**关于JavaScript**

1. **ECMAScript**是JS 的语法标准。（ECMAScript）

//ES5、ES6是ECMAScript最主流的两个版本。

//ES5下没有块作用域，同一个全局变量多个取值时，会取最后一个值。

1. Chrome控制台调试
   1. 快捷键：

Fn+F12打开控制台//控制台可以直接输入代码段并运行

Ctrl

+ r //刷新

+ L //清空之前内容

+ 鼠标滚轮 //放大缩小

Shift

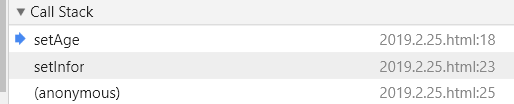
+ enter //输入过程中换行

* 1. Source调试
     1. 用断点进行调试：打开source，单击所在行数字序号。
     2. console.log调试
     3. trace调试
     4. error调试

1. 操作说明：

~0W1AHZJY5C_KJ{%_%]E}IS.png

* 1. 继续执行，下一个断点。
  2. 跳入下一个函数
  3. 跳出当前函数
  4. 逐行执行



Call Stack里可以查看正在执行的函数

1. 调试步骤：

① 代码测试

1. 找到调试代码文件
2. 添加断点
3. 继续执行、跳出函数、跳入函数、逐行执行

**JS进阶：**

**概述**

1. JS语言特点：
   1. JS是一种直译式脚本语言。（逐行编译逐行执行，开发效率高，运行效率低）
   2. JS是一种弱类型、动态类型语言。

-写程序时不用给变量指定特定的数据类型。

-可以动态的更改变量类型

* 1. 没有块作用域等。

- ES5没有块作用域、函数式编程、闭包、基于原型链的继承方式、动态添加属性等

- 借鉴了Java的语法、Self原型继承、Python的正则等。

1. JavaScript位置：

内部：

如果JS写在html之前（head内），DOM操作可能会报错，常写在body后。

外部：

<script src = ‘相对路径’></script>

1. 数据类型

String、number、boolean、undefined、null

//字符串默认转换布尔类型，空串为false。

//字符串和数值类型强制转换为字符串。

//字符串强制转换为数值型：

var str = “1022a”;

var num = parseInt(str); /\*1022\*/

//字符串默认转换布尔类型：

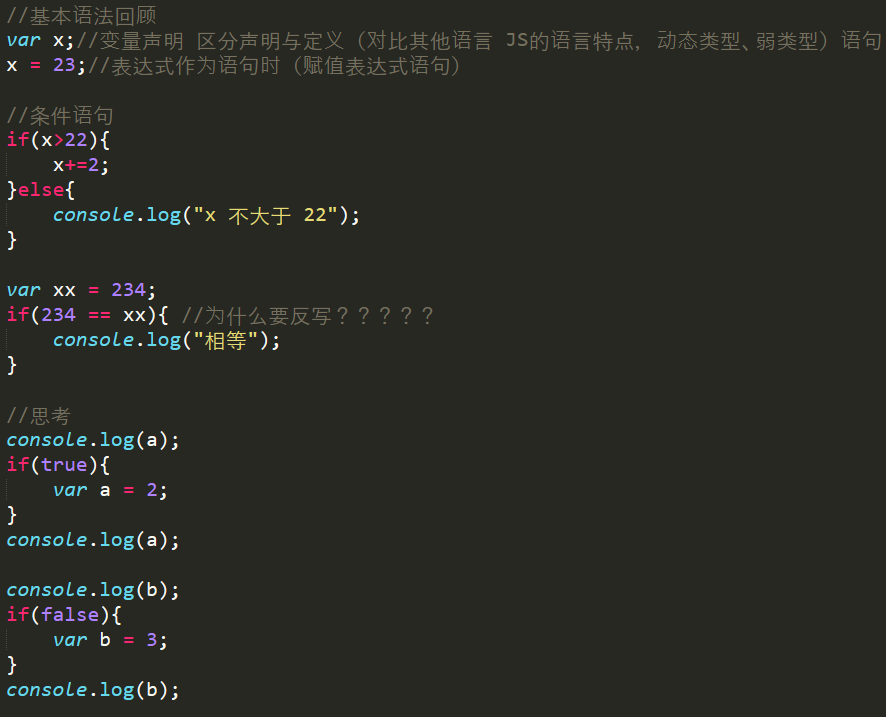
//var a = true&&false;

console.log(b);

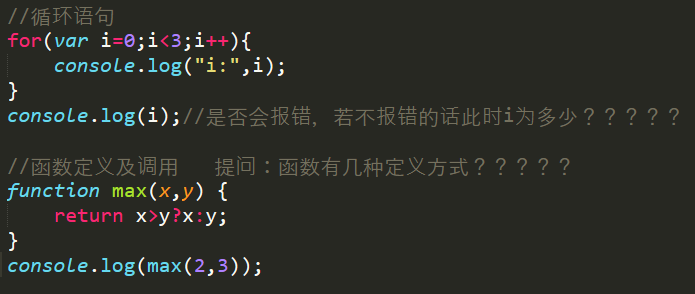
var b = abc&&false; /\*abc强制转换为true\*/

console.log(b);

1. 基础JavaScript回顾
   1. 背景、语法：



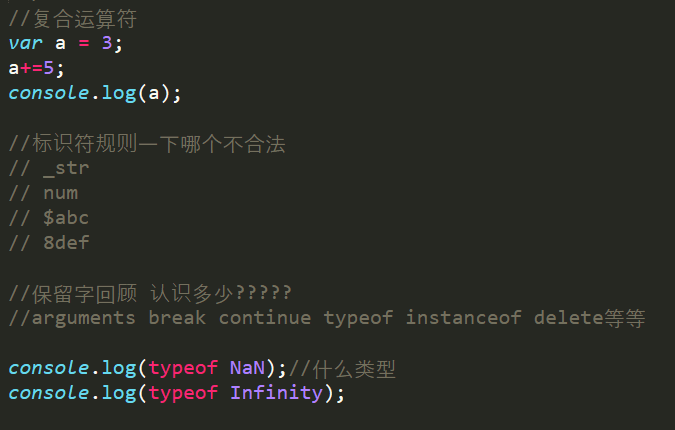
**反写原因：**



**i为**

**函数的定义方式有**

* 1. 变量和赋值



**保留字回顾：**

**类型：① number ②number**

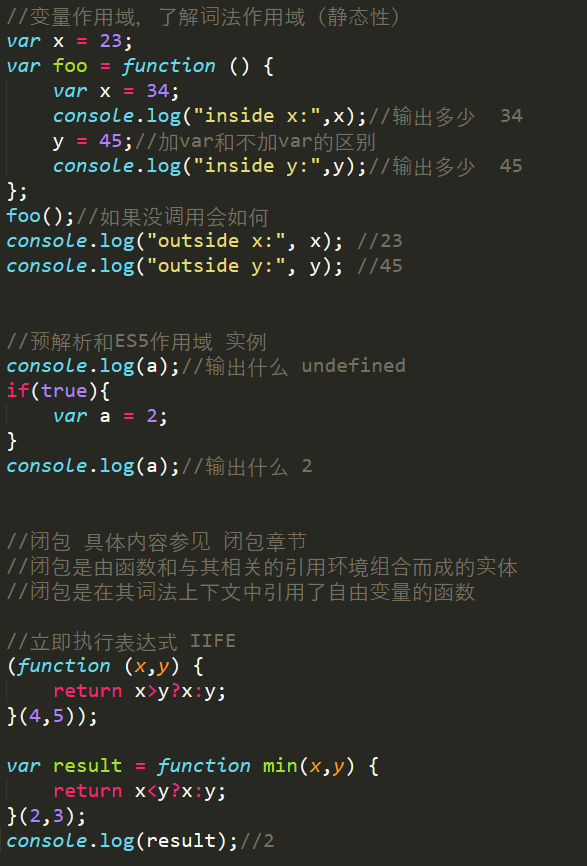
* 1. 函数、异常捕获、严格模式



(1)为什么foo函数可以声明提升，在定义前执行？

(2)参数数量问题。

④ 作用域和闭包



**IIFE的作用？**

* 1. 对象、构造函数





* 1. 数组、正则表达式、Math





**值和数据类型**

1. JS数据类型及其划分

基本数据类型：

String、Number、Boolean、Undefined、Null

引用数据类型：

Object（Array，Date，Function、Error等）

例：基本数据类型变量之间相互独立:

var num1 = 20;

var num2 = num1;

var num1 = 10;

此时num2仍为20.

引用数据类型变量之间不独立：

1.

var object1 = {“username”:”zhangsan”};

var object2 = object1;

object2.username = “lisi”;

console.log(object1);//lisi

2.

var arr1 = [1,2,3,4];

var arr2 = arr1;

arr2[1] = 5;

console.log(arr1);//[1,5,3,4]

console.log(arr2);//[1,5,3,4]

即：引用数据类型变量存储的是引用，根据引用来寻找堆区的值。

1. 基本类型与引用类型的区别

内存分配方式不同：

基本数据类型在**栈区**

引用数据类型在**栈区或堆区**，引用数据类型的对象在**堆区**,引用在栈区或堆区。（如果栈区的引用指向同一块堆区空间，一个引用变化，都将会发生变化。）



A和b分配在栈区。？

·思考：对象的属性如果是基本类型，那么该属性是分配在堆区还是栈区？

堆区？

赋值时不同：LS02-demo03

- 赋值、赋引用（地址）、深拷贝与浅拷贝

判等时的不同：LS02-demo04

- 值类型是判断变量的值是否相等（值比较）

- 引用类型是判断所指向的内存空间是否相同（引用比较）

函数参数传递时的不同：LS02-demo05

- 按值传递(call by value)

- 按引用传递(call by reference)

注意：真正决定这几种不同的是**数据类型**，而不是内存分配方式，内存分配方式决定的是变量的生命周期

1. 基本数据类型的值

·Number类型的值

- 整数与浮点数

- NaN(not a number)、Infinity(无穷大)、-Infinity(无穷小)、+0、-0

•String类型的值

- 空字符、字符和字符串、转义字符

eg：

**①**

var str = "abcde";

str.slice(0,-1);

//"abcd"

**②以特定分隔符分隔数组**

var str = "ab\_cd\_e";

str.split("\_");

console.log(str);

//ab\_cd\_e

console.log(str.split("\_",2));

//["ab", "cd", "e"]

•Boolean类型的值

- true、false

•Null与Undefined

- null、undefined



1. 包装对象(JS中所有过程都是对对象处理的一个过程)

//var str = new String(“abcdef”);

//var str = “abcdef”;

//以上两种写法相同

//装箱操作

var str1 = “abcdef”;

var strObject = new String(str1);

strObject.substr(0,2);

window.parseInt();//把全局window省略

1. 数据类型转换详见LS02-demo11
2. 详见LS02-demo12

·隐式类型转换

- 使用关系运算符时的转换（==、>、<、引用类型和基本类型比较时）

- 使用算数运算符时的转换（'img'+ 3 + '.jpg'; “25”-0;）

- 使用逻辑运算符时的转换（ !!0; ）

- 执行流程语句时的转换（if(obj){...}）

•显式类型转换（使代码更清晰）

- Boolean（）、Number（）、String（）、Object（）

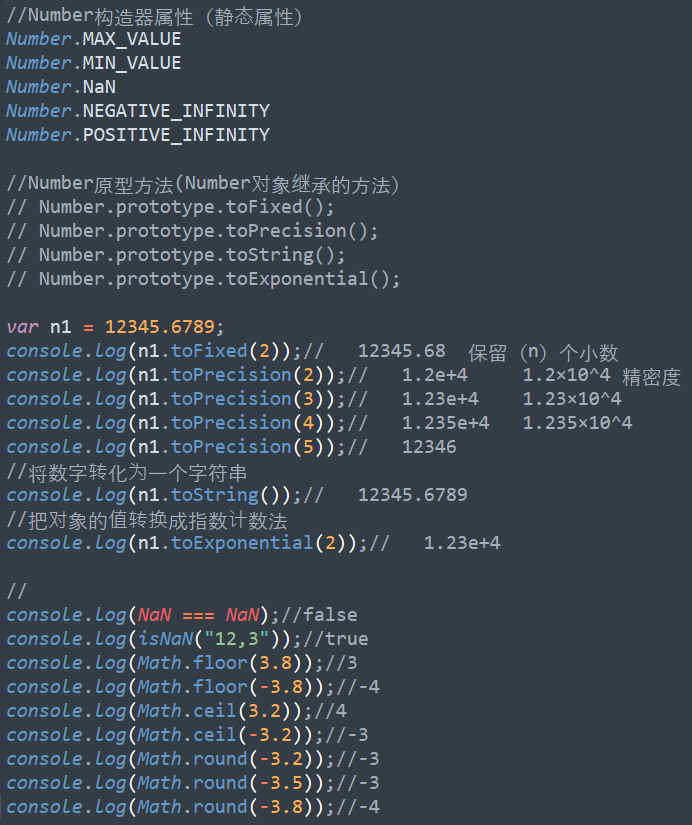
- 数转为字符串（toString()、toFixed()、toPrecision()、toExponential()）

- 字符串转为数字（parseInt()、parseFloat()）

- 对象转换为原始值（toString()、valueOf()）

**基本数据类型进阶**

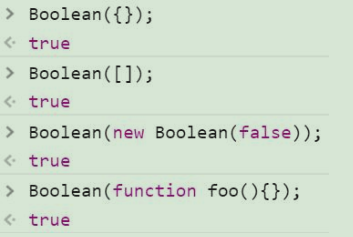
1. Number



1. String

**demo14**

1. Boolean



Boolean(null);//false

**JS语法、表达式及语句**

1. 字面量

- 对象字面量 var obj = {x:12，y:23};

- 数组字面量 var arr = [1,2,true,'xyz'];

1. 标识符与保留字

- 标识符用来给变量或函数进行命名，以字母、下划线或$为开始。（不以数字开头）

- 保留字：arguments、break、case、catch、class、const等。

1. 全局变量是JS一个重要缺陷。

解决办法：使用严格模式。

1. 严格模式
2. 严格模式下全局变量需显式声明。

// 严格模式使用方式

"use strict"//全局使用

function foo() {

//函数内部代码使用严格模式

"use strict"

username = "zhangsan";

}

foo();

console.log(username);//报错 demo04.js:24 Uncaught ReferenceError: username is not defined

1. ①非严格模式下，普通函数的内部this指向window对象。

function fn2() {

console.log(this);

}

fn2();

//Window {postMessage: ƒ, blur: ƒ, focus: ƒ, close: ƒ, parent: Window, …}

②严格模式下，this指向undefined。

function fn2() {

"use strict"

console.log(this);

}

fn2();//undefined

1. 严格模式下禁止删除不可改变的属性和未定义的变量
2. 严格模式下禁止函数参数重名
3. Switch详解
4. case在比较时使用的是全等操作符比较,因此不会发生隐式类型转换。

var i = "1";

switch(i){

case 1:

console.log("case 1 Number");

break;

case "1":

console.log("case 1 String");

break;

default:

console.log("default");

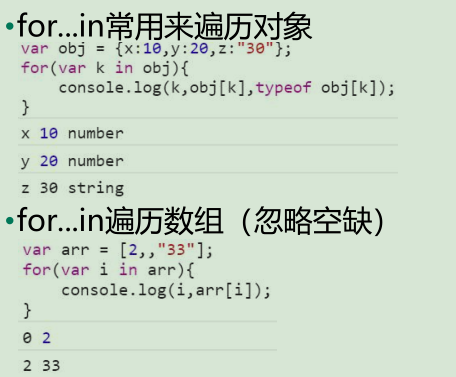
}

//此时输出 case 1 String

1. switch语句中的穿透性及其应用。

直到break才终止。

1. For...in
2. 遍历对象
3. 遍历数组

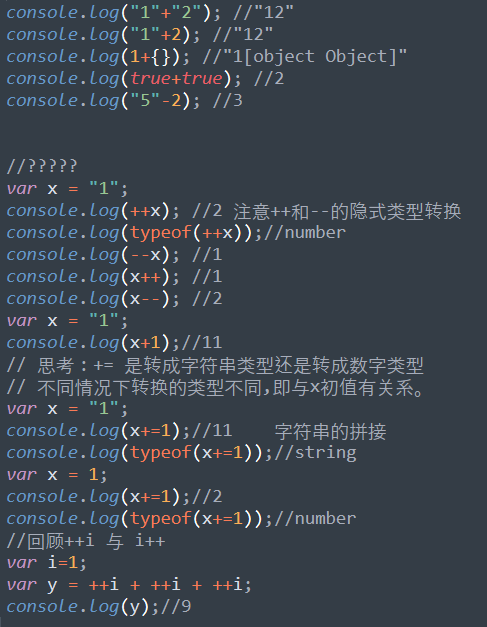


**JS运算符**

1. 赋值运算符
2. = 与 == 反写

好处： ==是判等，=是赋值，当在编写代码时，如果将==误写为=，反写能够及时发现并报错。

1. 算术运算符



1. 关系运算符

（1） == 与 ===

※ == （如果类型不同，先转换再比较，注：引用类型到基本类型的转换方向）

※ === （若类型不同则false，若类型相同则判断同 ==）

（2） != 与 !==

！=（相当于==的逆运算）

！==（先判断类型，若类型不同则返回true，相当于===的逆运算）

1. 逻辑运算符
2. &&与| |基本理解和应用

- 对于&&来说， 除了两侧都为真时为真，其他情况都为假

- 对于 | | 来说，除了两侧都为假时为假，其他情况都为真

（2）&&与||深层次理解

①&& ：左操作数为true，直接返回右操作数；左操作数为false直接返回左操作数。

②|| ：转换后的左操作数若为true，则直接返回原始左操作数；若为false则直接返回原始右操作数。

（3）遵循短路特性，使用&&和||可用来实现条件语句

（4）使用||来设置函数参数的默认值

**函数**

1. 定义的三种方式

见demo-1

1. 调用

（1）作为函数直接调用（非严格模式下this为全局对象，严格模式下为undefined）

1. 作为方法调用（this为调用此方法的对象）
2. 通过call( )和apply( )间接调用（this为函数对象的call/apply方法的首个参数，移花接木）
3. 作为构造函数调用（this为实例化出来的对象）
4. 函数参数的数量问题

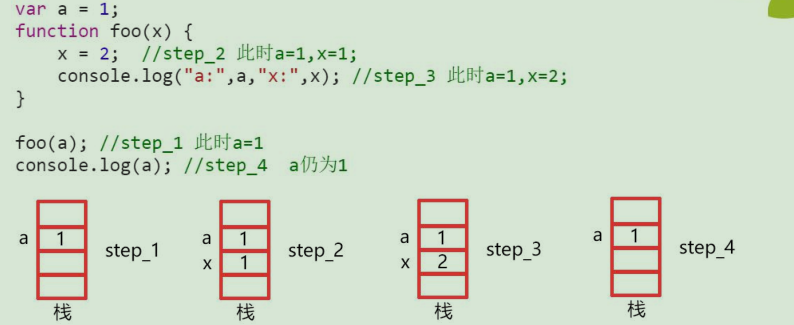
（1）JS函数调用时实参数量可以与形参不一致

①实参数量小于形参的情况（少的参数值为undefined、可使用| |来给出默认值）

②实参数量大于形参的情况（通过函数对象属性arguments获得所有实参、类数组对象、拥有对象属性）

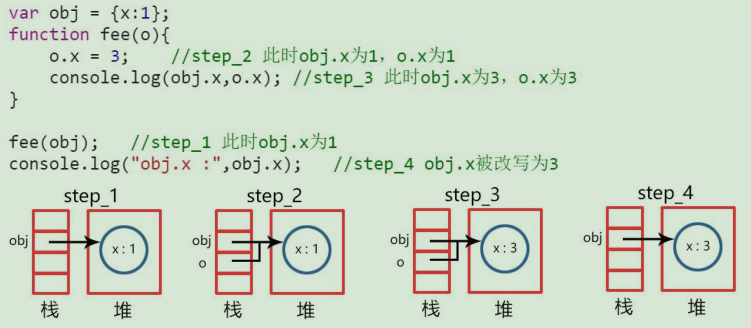
1. 参数类型与传递方式（值、引用）
2. 实参为基本数据类型时，形参改变不影响实参

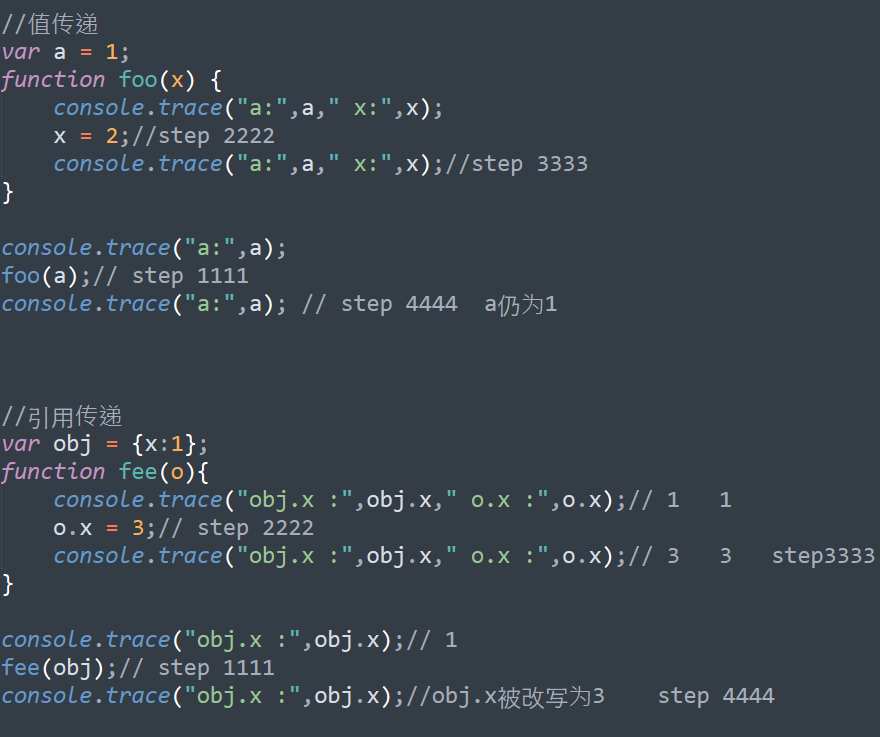
基本数据类型作为形参时，只是一个赋值的过程，基本数据类型本身的值不会受影响。



1. 实参为引用类型时，形参改变影响实参

引用数据类型一改全改。





**JS函数对象**

1. 函数对象（JS中每个函数都是作为对象来维护和运行的，即函数对象（既有属性也有方法））

详见demo08-5

1. 函数对象的属性及方法

函数对象的属性

- length、arguments（隐藏的局部变量）

- caller、callee（arguments的属性，常用于递归调用）

- prototype

函数对象的方法

- call、apply（可效果一致）

- bind

- toString、valueOf

1. 高阶函数

- 函数作为参数被传递（最常见的形式：回调函数）

- 函数作为返回值输出（与闭包有紧密联系）

**函数预解析**

1. JS解析及执行简介

（1）JS脚本语言（非提前编译，由解析器**边解析边执行，因此调试也相对简单**）

（2）解析和执行是一个不断交替的过程

1. JS预解析（声明提升）

（1）预解析主要工作（变量声明和函数声明提升）

- 解析器在执行代码前的进行代码扫描（var、function）

- 将变量和函数声明在当前作用域（全局、函数）内进行提升

（2）ES5中函数及变量声明重复的话，相当于覆盖。

1. 同时有var和function关键字时（情形1：函数表达式）
2. 当function前有运算符的话，认定为表达式，不 提升
3. 同时有var和function关键字时（情形2：变量名同函数名）
4. 预解析与作用域
5. 变量的作用域是指**变量在何处可以被访问到**

（2）全局变量与局部变量

- 全局变量：拥有全局作用域的变量（JS代码中任何地方都可以访问）

- 全局变量是跨域了所有函数自身作用域的自由变量，可以在函数内和函数外直接访问

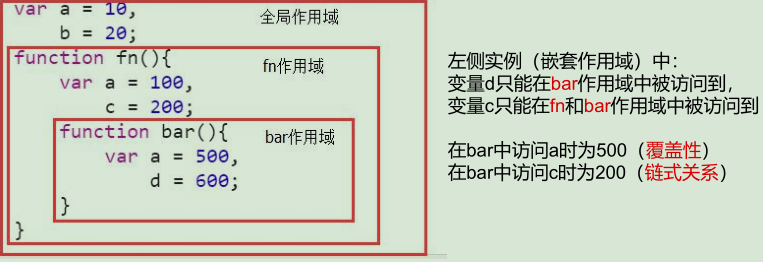
- 局部变量：函数内声明的变量，只在函数体内有定义，作用域是局部性的

- 在函数外不能直接访问函数的局部变量，但可以通过闭包来访问

- 函数内访问同名变量时，局部变量会覆盖全局变量。

**JS作用域及执行上下文**

1. JS作用域及其特点

（1）**作用域**就是变量和函数可以访问的范围。

**特点：**JS采用的是词法作用域（静态性），一般函数和变量定义完成后，作用域已经决定，与调用形式无关。.

特殊情况：new function，不遵循静态性。

（2）变量污染

1. 执行上下文（在sources的snnipets里新建，然后复制代 码，点ctrl+enter添加断点，再次点击运行，点击向右的播 放状箭头依次观察执行过程。）
2. 调用栈

以栈的方式来处理和追踪函数调用（函数调用栈 Call Stack）

1. 作用域链与执行上下文

- 执行时，当前执行上下文，对应一个作用域链环境来管理和解析变量和函数（动态性）

- 变量查找按照**由内到外**的顺序（遵循词法作用域），直到完成查找，若未查询到则报错

- 当函数执行结束，运行期上下文被销毁，此作用域链环境也随之被释放

**JS中的IIFE模式（**Immediately-Invoked Function Expression**）**

1. 适用方式
2. 解决问题
3. 案例