessage
                                                                                         父子页面通信
                                                                                                      iframe
                                                                                                                                                                                       eg: Array.isArray([1, 2, 3]); // true
                                                                                                                                                                            https://juejin.cn/post/6844904056159223816
                                HTTP协议只能由客户端主动发起通信
                                                                                                                                                                                         Promise.then
                               websocket协议可以让服务器主动给客户端推送消息。双向通信。实现客户端与服务器
                               的平等对话
                                                                                                                                                                                                                                                                  调用过程是要先给它绑定回调,得到MO实例,这个回调会在MO实例监听到变动时触
                                                                                                                                                                                                          MutationObserver: MO是HTML5中的API, 是一个用于监视DOM变动的接口, 它可
                                                                                                                                                                            微任务有哪些
                                                                                                                                                           eventloop
                                                                                                                                                                                                                                                                  发。这里MO的回调是放在microtask(微任务)中执行的。
                                                                                                                                                                                                          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
                                                                                                                                                                                         mutationObsever
                               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
                                                                                                                                                                                         process.nextTick
                                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
                                                                                                                                                                            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
                               低,浪费资源
                                                                                                                                                                            之后也会被执行。
                                                                                 建立在TCP协议上,比较容易实现
                                                                                          与HTTP协议兼容
                                                                                                                                                                                      require/module.exports
                                                                            数据格式轻量,性能开销小,通信效率高
                                                                                                                                                                                      node的模块方案
                                                                                     可以发送文本和二进制数据
                                                                                                                                                                                      同步加载
                                                                                                                                                                                                文件会需要很长时间
                                                                          没有同源限制,客户端和服务器可以任意通信
                                                                                                                                                                                      输出的是值的拷贝
                                           eg: ws://example.com:80/some/path
                                                                                                                                                                                      运行时加载
                                                            与服务器连接: var ws = new WebSocket('ws://localhost:8080');
                                                                                                                   websocket
                                                                                                                                                                                  require/define
                                                                      ws.onopen连接成功后的回调,发送数据ws.send()
                                                                                                                                                                                  异步模块加载
                                                           ws.onmessage收到数据后的回调,不需要连接的话关闭连接ws.close
                                                                                                                                                                                  依赖前置
                                                                                   ws.onclose连接关闭后的回调
                                                                                                                                                                                 是 RequireJS 在的推广和普及过程中被创造出来。
                                                                                   Connection: Upgrade
                                                                                                                                                                                  就近依赖,实现按需加载
                                                                                   Upgrade: websocket
                                                                                                                                                           JS模块化
                                                                                                      请求头变化
                                                                                                                                                                         CMD
                                                                                                                                                                                 异步加载
                                                    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
                                                                                                                                                                                  是 SeaJS 在的推广和普及过程中被创造出来。
                                                             Sec-WebSocket-Protocol: 客户端和服务端商量用哪个协议
                                                                                                                                                                                     import/export
                                                                                                                                                                                     输出的是值的引用
                  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
                                                                                                                                                                                     编译时加载
                   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
                                                                                                                                                                                     前面几种都是前端社区实现的,ES6才是真正的官方出品
                                                                                                                                                                                                                      在浏览器中,可以直接使用 <script> 标签通过指定 type 为 module 加载一个ESM。
                                                                                                                                                                                                                       <script type="module" src="main.mjs"></script> 在浏览器中加载 main.mjs文件
                                                                                                                                                                                                                                                                           可以识别 type="module" 的浏览器会忽略带有 nomodule 属性的 script 标签,可以通过
                                                                                                                                                                                                                       <script nomodule src="fallback.js"></script> 优雅降级
                                                                                                                                                                                                                                                                          这种方式为不支持 ESM 的浏览器做降级处理。
                                                                                                                                                                         ES6模块化
                                                                                                                                                                                                                       如果浏览器支持 ESM, 这也意味着浏览器支持其他的 ES6 语法,像 Promise,
                                                                                                                                                                                                                      async-await,可以通过这种方法减少对代码中 ES6 等高级语法的编译,减少代码打
                                                                                                                                                                                                                      包体积,提升加载速度和 JS 代码的执行速度,提高页面性能,然后对不支持 ESM 的
                                                                                                                                                                                                                       浏览器提供编译后的版本,做降级处理
                                                                                                                                                                                     既可以在服务器端使用,也可以在浏览器中使用
                                                                                                                                                                                                                                                                               defer 与 async 的区别是: defer 要等到整个页面在内存中正常渲染结束 ( DOM 结构完全
                                                                                                                                                                                                                                                                               生成,以及其他脚本执行完成),才会执行; async 一旦下载完,渲染引擎就会中断渲染,
                                                                                                                                                                                                                                                                               执行这个脚本以后,再继续渲染。一句话,defer 是"渲染完再执行",async 是"下载完
                                                                                                                                                                                                                      默认的 <script> 标签加载资源会阻塞 HTML 解析,可以通过 defer 和 async 属性来让 JS
                                                                                                                                                                                                                                                                               就执行"下载是异步的。另外,如果有多个 defer 脚本,会按照它们在页面出现的顺序加
```

可以遍历对象,数组。但更适合遍历对象

脚本异步加载。

ESM 在浏览器中会有 CORS 跨域问题,所有跨域的 ESM 资源加载都需要在资源响应头上添加 Access-Control-Allow-Origin 的响应头,而在之前的 js 资源加载上是不需要的。

载,而多个 async 脚本是不能保证加载顺序的。

for..in..取得是key/index