**预解释：**在当前作用域下(刚开始是window全局作用域)，JS代码从上到下执行之前，首先会把所有带var和function关键字的进行提前的声明(或者定义)

带var关键字的和带function关键字的是不一样的

var:预解释的时候只是声明

function:预解释的时候声明和定义一起完成了

var a=12;

声明：declare var a; 只声明不定义，默认值是undefined，是在预解释的时候完成的

定义：defined a=12; 赋值，其实相当于我们的定义，是在代码执行的时候完成

预解释是毫无节操的一种机制

1、预解释发生在当前的作用域下

2、不管条件是否成立，{}(代码块)中的代码都要进行预解释

3、只有=(赋值号)左边进行预解释，右边的不进行预解释

4、匿名函数中的自执行函数，在window下不进行预解释

5、在函数中return后面的代码不在执行了，但是需要进行预解释

6、在预解释的时候，如果发现重名了(不管是函数名还是变量名，名字一样也是重名，函数名和变量名也不能相同)，不重复的声明，但是如果预解释的时候还需要定义的话，需要重新定义

7、预解释只发生在同一个脚本块中

8、在window下function是不进行预解释的，但是当代码执行到这的时候，自执行函数执行，形成一个私有的作用域，在这个私有作用域中是需要预解释的

9、由于函数在预解释的时候声明和定义都完成了，所以函数可以在页面中的任何位置执行(即使在我们看到的定义部分代码之上执行也是没有问题的)

普通的函数写法和函数表达式的区别:

预解释的时候，声明和定义都完成了

fn(); 可以正常的执行

function fn() {}

预解释的时候，只是声明一个fn

fn();//-->undefined(); Error:fn is not a function

var fn = function () {}

if (!("a" in window)) {

var a = 13;

}

console.log(a);//undefined

window下的预解释：var a;

不管条件是否成立都要进行预解释

**in：**检测某一个属性名是否属于这个对象，是的话返回true，不是的话返回false 例如："a" in window 检测a是否为window这个对象的一个属性

num = 13;//没有加var默认是window.num=13; (非严格模式下) 以后定义变量var必加

function fn(a) {

arguments[0] = 20;

//在非严格模式下，arguments通过索引修改传递参数值，可以映射到我们的形参变量上

console.log(a);//20

}

fn(10);

**this关键字：**

this:我们所学的this基本上都是在函数中应用的，

this指的是当前行为（方法）执行的主体

context:上下文指的是是当前行为（方法）执行的环境

JS中的this:

this是谁跟函数在哪定义的和在哪执行的无关

1.this是谁，就看函数执行方法名前面有没有"."，有的话，"."前面是谁，this就是谁；没有的话，this就是window.(万能规律)

2.自执行函数里面的this永远是window，这个和自执行函数在哪执行的没关系

3.给元素绑定事件（例如点击事件），当触发事件的时候，函数里的this是当前被绑定事件的这个元素

4.在构造函数模式中，this就是当前的实例

5. 使用call/apply强制改变this

**作用域：**

**栈内存：**用来运行JS代码和存储基本数据类型值的 例如：作用域(全局作用域)

**堆内存：**引用数据类型在操作的时候，首先会开辟一个新的内存空间，用来存储数据(对象的属性或者是函数的代码字符串)，这个新开辟的内存空间就是堆内存

作用域是一个栈内存，用来提供JS代码执行的环境和存储基本数据类型的值的

在JS中的作用域只有两种：私有作用域(函数执行的时候形成的)、全局作用域(window)

**全局作用域：**当浏览器加载页面的时候，会默认开辟一个供JS代码执行的环境，把这个执行环境叫做全局作用域。

全局作用域只有在页面的关闭的时候才会销毁。

全局作用域下声明的变量都是全局变量，不仅是全局变量，还属于window对象的一个属性(只有全局作用域才是这样)。

全局变量可以在页面中的任何位置获取使用，只有关闭(刷新)页面的时候，全局作用域才销毁。

项目中应避免使用全局变量，防止太多的全局变量造成变量的冲突(防止全局变量的污染)

**私有作用域：**只有函数执行才会形成私有的作用域

**私有变量有且只有以下两种情况:**

在私有作用域中声明过的变量属于私有的变量

形参也是私有的变量

在私有作用域中定义的私有的变量和函数，我想在全局下使用

1.加前缀window, window.xxx

2.通过return返回 return xxx

在私有作用域中如果想修改和使用全局的变量值，最简单的方式是加前缀window，其实就是给window加一个属性，即全局变量。（所有的全局变量都是window的属性，但window的属性不一定是全局变量）

A、在新的私有作用域中第一步给形参赋值，形参属于我们当前私有作用域中的私有变量;

B、私有作用域下的预解释，也是找到所有带var和function关键字的，在私有作用域中声明的变量(函数)都是私有的变量;

C、代码从上倒下执行，如果遇到一个变量，首先看是否为私有的变量，如果没有则往上一级作用域查找，上级没有继续向上查找，直到找到window为止；如果是私有的变量，那么里面出现的所有的变量都是私有的，和外面没有关系；

D、一般情况下，当我们的代码执行完成，当前作用域就立即销毁了

**如何查找当前作用域的上一级作用域：**

看当前作用域（C）对应的堆内存是在哪个作用域（A）下定义的,那么C作用域的上一级作用域就是A

就看当前函数对应的堆内存，是在哪个作用域下定义的，在谁下面的定义的它的上级作用域就是谁！与函数在哪执行无关。

**什么情况下不会销毁私有作用域？**

函数执行,在形成的私有作用域中返回了一个引用数据类型值,如果在外面有一个变量接收了这个值(外面变量占用了这里面的东西)，当前这个作用域就不能销毁了;

如果外面并没有变量接收，则当前的这个作用域是不立即销毁，当浏览器空闲的时候再进行销毁

在一个私有作用域中，我们定义了一个引用数据类型的值，并且把这个地址给了元素的绑定事件，这样当前的这个作用域也不销毁

**闭包：**

当函数执行的时候，会形成一个新的私有的作用域(供函数体中的代码执行和存储基本数据类型值的--栈内存)，来保护里面的私有变量不受外界的干扰(外面不能直接的修改，也不能直接的获取)，我们把函数的这种保护机制称为**闭包**

**闭包不是具体的某一种形式而是一种机制**

函数执行会形成一个新的私有的作用域，保护里面的私有变量不受外界的干扰(函数的这种保护机制叫做闭包)

函数：实名函数和匿名函数

**匿名函数:**

**1、函数表达式：**把函数定义的部分给一个变量或者一个事件

var fn=function(){}

oDiv.onclick=function(){}

**2、自执行函数：**定义和执行一起完成了

;(function(){})();

~function(){}();

!function(){}();

+function(){}();

-function(){}();

**垃圾回收机制：**

浏览器是有自己的内存回收机制的(垃圾回收机制，不同的浏览器是不一样的)

谷歌：

开一个内存有一个地址，浏览器看当前这个地址都谁知道(看哪个变量存储了这个地址)，只要有一个知道的，当前的这个内存就不能销毁，但是如果一个知道的都没有，浏览器在空闲的时候，会进行清除

var obj={};//obj=xxxfff000

obj=null;//把obj存储的地址清除，这样以后就没有人知道这个地址了，浏览器空闲的时候会自动的进行清除

火狐和IE：

开一个内存，有一个变量引用这个内存的话，我们就在内存的计数器上累加1，当我们把变量的这个地址清除的时候，计数器减少1，当计数器为0的时候，浏览器清除这个内存

var obj={}; 开一个内存，计数器=1

obj=null; 计数器=0 浏览器销毁这个内存

IE下浏览器反应慢，可能会导致内存泄露

**i++和++i：**

i++和++i：都是自身数值累加1，只在于和其他的值进行运算的时候有区别

i++:先拿i的值个其他值相加，完成后在执行我们的i的累加操作

++i:先让i累加1，然后在让累加后的值个其他的值运算

（i—和—i同理）

**逻辑运算符：**

&& ,||

在判断的条件中出现:&&只有所有的条件都为真，整体才为真 ||只要有一个条件为真，整体就为真

不在判断中出现：

**value1||value2** 首先把value1转换为布尔值，判断是真还是假，如果为真 当前返回的值是value1，如果为假 当前返回的值是value2 (最常用的)

**value1&&value2** 首先把value1转换为布尔值，判断是真还是假，如果为假 当前返回的值是value1，如果为真 当前返回的值是value2

&&的优先级高于||

在if条件中，不管条件写的多复杂，我们首先按照&& || 将大条件拆分成小条件

if(g()&&[]==![]) ==>true 条件成立

1、g() -->true 2、[]==![]

如果是两个值比较，首先按照==拆分左右的两部分

1)[] 2) ![] --> false []==false -->true