1.javaScript:负责网页功能的

ECMAScript(核心) + DOM(页面操作) + BOM(浏览器操作相关的)

script需要放在body的结束标签之前

2.编程语言

c： 操作系统、嵌入式、驱动开发

c++： 桌面软件、游戏 (英雄联盟)

c#： Windows桌面软件、.NET Web、服务器

java:企业级应用，web开发、服务器后端

python

php

javaScript

3.程序员

变量：内存中专门用来存储数据的空间

程序: 处理数据的　接受数据--处理数据--输出数据　程序运行在内存中

数据：语言啊 阿拉伯数字

4.变量

临时存储数据的

何时使用:如果数据需要临时存储的时候，那么就需要变量

关键字：程序中特殊含义的字符

如何使用:

1.声明变量(var)　2.命名 (不能用关键字，不能用中文，不能用特殊符号，见名知意，不能用纯数字，不能用数字打头，可以大写)　3.初始化

2.使用变量就相当于使用变量里面的数据

3.一个变量只能存一个数据

4.变量是可以更改的　变量名=新值

5.对于未声明的变量直接赋值，那么js会自动在全局声明

6.变量声明提升：变量声明语句会自动提升到当前作用域最顶部

7.等号左边一定是变量，等号右边一定是数据或者表达式(结果是个数据)

5.控制台

是一个可以输出js代码的地方

作用：1.用来调试的　2.提示错误　3.扯淡

控制台不报错不代表没有错，控制台报错了，不一定代表是所标注的错误，但是一定代表有i错

console.log(要输出的数据)

alert()

作用：调试的 提示/警告 会中断程序运行

6.数据类型

原始数据类型

1.Number(数字类型)

2.String(字符串) 字符或者字符与其他数据的组合

字符串一定要加引号，加引号的数据一定是字符串

如果引号嵌套的情况，双引号里面可以放单引号，单引号里面不允许放双引号

变量名不能加引号

3.Boolean(布尔类型) true false

4.undefined(未定义) 用来自动初始化变量的

5.null（空） 用来主动释放对象的

引用数据类型

1.array(数组)

2.object(对象) 函数也是对象

区别：

原始数据类型：数据存在变量本地 栈

引用数据类型：数据不存在变量本地 堆

栈 /堆：本质上是内存中的一块存储空间

7.运算符

表达式：由运算符连接的，最终运算结果是一个值的式子

表达式跟值是等效的

程序模拟人类进行计算的符号

算术运算符：+ - * / % ++ -- 仅适用于number类型的数据

关于++，如果单独使用，放前放后都可以

如果不是单独使用，后++，先用旧值参与表达式，表达式结束之后再+1

前++，先+1，再参与表达式

关系运算符: > < >= <= == === != !==

不允许连着写

逻辑运算符：与（&&） 或（||） 非 （！）

关于逻辑运算符的返回值

> 1、只要“||”前面为false，无论“||”后面是true还是false，结果都返回“||”后面的值。
> 2、只要“||”前面为true，无论“||”后面是true还是false，结果都返回“||”前面的值。
> 3、只要“&&”前面是false，无论“&&”后面是true还是false，结果都将返“&&”前面的值;
> 4、只要“&&”前面是true，无论“&&”后面是true还是false，结果都将返“&&”后面的值;

> 需要说明的是“&&”的优先级是高于“||”的，下面测试：

```
console.log(1||'a'&&2);  //这段代码返回的是1
```

赋值运算符：= += -= *= /= %=

字符串连接运算符：+ 任何数据与字符串拼接，结果都是字符串

三目(元)运算符： 条件?条件成立时候的值:条件不成立时候的值

typeof():用来检测数据类型

8.语句

if else 可以进行多重条件判断

if(条件){条件成立时候执行的代码}else{条件不成立时候执行的代码 }

以下六种情况都算false

false 0 undefined null "" NaN。

9.隐式类型转换

js中的数据会根据具体情况自动改变数据的类型。

10.函数/方法

什么情况下用函数：如果一段代码要反复调用，那么就考虑封装成函数了

封装一段执行专门任务的代码段。

函数是不调用不执行的

function 函数名(参数){

代码段//想干的事

}

var 函数名=function(参数){代码段}

函数的调用:函数名()

参数：函数内独有的变量，接受函数外的数据，在函数内部处理，参数可以让方法更灵活

形参：形式上的参数

实参：实际上的参数

参数不限制数量，不限制数据类型，多个参数之间以逗号隔开就行

函数提升：整体提前

return关键字：

函数是一个纯过程，没有任何结果。

如果函数的执行你需要一个结果，可以加return关键字

return 你想要的值 函数的结果就是return后面的表达式

return的本意其实是退出函数的运行，如果return后面有值的话，那么会在退出的同时，返回一个结果

11.作用域

一个变量可用的范围

全局作用域：函数外，全局不能访问局部的数据

局部作用域：函数内，局部内可以访问全局的数据

函数调用的时候才创建，调用结束之后立即销毁

局部内要更改某个数据，优先用局部内的，局部内没有会往外层找

全局变量：在全局作用域内声明的变量叫全局变量

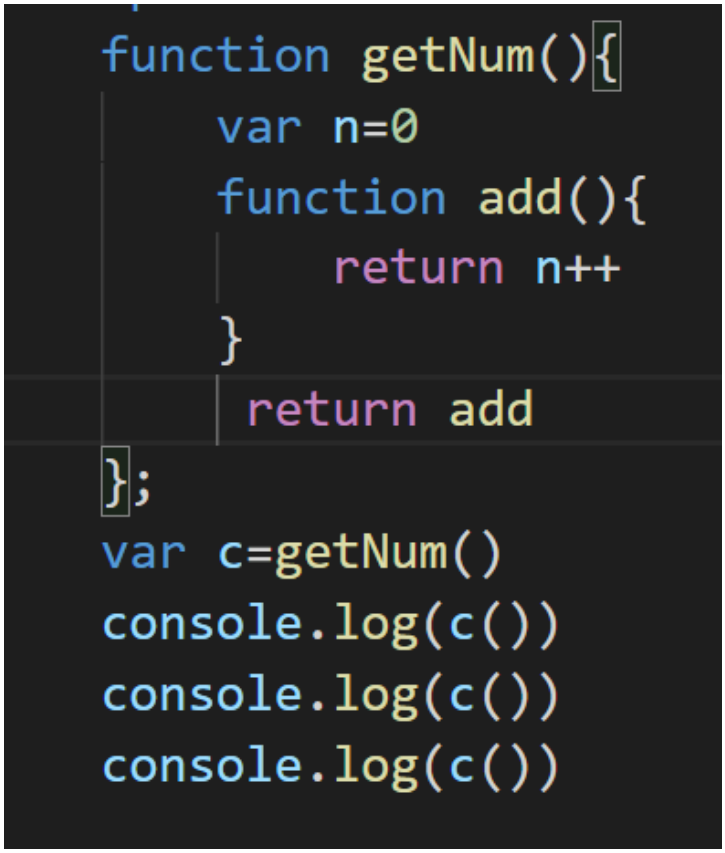
局部变量：在局部作用域内声明的变量叫局部变量

12.闭包

概念：函数使用了不属于自己的局部变量，这种结构叫闭包（函数套函数）

作用：保护变量的/避免全局污染

性能问题：内存泄漏　作用域中的局部变量一直被使用着，导致该作用域释放不掉



```
function a(){
  var name = 'dov';
  return function(){
    return name;
  }
}
var b=a();
console.log(b()) //dov
```



```
function fn(){
    var num = 3
    return function(){
        var n = 0;
        console.log(++n)
        console.log(++num)
    }
}
var fn1=fn()
fn1() //1 4
fn1() //1 5
```

13.循环语句

循环：程序反复执行一套相同的代码

循环三要素：

1.循环变量:循环中做判断的量 循环变量一定是向着循环退出的趋势去变化

2.循环条件：保证循环继续运行的条件

3.循环体：循环中每次要做的事

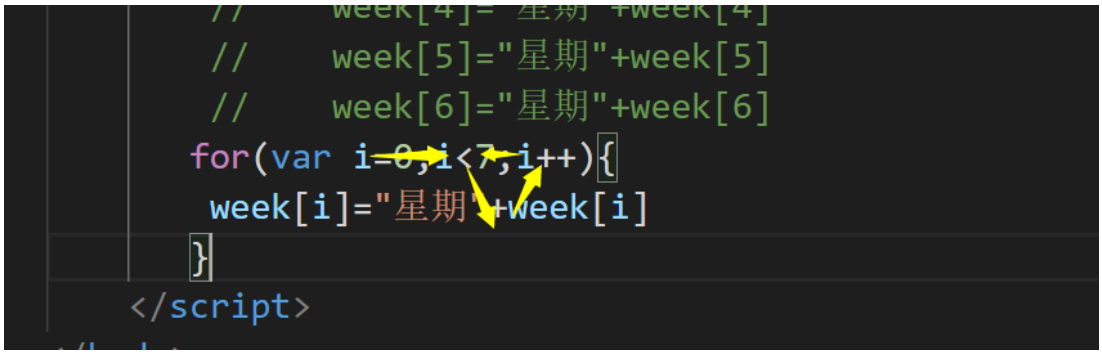
while循环

while(循环条件){要做的事}

for循环

用途：用作数组遍历

for(var i=0;i<10;i++){要做的事}



14.数组

数组：批量存储多个同类数据的，多个数据以逗号隔开，数组是没有任何数据类型限制的也没有任何数量限制的。

数组其实就相当于多个变量的集合

数组的访问：数组名[角标]

数组的更改 ： 数组名[角标]=新值

数组的属性：length 直接返回数组的长度

15对象

对象：用来存储多个数据的 是由多个键值对组成的 用来描述一个事物的

相当于多个变量的集合

格式 {key:value,key:value} 键/值对 属性名：属性值

对象的属性值是不限制数据类型的

对象的属性名一定是字符串，所以属性名可以省略引号，如果不加，js会自动帮你添加

对象的访问：对象.属性名

对象的更改：对象.属性名=新值 如果本身存在这个属性就是更改，本身如果没有，那就是添加

对象的属性的删除 delete 对象.属性

对象的循环

```
for(var 变量名 in 要遍历的对象){
    变量名代表的是属性
}
```

对象的属性名如果是变量的话，那么需要加 [ ]

16.内置对象

面向对象：通过操作对象去实现需求，不关心其中的过程

面向过程：

内置对象：js中已经存在的，有着现成的属性和方法供我们使用

js中一共26个内置对象

自定义对象：我们自己创建的对象

基本包装类型：为了便于操作“基本类型值”，JS 提供了 三个 特殊的引用类型：Boolean、Number、String。这些类型和其他引用类型相似，但同时 也具备 与各自基本类型相应的特殊行为。 实际上：每当读取一个基本类型值的时候，“后台就会创建一个 对应的基本包装类型的对象”，从能能够调用一些方法来操作这些数据。

1.String对象　　字符串是不容更改的

length

toUpperCase(转大写)

toLowerCase(转小写)

substring(截取子字符串) 含头不含尾

　　如果只给一个参数，代表从哪一位开始截取，截到最后

slice()同上

indexOf(查找关键字) 关键字符的角标 找到就结束 找不到返回-1

toString(转成字符串)

split(切割符) 可以把字符串切割成数组 一定会切成数组

2.Number对象

toString(转成字符串)

toFixed()按几位小数四舍五入取整

3.Boolean对象

toString(转成字符串)

4.Array对象

toString(转成字符串)

length

join(连接符) 把数组连接成字符串 结果一定是字符串

slice()截取子数组

indexOf()

map() 对数组进行处理，返回一个全新的数组

filter()返回一个符合指定条件的数组

----------以上所有方法都不能修改原数组----------

push()向数组的结尾追加元素

unshift()向数组开头追加元素

pop()删除数组最后一位元素

shift()删除数组第一位元素

splice(从哪一位开始删除,删几个,新值) 在任意位置添加和删除元素的

　　返回的是被处理之后的数组

reverse()数组反转

sort()数组排序

d.sort(function(a,b){return a-b});

冒泡排序：

```
        function arrSort(arr) {
            for (var j = 0; j < arr.length - 1; j++) {
                for (var i = 0; i < arr.length - 1 - j; i++) {
                    if (arr[i] > arr[i + 1]) {
                        var box = arr[i]
                        arr[i] = arr[i + 1];
                        arr[i + 1] = box
                    }
                }
            }
            return arr
        };
        var c=[[2,54,6,9,878,123],[12,63,5,778,986,156],[12,6,5,778,76

        for(var m=0;m<c.length;m++){
            console.log(arrSort(c[m]))
        }
```

5.正则表达式对象RegExp

正则表达式：定义字符串中字符出现的规律

正则表达式要求写在/正则表达式/中

中括号用来存放备选字符

一个中括号只能代表一位字符的匹配规则

正则表达式对于任意连续的区间都可以用-连接

数量词{}

{num} 代表前面一位规则重复几次

{min,}代表前面一位规则至少min,

{min,max}代表前面一位规则至少min，最多max

特殊数量词：

? 可有可无 最多一次 {0，1}

* 可有可无 最多不限 {0，}

+ 至少一次 {1,}

预定义字符集：在正则表达式中一些有特殊含义的字符

\d 代表了所有的数字

\w 代表所有的数字字母下划线

. 代表任意字符

\s 代表空格

如果备选字符中只有一个备选字符或者只有一个预定义字符集，那么中括号可省略

对于在正则表达式中有特殊含义的字符，如果希望以原文形式去匹配，需要用\转义

test()

reg.test(被检验的字符串) 返回布尔值

注意：正则表达式是部分匹配

在整条正则表达式的开头加^代表以...开头，在整条正则表达式的结尾加$,代表以...结尾

在中括号开头加^代表除了...都行

7.Math对象

abs()取绝对值

round()四舍五入取整

ceil()向上取整

floor()向下取整

min()/max() 注意不接受数组当参数，只能接受参数序列

random()取0-1之间的随机数

8.Date对象： 封装了所有与日期相关的api

1.创建日期对象：new Date() 默认保存的是当前时间

2.日期对象可以直接相减，得到的是间隔毫秒数

3.getFullYear() 返回是哪年 number

4.getMonth()返回的是月份 0-11 要+1修正

5.getDate() 1-31

6.getDay() 0-6

7.getHours() 0-23

8.getMinutes() 0-59

9.getSeconds() 0-59

10.getMilliSeconds() 0-999

11.getTime()返回的是1970-1-1至今的毫秒数

12.以上get方法全改为set即为更改时间，注意，没有setDay()方法

9.Error对象

SyntaxError(语法错误) ReferenceError(引用错误) TypeError(类型的错误)

10.DOM document object model

1.增删改查 crud

所有写在元素开始标签之中的都是元素对象的属性

1.查： 查找元素

document.getElementById("id名") 返回元素对象

document.getElementsByClassName("class名") 返回数组

document.getElementsByTagName("标签名") 返回数组

docuemnt.getElementsByName("名字") 返回数组

document.querySelector(css选择器) 返回元素对象

document.querySelectorAll(css选择器) 返回的数组

2.改

改属性 1.通过对象的方式

2.setAttribute("属性名","属性值")

getAttribute("属性名")

改内容

innerText 可以获取到元素开始标签到结束标签之间的文本内容

innerHTML可以获取到元素开始标签到结束标签之间的内容

3.删

删内容

innerHTML=""

删属性

removeAttribute("属性名")

删元素

父元素对象.removeChild(子元素对象)

4.增

增加元素

（1）创建元素

document.createElement("标签名")

（2）添加到对应的位置

父元素对象.appendChild(子元素对象)

（3）添加属性和内容

11.事件

用户的动作触发的

onclick:点击事件

onfocus：获得焦点事件

onblur：失去焦点事件

12.this

this指向的是函数运行时所在的对象(谁调用了函数，那么函数中的this就指谁)

window对象的属性和方法在访问的时候都可以省略前缀

13.定时器

定时器是让网页自动运行的唯一办法

周期性定时器：每隔一段时间，做什么事

setInterval(函数，间隔 毫秒数)

一次性定时器：等待一定的时间，做什么事

setTimeout(函数，等待毫秒数)

定时器是一个异步多线程的程序

定时器的执行结果是线程号

停止定时器

clearInterval(线程号)

clearTimeout(线程号)

14.原型与继承

原型(prototype)：方法背后，专门保存由方法创建出来的对象的共有属性

1.对象字面量的形式　var obj={name:"小明",age:18}

2.通过构造函数的形式 new Date() new Array() new Object() new RegExp()

构造函数/对象模板：专门用来创建相同结构对象的专门方法

new关键字做了哪几件事?

1.创建了一个空对象　var ll={}

2.改变this指向 call apply

3.加属性 this.name="小明" this.age=18

4.返回一个全新的对象　return ll

共有属性：由同一构造函数创建出来的对象共同享有的属性

自有属性：属于对象实例私有的属性

任何对象没有权利修改原型中的属性

继承：使用现有类型，创建出新的类型，新的类型可以使用现有类型的属性和方法，也可以拓展出现有类型没有的属性和方法

15.原型链

Function 代表的是所有函数的父类

__proto__:隐式原型。 任何一个对象都有隐式原型，用来实现继承的

　　一个对象的隐式原型默认指向创建该对象的构造函数的原型 对象

16.绑定事件的第三种方式

元素对象.addEventListener("事件名",方法对象,是否在捕获阶段触发)

17.事件触发周期

事件捕获(外--里)--目标触发--事件冒泡(里--外)

事件对象：默认在事件触发的时候自动传入函数的第一个参数，与生俱来的，不是后天传入的

阻止冒泡: e.stopPropagation()

事件委托:利用冒泡

事件源对象:e.target

18.ajax : 前端向后端异步的取数据的而无需刷新页面技术

1.使用ajax 的步骤

```
//1.创建ajax核心对象
        var xmlhttp=new XMLHttpRequest();
        //2.创建请求  get/post    请求地址    是否异步
        xmlhttp.open("get","my.php?user=982395099&psd=123456",true)
        //3.发送请求参数  key=value形式的字符串 多个参数之间以&连接
            //get请求 请求参数不能写在send里面，要写在请求地址后面，以?连接 send
        xmlhttp.send(null);
            //如果请求方式为post，需要设置请求头
            xmlhttp.setRequestHeader("Content-Type","application/x-www-
         //4.接收响应
             //onreadystatechange
             // readyState (请求的状态) 0(尚未初始化) 1(正在请求)2(请求完毕)
             // status(服务端返回的状态码) 404 500 200(ok) 301 304
           xmlhttp.onreadystatechange=function(){
               if(xmlhttp.readyState==4&&xmlhttp.status==200){
                   //responseText    可以服务器端返回的文本格式的数据
                  var data=xmlhttp.responseText
               }
           }
```

2.get和post的区别：

参数的位置不同

get更快更简单， get有参数数量的限制

post请求更安全更稳定 表单提交（包含未知的用户输入的时候） 没有参数数量限制

3. json:数据格式

JSON.parse()

JSON.stringify(str)

19.jquery： js库

简化版本的js

1.常用的选择器

*	$("*")	所有元素
#id	$("#lastname")	id="lastname" 的元素
.class	$(".intro")	class="intro" 的所有元素
.class,.class	$(".intro,.demo")	class 为 "intro" 或 "demo" 的所有元素
element	$("p")	所有 <p> 元素
el1,el2,el3	$("h1,div,p")	所有 <h1>、 和 <p> 元素
:first	$("p:first")	第一个 <p> 元素
:last	$("p:last")	最后一个 <p> 元素
:even	$("tr:even")	所有偶数 <tr> 元素，索引值从 0 开始，第一个元素是偶数 (0)，第二个元素是奇数 (1)，以此类推。
:odd	$("tr:odd")	所有奇数 <tr> 元素，索引值从 0 开始，第一个元素是偶数 (0)，第二个元素是奇数 (1)，以此类推。
:first-child	$("p:first-child")	属于其父元素的第一个子元素的所有 <p> 元素

parent > child	$("div > p")	元素的直接子元素的所有 <p> 元素
parent descendant	$("div p")	元素的后代的所有 <p> 元素
element + next	$("div + p")	每个 元素相邻的下一个 <p> 元素
element ~ siblings	$("div ~ p")	元素同级的所有 <p> 元素
:eq(index)	$("ul li:eq(3)")	列表中的第四个元素（index 值从 0 开始）
:gt(no)	$("ul li:gt(3)")	列举 index 大于 3 的元素
:lt(no)	$("ul li:lt(3)")	列举 index 小于 3 的元素

:disabled	$(":disabled")	所有禁用的元素
:selected	$(":selected")	所有选定的下拉列表元素
:checked	$(":checked")	所有选中的复选框选项

2.显示隐藏的方法

hide(ms) show(ms) toggle(ms)

fadeOut() fadeIn() fadeToggle()

slideUp() slideDown() slideToggle()

3.css() 直接改样式没有过程

4.animate(cssObj,ms) 有动画有过程的去改样式

5.stop() 停止当前动画

6.jquery允许链式调用：允许你对连续执行多个jq方法，会按照绑定的顺序依次执行

7.jquery的callback(回调函数)

8.html() text() val() attr("src")

9.append() prepend() after() before()

10.remove() empty()

11.$(this)

12.遍历方法 parent() parents() parentsUntil() children(".box") find("div") siblings() next() nextAll() nextUntil() prev() prevAll() prevUntil()

13. jquery ajax

跨域问题：

浏览器的同源策略 同协议 同域名 同端口号

解决跨域：jsonp 请求代理

20. ES6

1.let 声明的变量没有提升 不允许重复声明 只在块级作用域内有效

暂时性死区：在一个块级作用域内，如果用let声明了某个变量，那么该变量就自动绑定了该作用域，该作用域就形成了一个封闭的作用域。

2.const 声明常量

3.模板字符串 `` 允许换行 可以直接写变量，变量要写在${}里面

4.扩展运算符 ... 将数组或者类数组结构拆分为参数序列

5.函数的扩展 箭头函数 let fn=(m)=>{return m}

如果只有一个参数，可以省略小括号，如果想返回一个值，return连同大括号一起省略，如果要返回一个对象的话，对象外面要套括号避免歧义

箭头函数中的this指向：指向的是函数定义时所在的对象

箭头函数不允许当作构造函数，也就是不能new

6.变量的解构赋值

let [变量1，变量2，变量3]=[值1，值2，值3]

7.对象的扩展

如果对象的属性值是个变量，并且该变量名跟属性名一样，那么可以省略为一个，

对象中的函数属性，可以省略：function

8.对象的解构赋值

let {name,age}={name:"小红",age:18}

```
let obj={
            name:"小红",
            age:18,
            sex:"girl",
            address:"北京"
```

```
        }
    function fn({name,age}){
        console.log(name+age)
    }
     fn(obj)
```

如果出现多层嵌套的对象需要解构

```
let obj={
            name:"小红",
            age:18,
            sex:"girl",
            hobby:{
                sport:{
                    aaa:""
                },
                movie:"生化危机"
             }
        }
    let {hobby:{sport:{aaa},movie}}=obj;
```

9.Symbol 新的原始数据类型

symbol:类似于字符串的，值永远是独一无二的

10.Set 类似与数组的一种数据结构

set数据结构值都是唯一的

add()

delete()

has()

size()

keys（）遍历属性名

values()遍历属性值

entries()遍历属性名和属性值的

11.Map类似于对象的一种数据结构

```
let map=new Map([['name','小明'],[12,18],[[1,2,3],567]])
```

应该叫值值对

map的属性名不局限于字符串

set(“属性名”,"属性值")

get("属性名")取值

has("属性名")

delete("属性名")

clear()

size()

keys（）遍历属性名

values()遍历属性值

entries()遍历属性名和属性值的

12. Promise

异步编程：定时器 ajax

解决方案： 回调函数

then()方法返回的是一个全新的promise实例

```
function fn() {
                    var a=new Promise(function(resolve,reject){
                    setTimeout(function(){
                            console.log(666);
                            resolve("haha")
                    },2000)

            })
                    return a
            }
            function fn2(){
                    var b=new Promise(function(resolve,reject){
                    setTimeout(function(){
                            console.log(777);
                            resolve("呵呵")
                    },1000)

            })
                    return b
            }

            fn().then((res)=>{
                    console.log(123,res);
                    return fn2()
            }).then((res)=>{
                    console.log(99,res)
            })
```

13.async await函数：

async 是“异步”的简写，而 await 可以认为是 async wait 的简写。所以应该很好理解 async 用于申明一个 function 是异步的，而 await 用于等待一个异步方法执行完成。

async函数会返回一个promise对象，如果在函数中return一个值，那么该值会通过Promise.resolve()传递出去

一般来说，都认为 await 是在等待一个 async 函数完成。不过按语法说明，，await 等待的是一个表达式，这个表达式的计算结果是 Promise 对象或者其它值（换句话说，就是没有特殊限定）。

因为 async 函数返回一个 Promise 对象，所以 await 可以用于等待一个 async 函数的返回值——这也可以说是 await 在等 async 函数，但要清楚，它等的实际是一个返回值。注意到 await 不仅仅用于等 Promise 对象，它可以等任意表达式的结果，所以，await 后面实际是可以接普通函数调用或者直接量的。所以下面这个示例完全可以正确运行

```
function getSomething() {
    return "something";
}

async function testAsync() {
    return Promise.resolve("hello async");
}

async function test() {
    const v1 = await getSomething();
    const v2 = await testAsync();
    console.log(v1, v2);
}

test();
```

await 等到了它要等的东西，一个 Promise 对象，或者其它值，然后呢？我不得不先说，
await 是个运算符，用于组成表达式，await 表达式的运算结果取决于它等的东西。
如果它等到的不是一个 Promise 对象，那 await 表达式的运算结果就是它等到的东西。
如果它等到的是一个 Promise 对象，await 就忙起来了，它会阻塞后面的代码，等着 Promise 对象 resolve，然后得到 resolve 的值，作为 await 表达式的运算结果。
案例

```
function fn() {
                    var a=new Promise(function(resolve,reject){
                    setTimeout(function(){
                            console.log(666);
```

```
                    resolve("haha")
                },2000)

        })
                return a
        }
        async function fn2(){
                let b=await fn();
            console.log("执行",b)
        }
        fn2()
```

14.module

14.module语法

(1)export var a=1 可以导出任何的变量声明语句或者函数声明语句

export {a,b,fn}

export var a=1 一定是完整的变量声明语句

import {a} from "./a.js"

(2)改名字

导出 export {原名 as 新名}

引入 import {导出的名字 as 新名}

(3)默认导出:从前面的例子可以看出，使用import命令的时候，用户需要知道所要加载的变量名或函数名，否则无法加载。但是，用户肯定希望快速上手，未必愿意阅读文档，去了解模块有哪些属性和方法。

export default:

一个文件只能有一个默认导出

引入的时候不需要大括号

（4）整体引入

import "./common.css"

import "./../js/main.js"