



# JS高频笔试题精讲——闭包专题

Web学科教学总监

张东（东神）





# 高频笔试题包括





# Q1: Q/A



“1. 上课能不能多回答聊天室的问题，”



“答: 大班课, 几百人在听,  
如果频繁暂停讲解, 回答问题,  
会导致课程内容不连贯,  
大多数人听课体验很差。”



所以：

“听课过程中先把问题记下来，  
私下私信问我或者助教老师。

最好不耽误几百人听课进度。”

众口难调。



“2. 课后是否可以有  
补充练习”





“答：课上我们讲的内容已经汇总了最高频的笔试题目。

课下我们鼓励大家自己多收集自己遇到的笔试**真题**。然后可以发给我或助教老师。我们把答案给你或者发到群里大家一起分享，形成“情报站”。



“ 3. 能不能安排10分  
钟休息

——今天安排！ ”





# Q2: 作用域和作用域链

## Scope & Scopes



“1. 回顾：  
作用域 & 作用域链,”



# 作用域(scope)

- 浅显的理解：

作用域就是**变量的可用范围**(scope)

- 为什么要有作用域：

目的是**防止不同范围的变量之间互相干扰。**

```
var x=1;
```

```
function f() {  
    var y=2;
```

```
        function g() {  
            var z=3;  
        }
```

```
}
```



# js中包括**2级**作用域

- 1). 全局作用域

```
var x=1;
```

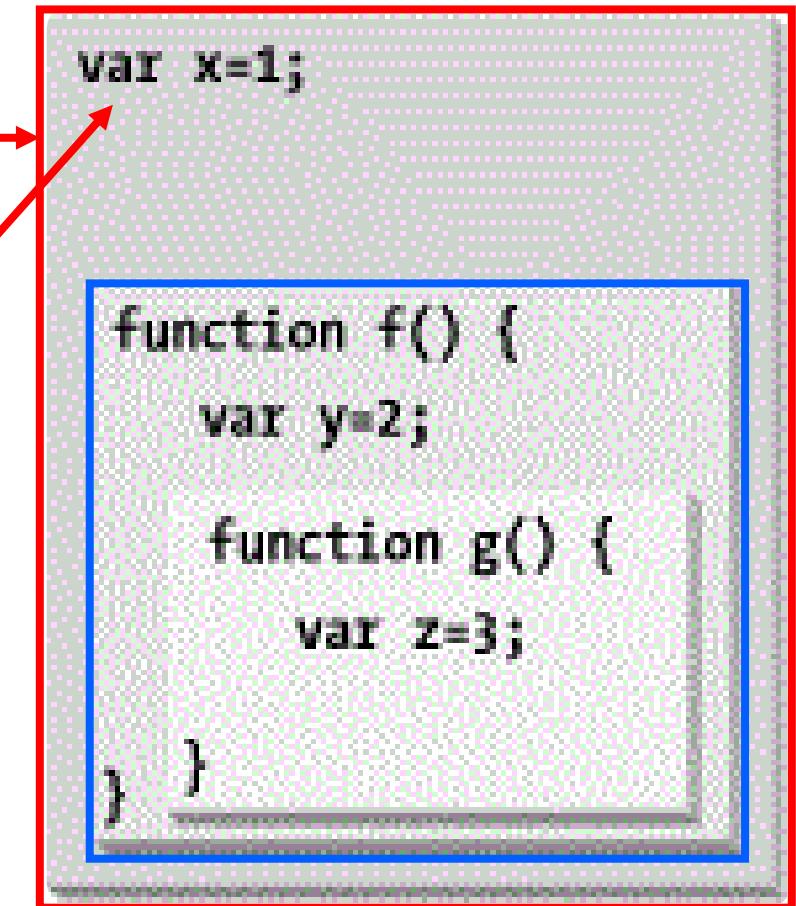
- 2). 函数作用域

```
function f() {  
    var y=2;  
  
    function g() {  
        var z=3;  
  
    }  
}
```



# 全局作用域:

- 不属于任何函数的外部范围称为全局作用域。
- 保存在全局作用域的变量称为全局变量





# 全局变量的特点

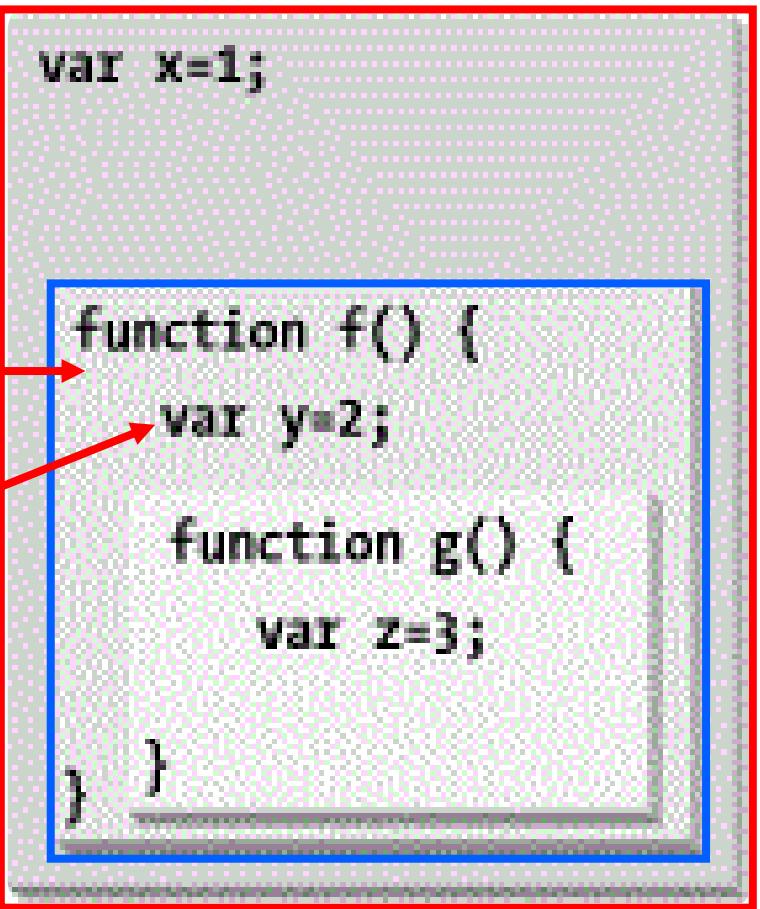
- 优点: 可反复使用 ——————
- 缺点: 全局污染——开发时禁止使用
- 灵魂拷问:
  - 如果全班共用一个喝水杯
  - 你会不会用?
  - 为什么?

```
var x=1;  
  
function f() {  
    var y=2;  
  
    function g() {  
        var z=3;  
  
    }  
}
```



# 函数作用域:

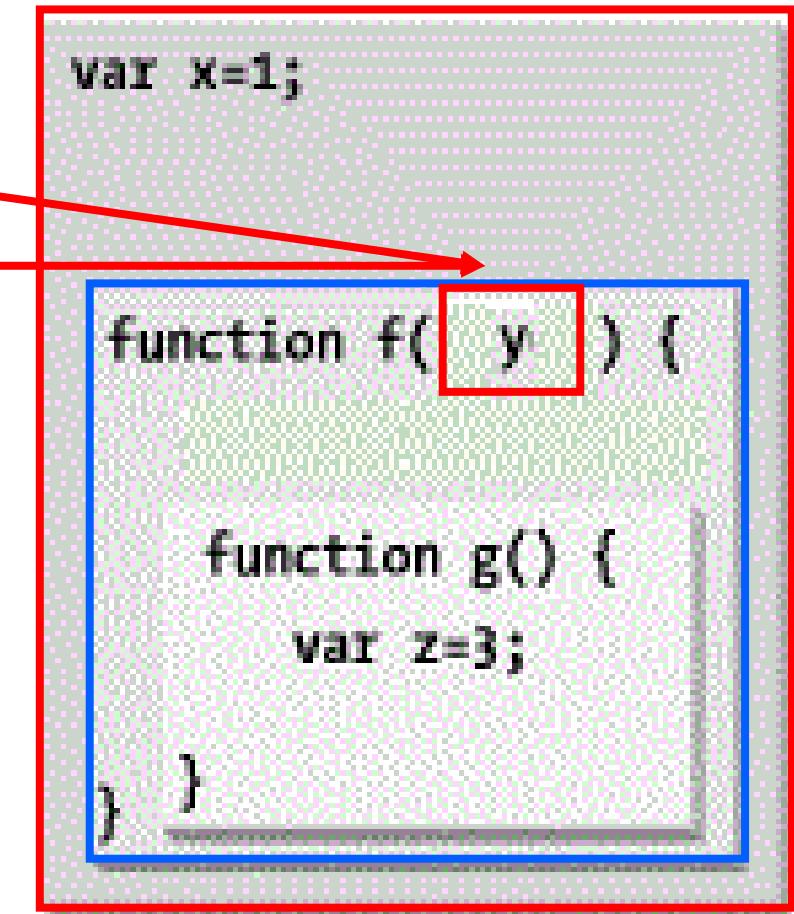
- 一个函数内的范围称为函数作用域
- 保存在函数作用域内的变量称为**局部变量**



# 特例: 形参变量也是函数内的局部变量



- 形参变量虽然没有用var声明
- 但是形参变量也是函数内的局部变量!





# 局部变量的特点

- 优点: 不会被污染
- 缺点: 无法反复使用

```
var x=1;
```

```
function f() {  
    var y=2;  
  
    function g() {  
        var z=3;  
  
    }  
}
```



**强调：只有函数的{}，才能形成作用域**

- 不是所有{}都能形成作用域。
  - 也不是所有{}内的数据都能是局部变量
  - 比如：
    - 对象的{}, 就不是作用域! —————> var lilei={
    - 对象中的属性, 也不是局部变量 —————> sname: "Li Lei",



# 谁轻视原理，谁终将受到惩罚

```
var lilei={  
    sname: "Li Lei",
```

- 为什么这里必须加this?
  - 这个function能不能换成箭头函数?
- ```
intr:function(){  
    console.log("我是${this.sname}")  
}  
}
```

都是因为对象的{}不是作用域，对象中的属性不是局部变量。  
且听下回分解……



# 强调: JS中没有块级作用域

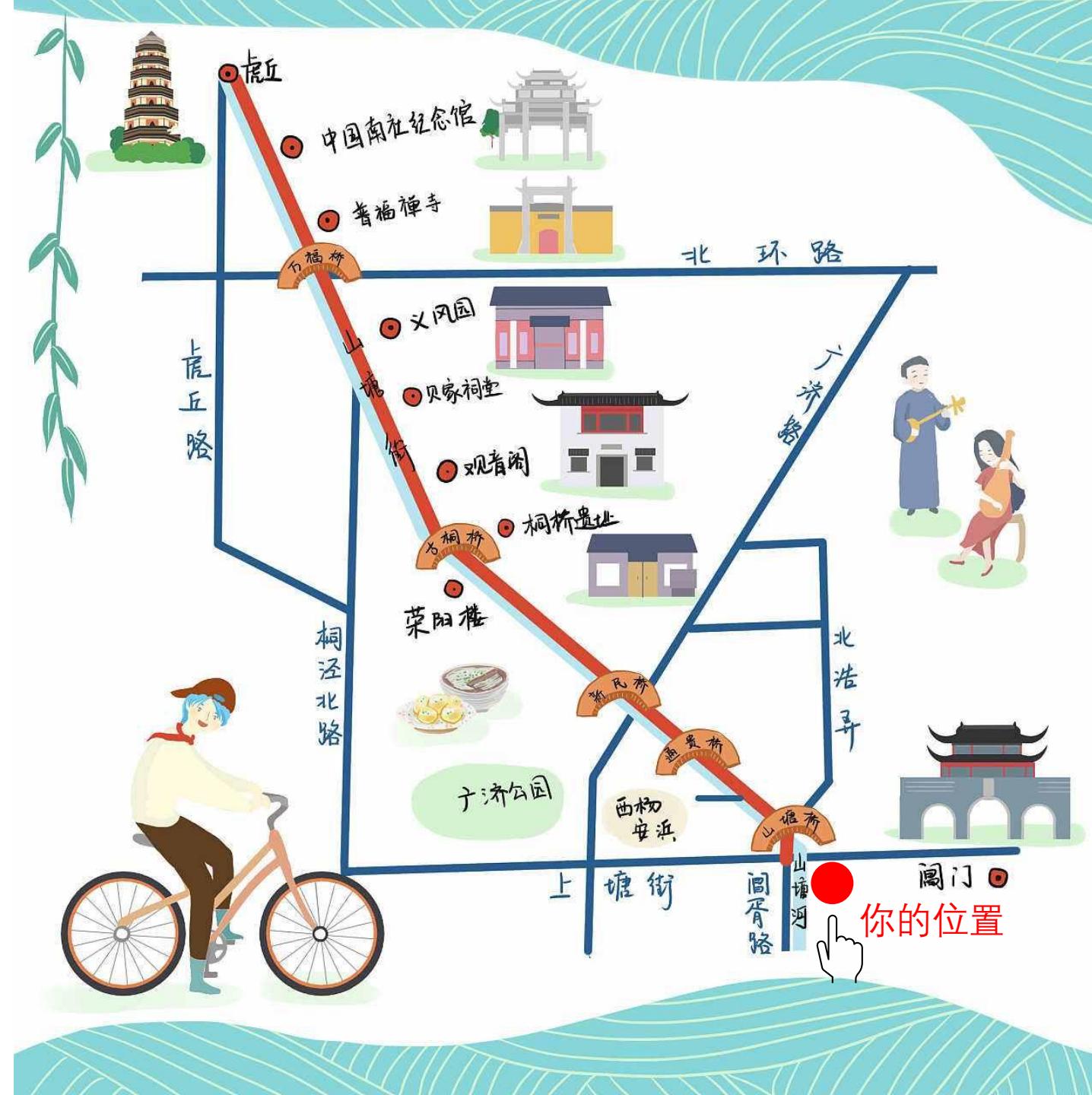
- 除函数{}之外的其余{}, 都不是作用域。
- 都拦不住内部的变量超出{}的范围影响外部程序
- 比如:

```
console.log(a);
if(false){
    var a=10;
}
console.log(a);
```

# 作用域链

(scopes / scope chain)

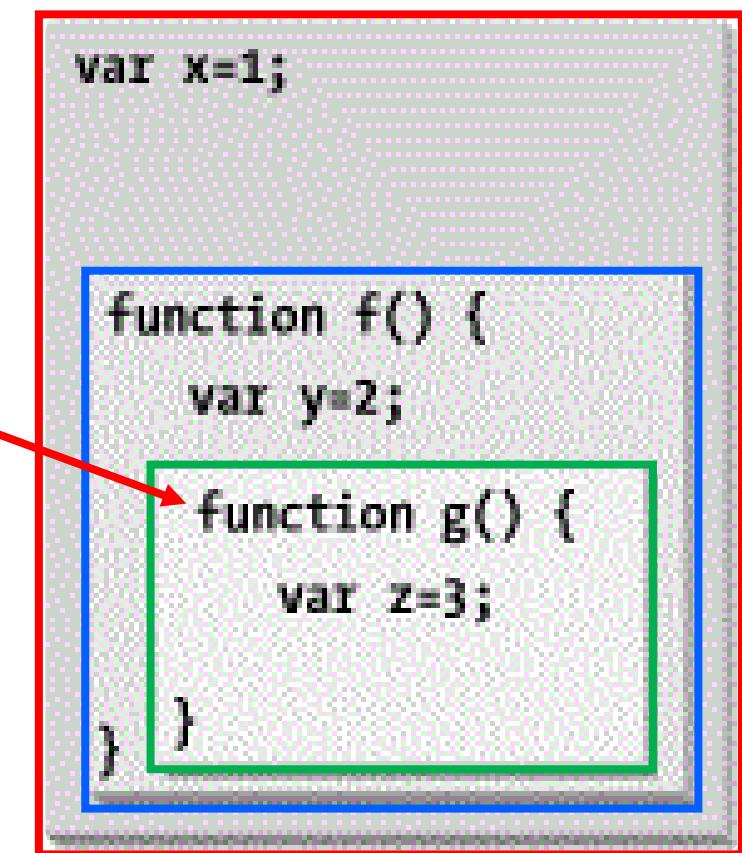
- 因为，js规定，一个函数，既能用自己作用域的变量，又能用外层作用域的变量。
- 所以，需要一个“**路线图**”告诉每个函数，自己都可以去哪里找到想用的变量。
- 就像生活中我们规划旅游景点的游览路线一样。





# 作用域链 (scopes / scope chain)

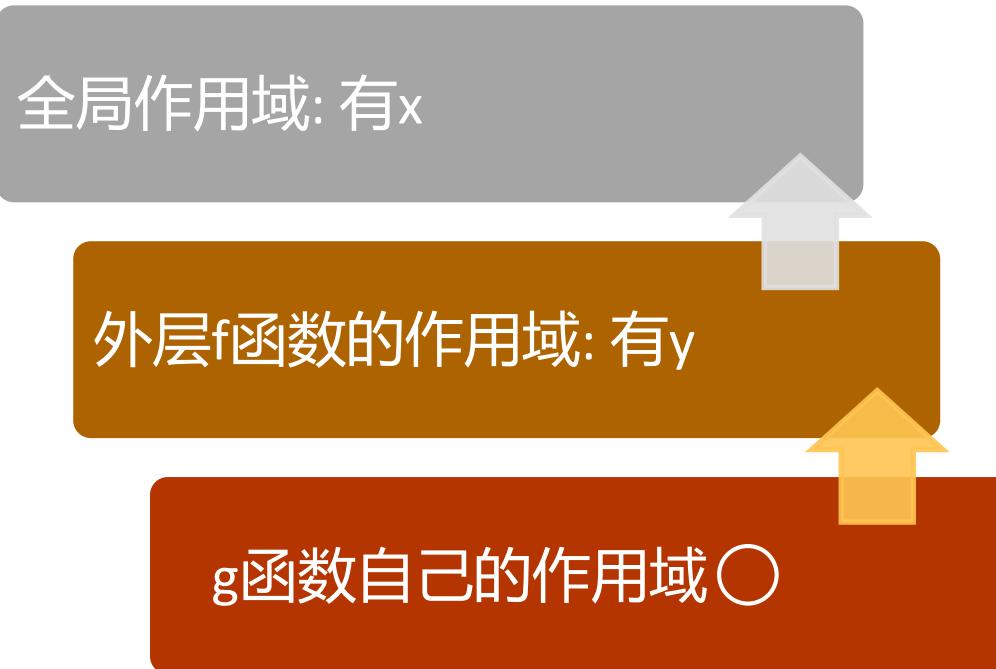
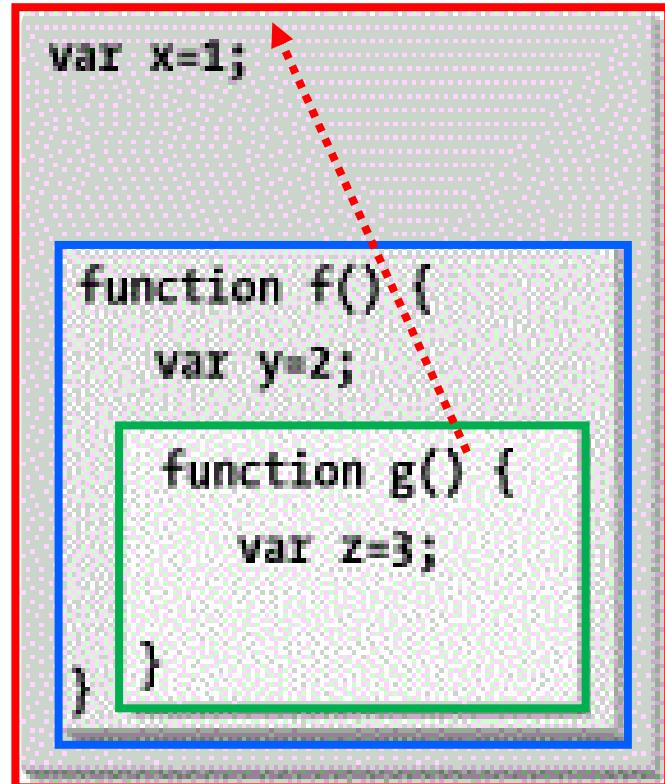
- 其实每个函数**在定义时**，就已经规划好了自己专属的一个查找变量的**路线图**，称为**作用域链**
- 比如：
  - 当我们**定义函数g()**时





# 作用域链 (scopes / scope chain)

- 函数g就为自己规划好了一个**由内向外的查找路线**。
- 以防未来运行时，一旦自己缺少变量，应该去找谁——未雨绸缪。





# 作用域链 (scopes / scope chain)

- 一个函数可用的所有作用域串联起来，就行成了当前函数的作用域链。

```
var x=1;  
  
function f() {  
    var y=2;  
  
    function g() {  
        var z=3;  
    }  
}
```

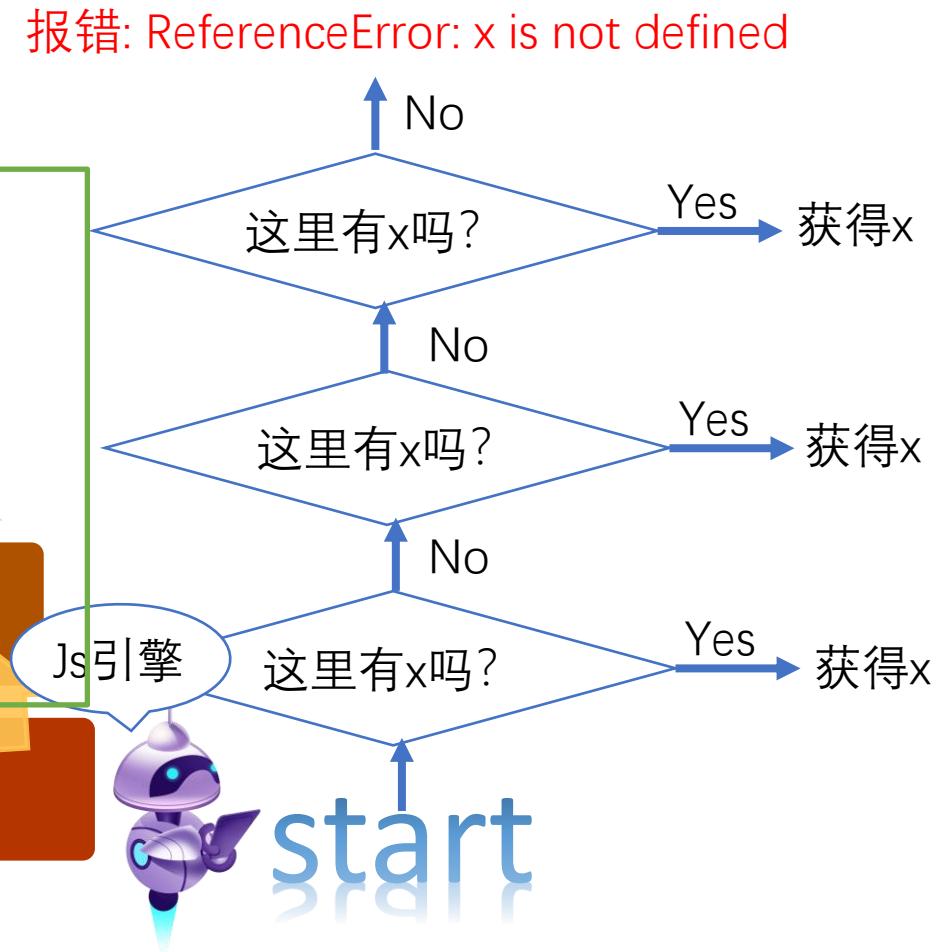




# 作用域链 (scopes / scope chain)

- 当执行到某条语句时，Js引擎会自动延函数的作用域链查找要用的变量，查找路径是这样的：

```
var x=1;  
  
function f() {  
    var y=2;  
  
    function g() {  
        var z=3;  
        console.log(x)  
    }  
}
```

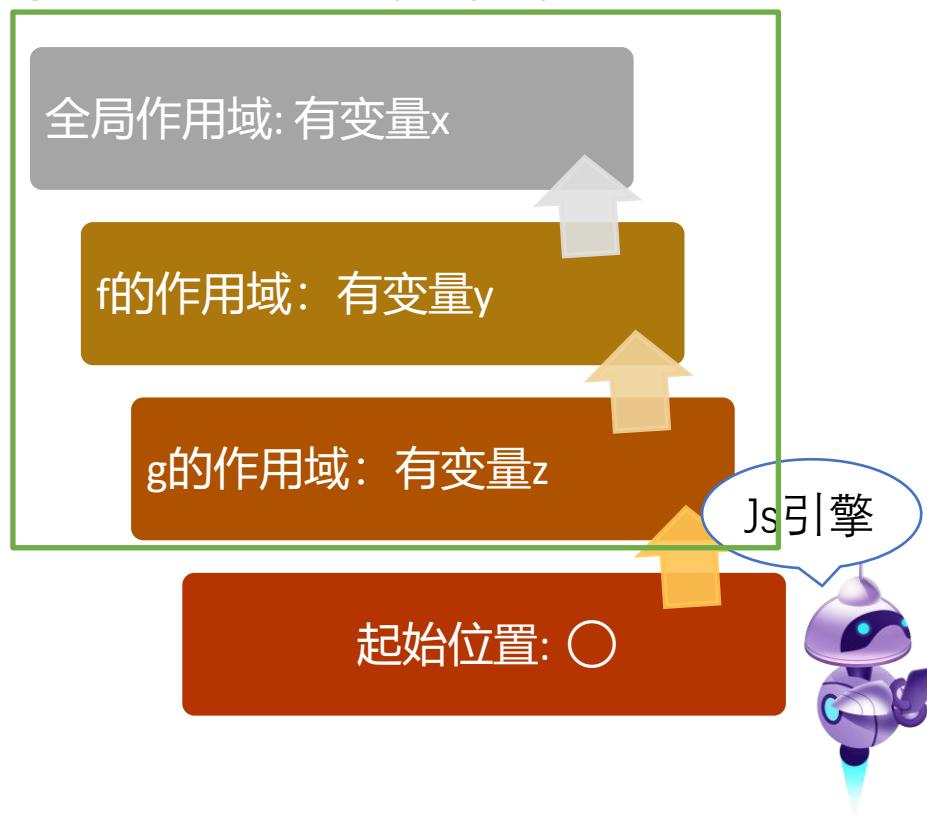
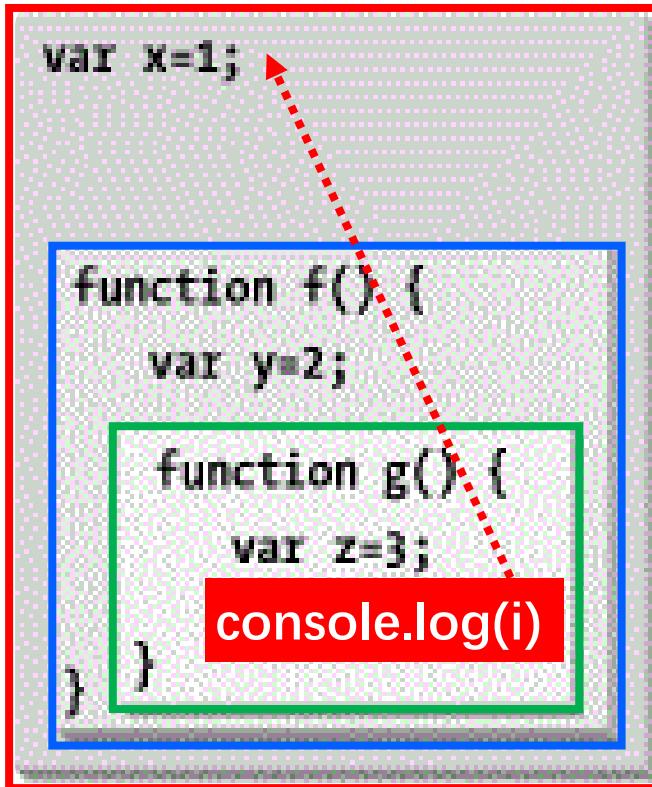




# 作用域链 (scopes / scope chain)

- 如果程序要用变量i，会不会报错？

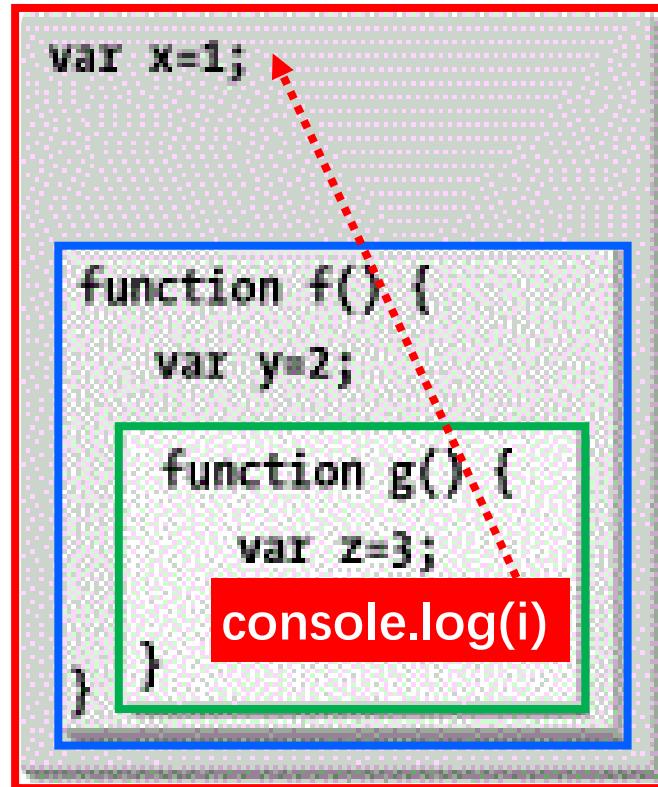
g函数的作用域链(scopes)





# 作用域链 (scopes / scope chain)

- 如果程序要用变量i，则查找逻辑是



g函数的作用域链(scopes)

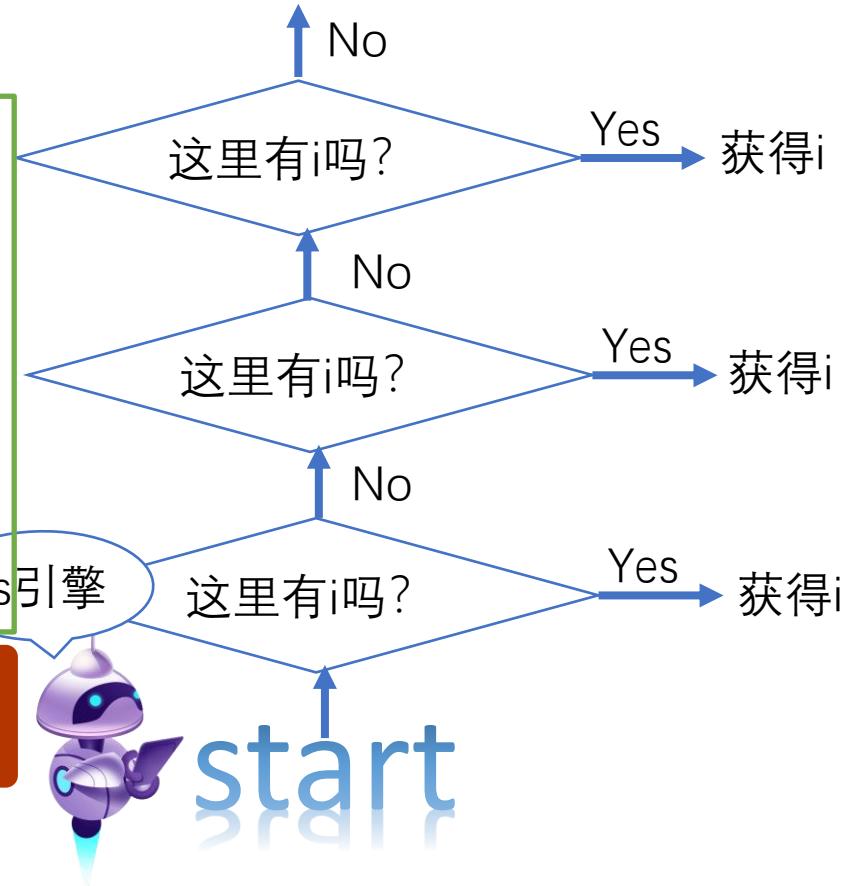
全局作用域: 有变量x

f的作用域: 有变量y

g的作用域: 有变量z

起始位置: ○

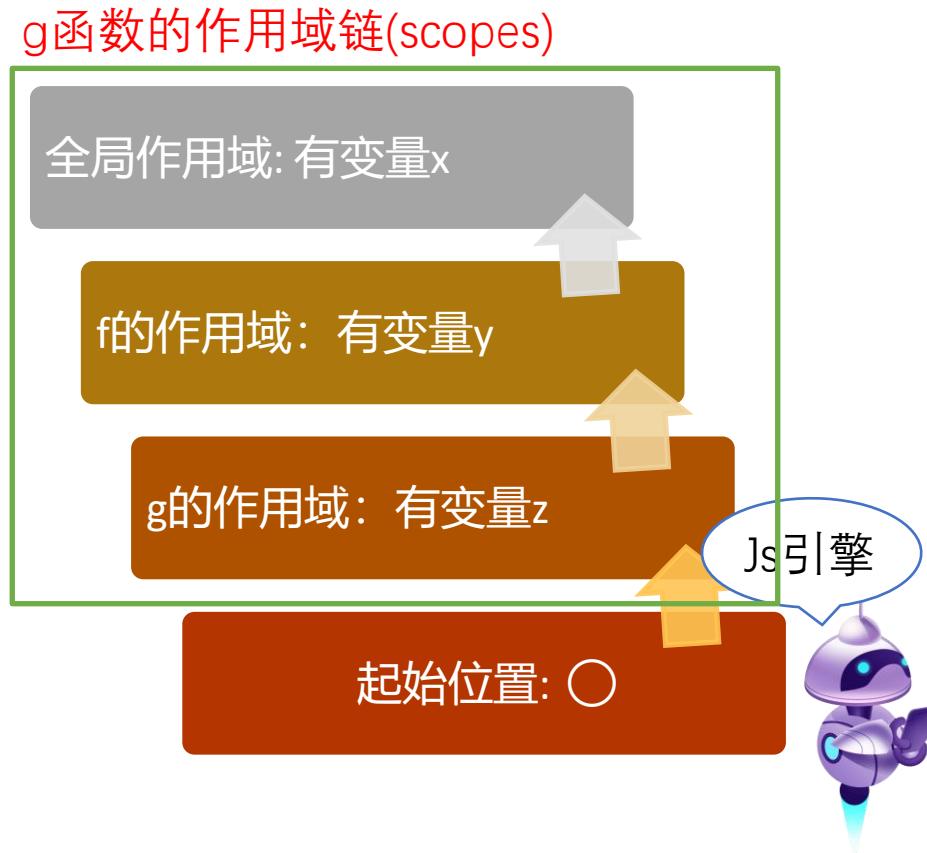
报错: ReferenceError: i is not defined



# 特殊: 给从未声明过的变量赋值

- 如果程序改成给*i*赋值，会不会报错：

```
var x=1;  
  
function f() {  
    var y=2;  
  
    function g() {  
        var z=3;  
        i=10;  
    }  
}
```





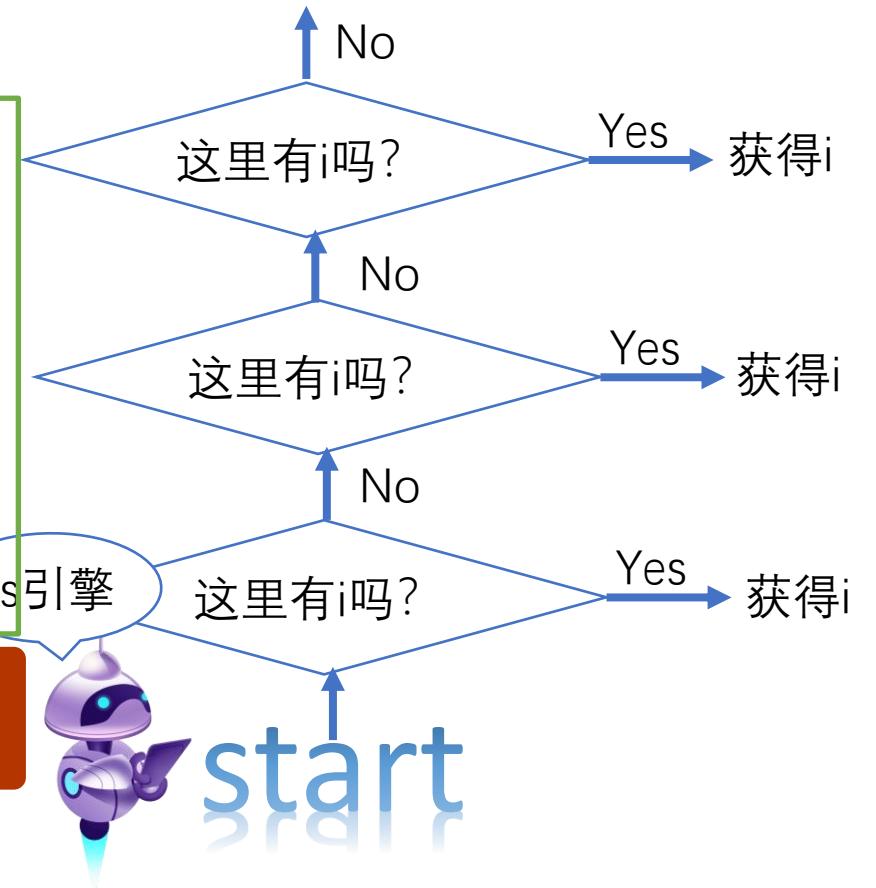
# 特殊: 给从未声明过的变量赋值

- 尝试给从未声明过的变量赋值，逻辑是：

```
var x=1;  
i=10;  
  
function f() {  
    var y=2;  
  
    function g() {  
        var z=3;  
        i=10;  
    }  
}
```



不报错！而是自动在全局创建变量“i”





总结: Js中只有**两种局部变量**:

“

1. 函数内**var**出来的
2. 函数的**形参**变量

**看不见var, 形参里也没有,**

”

**就不是局部变量。**



“做笔试题时：  
把自己当成js引擎。 ”





“

小试牛刀... ...

”





# 第1题: 定义fun函数时:

- var a=10;
- function fun(){
- var a=100;
- a++;
- console.log(a); //?
- }
- fun(); //?
- console.log(a); //?
- 程序的输出结果是?



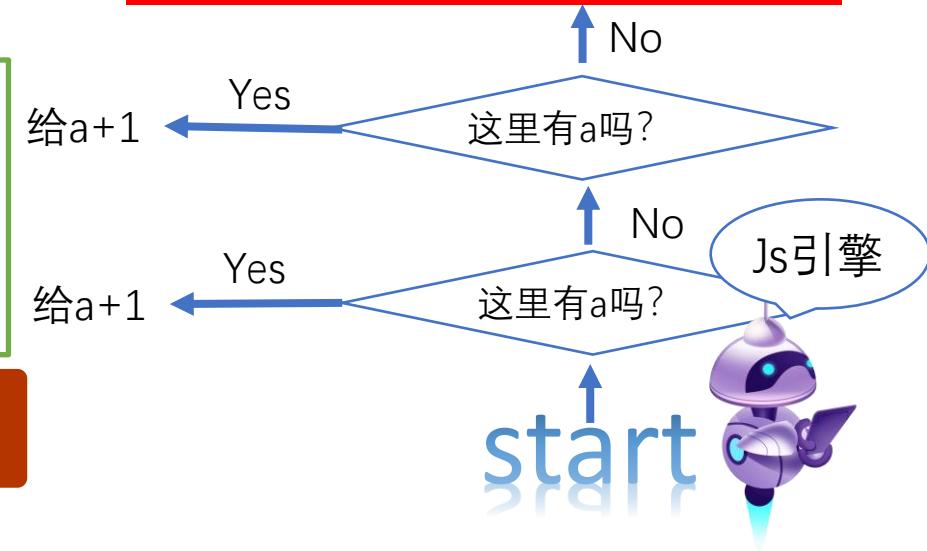
# 第1题：调用fun()函数时



- var a=10;
  - function fun(){
    - var a=100;
    - a++; 
    - console.log(a); //?
  - }
  - fun(); //?
  - console.log(a);
  - 程序的输出结果是？



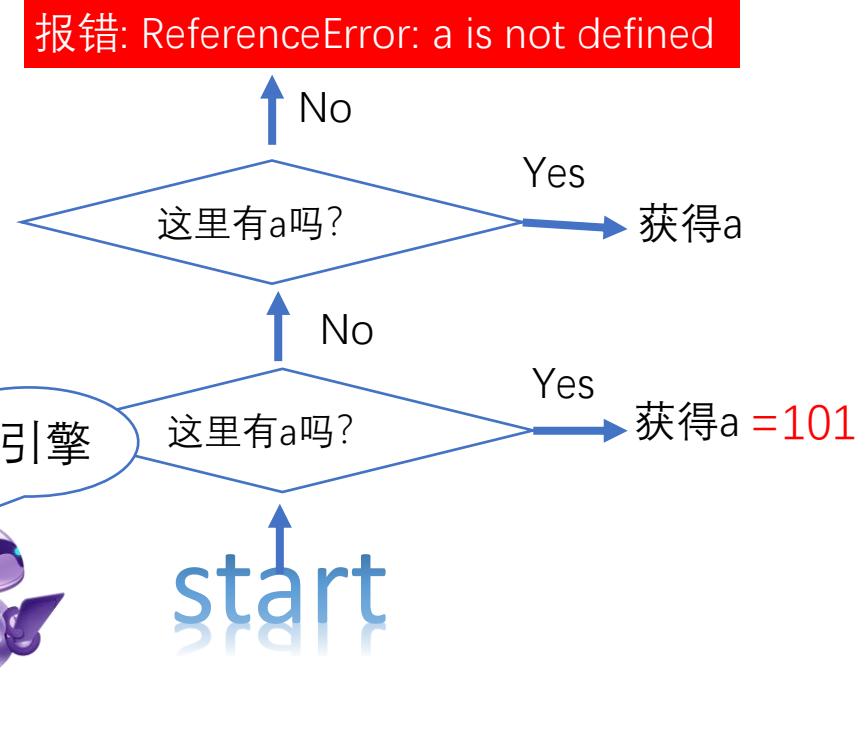
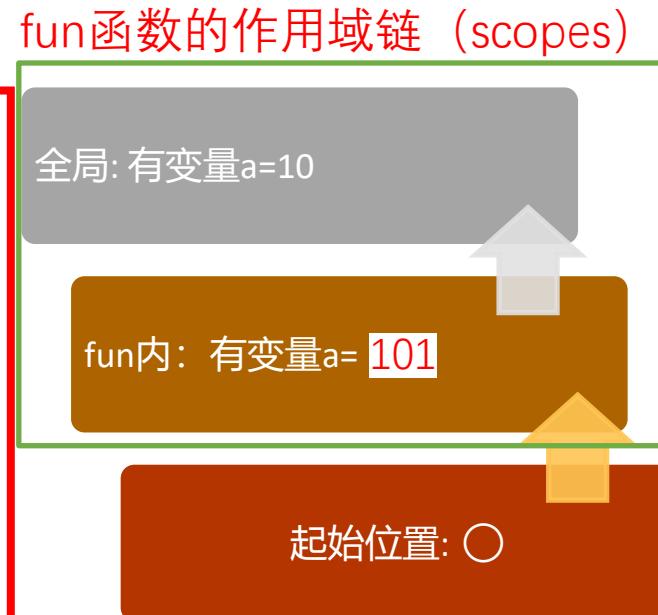
报错: ReferenceError: a is not defined



# 第1题：调用fun()函数时



- var a=10;
- function fun(){
  - var a=100;
  - a++;
  - console.log(a); //? 
- }
- fun(); //? 
- console.log(a);
- 程序的输出结果是?





# 第1题: 调用fun()函数后

```
• var a=10;  
• function fun(){  
•     var a=100;  
•     a++;  
•     console.log(a);  
• }
```

fun(); // 101

• console.log(a); //10

• 程序的输出结果是?

全局: 有变量a=10

fun函数

自己的作用域

fun函数的作用域链 (scopes)

Js引擎





# 第2题: 定义fun函数时:

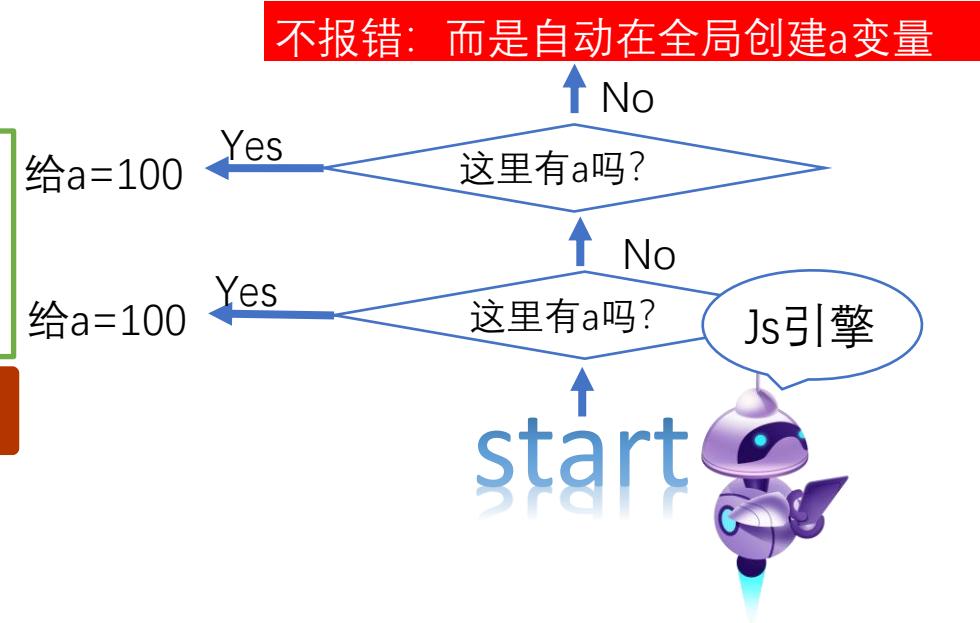
- var a=10;
- function fun(){  
    a=100;  
    a++;  
    console.log(a); //?  
}
- fun(); //?
- console.log(a); //?
- 程序的输出结果是?



问题:  
fun函数的作用域中有  
没有局部变量?

# 第2题: 调用fun()函数时

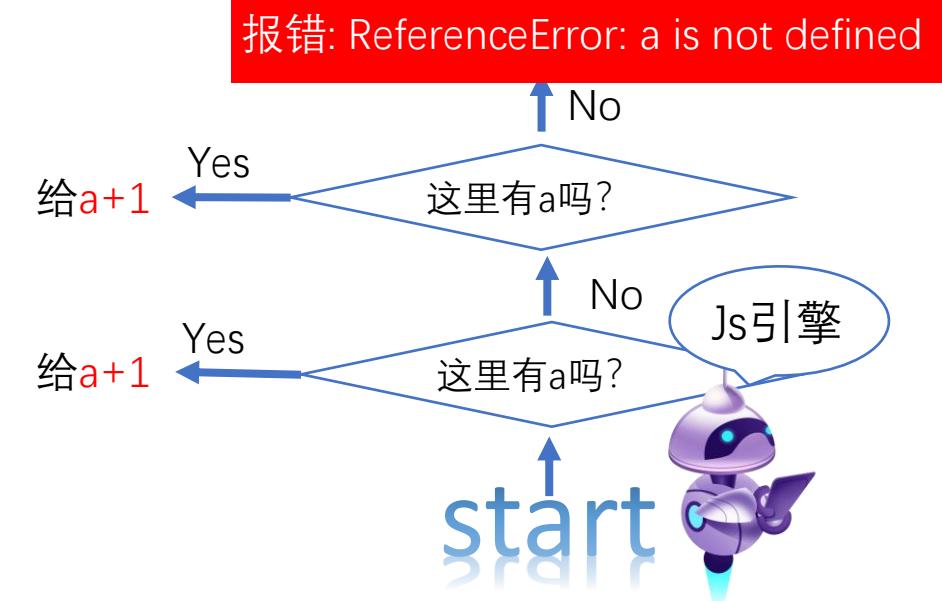
- var a=10;
- function fun(){
  - a=100; ←
  - a++;
  - console.log(a); //?
- }
- fun(); //?
- console.log(a);
- 程序的输出结果是?





# 第2题: 调用fun()函数时

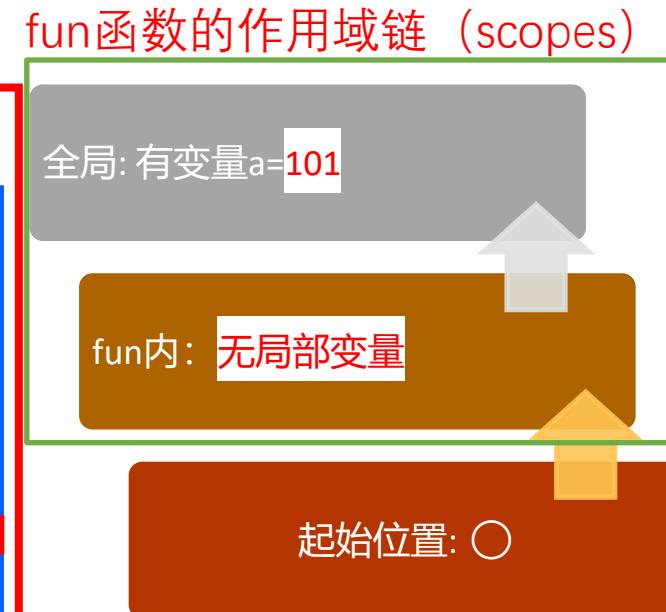
- var a=10;
- function fun(){  
    a=100;  
    a++; ←  
    console.log(a); //?  
}
- fun(); //?
- console.log(a);
- 程序的输出结果是?



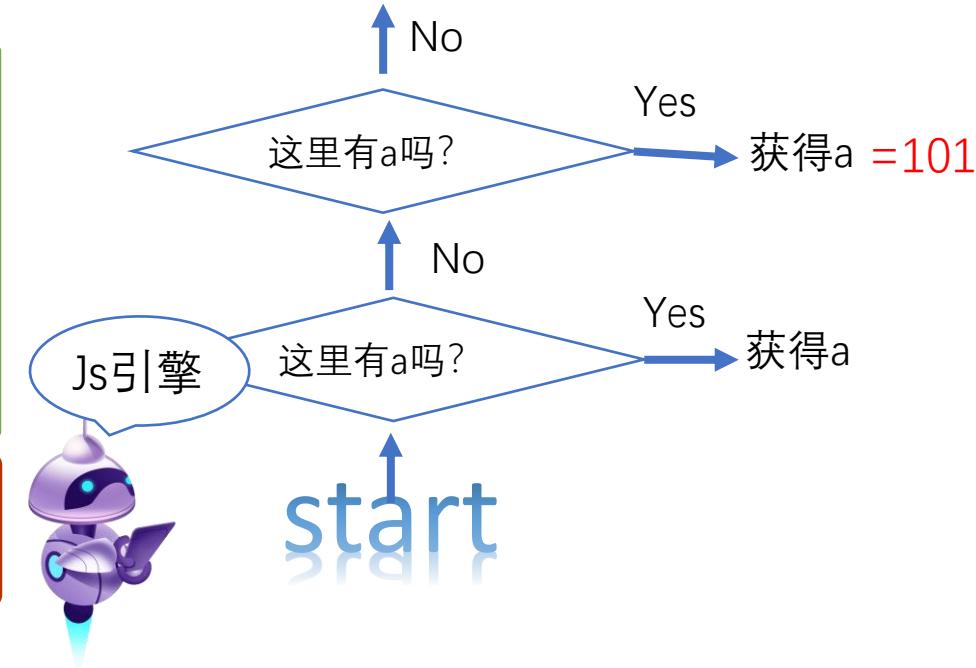


# 第2题: 调用fun()函数时

- var a=10;
- function fun(){  
    a=100;  
    a++;  
    console.log(a); //? 101  
}
- fun(); //?
- console.log(a);
- 程序的输出结果是?



报错: ReferenceError: a is not defined





# 第2题: 调用fun()函数后

```
• var a=101;  
• function fun(){  
•     a=100;  
•     a++;  
•     console.log(a);  
• }
```

```
fun(); // 101
```

```
• console.log(a); //101
```



• 程序的输出结果是?

全局: 有变量a=101

fun函数

自己的作用域

fun函数的作用域链 (scopes)



# 第3题: 定义fun函数时:

- var a=10;
- function fun(a){  
    •     a++;  
    •     console.log(a);  
    • }
- fun(a); //?
- console.log(a); //?
- 程序的输出结果是?

全局: 有变量a=10

fun函数

自己的作用域

fun函数的作用域链 (scopes)

问题:  
fun函数的作用域中有  
没有局部变量?



# 第3题: 调用fun()函数时

```
var a=10;  
• function fun(a){ ←  
•     a++;  
•     console.log(a); //?  
• }  
  
fun(a); //?  
• console.log(a);  
• 程序的输出结果是?
```



强调:

- 形参变量也是函数的局部变量
- 函数传参采用的是按值传递。
- 原始类型的值，在传参时，是将原变量的值复制一个副本给函数形参变量。
- 所以，在函数内，修改形参变量，不影响外部原变量的值。
- 所以，此时内存中有两个10

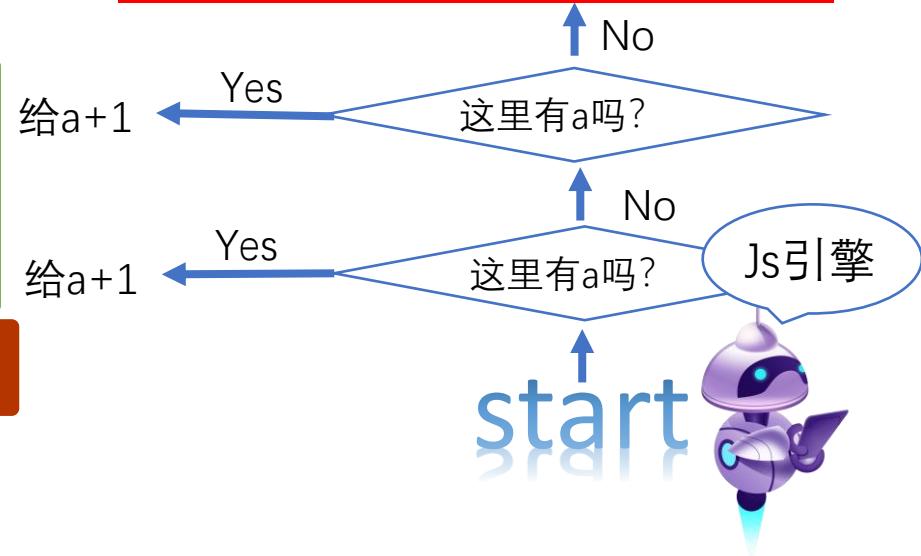


# 第3题: 调用fun()函数时

- var a=10;
- function fun(a){  
    • a++; ←  
    • console.log(a); //?  
  • }
- fun(a); //?
- console.log(a);
- 程序的输出结果是?



报错: ReferenceError: a is not defined



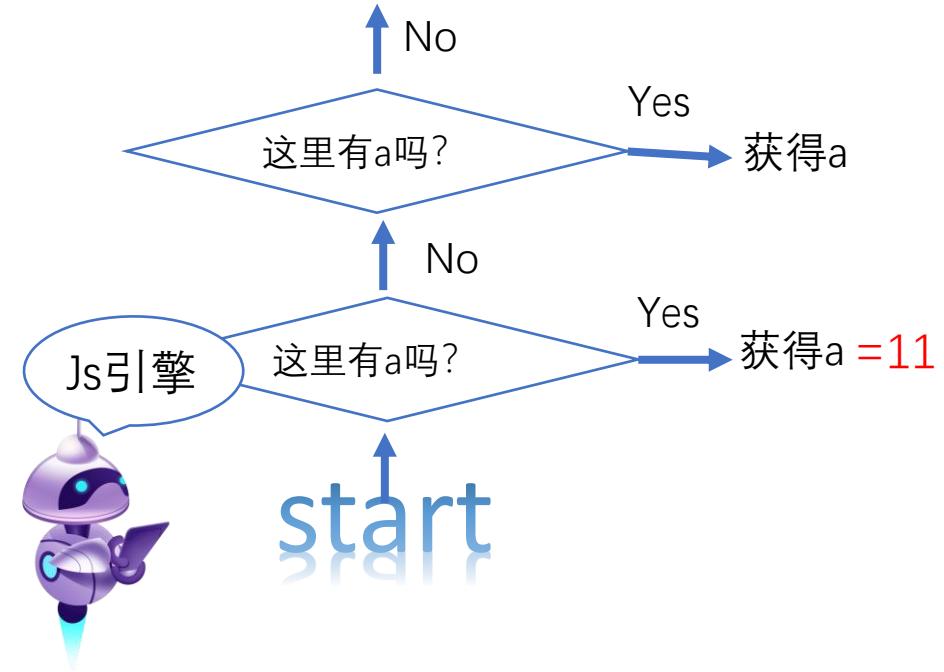


# 第3题: 调用fun()函数时

- var a=10;
- function fun(a){
  - a++;
  - console.log(a); //? 11
- }
- fun(); //?
- console.log(a);
- 程序的输出结果是?



报错: ReferenceError: a is not defined





# 第3题: 调用fun()函数后

```
• var a=10;  
•  
• function fun(a){  
•     a++;  
•     console.log(a);  
• }
```

fun(a); // 11

• console.log(a); //10

• 程序的输出结果是?

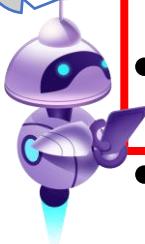
全局: 有变量a=10

fun函数

自己的作用域

fun函数的作用域链 (scopes)

Js引擎





“但是，  
只了解到这个程度，  
不足以理解闭包！”





# Q3：作用域的本质

# JS中，作用域和作用域链都是对象结构



- 耳听为虚，眼见为实

# JS中，作用域和作用域链都是对象结构



- 第1步: 新建名为1\_scopes.html的网页, 其中编写刚才第1题程序:

```
var a=10;  
function fun(){  
    var a=100;  
    a++;  
    console.log(a);  
}  
fun();  
console.log(a);
```

# JS中，作用域和作用域链都是对象结构



- 第2步: 用浏览器运行，并按F12打开控制台

The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The console output area displays the following:

```
101
10
> |
```

The first two entries are linked to the file `1_scopes.html`: one at line 17 and another at line 20. The third entry is preceded by a blue arrow, indicating it is the current input line.

# JS中，作用域和作用域链都是对象结构



- 第3步: 然后, 点击js程序每一行左侧的行号, 添加断点

The screenshot shows the Chrome DevTools Sources tab with the file '1\_scopes.html' selected. The code editor displays the following JavaScript code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<script>
    var a = 10;
    function fun() {
        var a = 100;
        a++;
        console.log(a);
    }
    fun();
    console.log(a);
</script>
<!-- Code injected by live-server -->
<script type="text/javascript">
    // <![CDATA[ <-- For SVG support
    if ('WebSocket' in window) {
        (function () {
            function refreshCSS() {
                var sheets = [...].slice.call(document.getElementsByTagName("link"));
                var head = document.getElementsByTagName("head")[0];
                for (var i = 0; i < sheets.length; ++i) {
```

The right panel shows the Breakpoints sidebar with several breakpoints set across the file. The breakpoints are located at:

- Line 13: `var a = 10;`
- Line 15: `var a = 100;`
- Line 16: `a++;`
- Line 17: `console.log(a);`
- Line 18: `}`
- Line 19: `fun();`
- Line 20: `console.log(a);`
- Line 22: `// <![CDATA[ <-- For SVG support`
- Line 24: `if ('WebSocket' in window) {`
- Line 26: `(function () {`
- Line 27: `function refreshCSS() {`
- Line 28: `var sheets = [...].slice.call(document.getElementsByTagName("link"));`
- Line 29: `var head = document.getElementsByTagName("head")[0];`
- Line 30: `for (var i = 0; i < sheets.length; ++i) {`

# JS中，作用域和作用域链都是对象结构



- 第4步：刷新页面，收起右侧断点一栏，展开作用域一栏，同时展开全局，
- 你看到了什么？

已在调试程序... ⏪ ⏹ 元素 控制台 来源 网络 性能 内存 应用 安全 Lighthouse

左上角显示：  
正在调试程序

1 \_scopes.html x

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<script>
var a = 10;
function fun() {
    var a = 100;
    a++;
    console.log(a);
}
fun();
console.log(a);
</script>

<script type="text/javascript">
// <![CDATA[ <-- For SVG :
if ('WebSocket' in window) {
    (function () {
        function refreshCS() {
            var sheets = [];
            var head = document.head;
            for (var i = 0; i < head.length; i++) {
                var elem = head[i];
                var parent = elem.parentElement;
                parent.removeChild(elem);
                var rel = elem.getAttribute('rel');
                if (elem.href) {
                    var url = new URL(elem.href);
                    elem.href = url.origin + url.pathname;
                }
                parent.appendChild(elem);
            }
        }
        var protocol = window.location.protocol;
        var address = protocol + window.location.host;
        var socket = new WebSocket(address);
        socket.onmessage = function (event) {
            var data = JSON.parse(event.data);
            if (data.type === 'refresh') {
                refreshCS();
            }
        };
    })();
}
var protocol = window.location.protocol;
var address = protocol + window.location.host;
var socket = new WebSocket(address);
socket.onmessage = function (event) {
    var data = JSON.parse(event.data);
    if (data.type === 'refresh') {
        refreshCS();
    }
};
</script>
```

在遇到异常时暂停  
已在断点暂停  
▶ 监视  
▶ 断点  
▼ 作用域  
▼ 全局  
a: undefined  
► alert: f alert()  
► atob: f atob()  
► blur: f blur()  
► btoa: f btoa()  
► caches: CacheStorage {}  
► cancelAnimationFrame: f cancelAnimationFrame()  
► cancelIdleCallback: f cancelIdleCallback()  
► captureEvents: f captureEvents()  
► chrome: {loadTimes: f, csi: f}  
► clearInterval: f clearInterval()  
► clearTimeout: f clearTimeout()  
► clientInformation: Navigator {vendorSub: '', productSub: '20030107', vendor: 'Google Inc.', maxTouchPoints: 0, userActivation: UserActivationState.ACTIVE}  
► close: f close()  
closed: false  
► confirm: f confirm()  
► cookieStore: CookieStore {onchange: null}  
► createImageBitmap: f createImageBitmap()  
crossOriginIsolated: false  
► crypto: Crypto {subtle: SubtleCrypto}  
► customElements: CustomElementRegistry {}  
defaultStatus: ""  
defaultStatus: ""  
devicePixelRatio: 1.75  
► document: document  
► external: External {}  
► fetch: f fetch()  
► find: f find()  
► focus: f focus()  
frameElement: null  
► frames: Window {window: Window, self: Window, document: document, name: '', location: Location, ...}  
► fun: f fun()  
► getComputedStyle: f getComputedStyle()  
► getSelection: f getSelection()

全局作用域  
其实是一个  
名为widow的对象结构

已在调试程序... | ⏪ ⏹ 元素 控制台 来源 网络 性能 内存 应用 安全 Lighthouse | ⚙️ ⋮ ×

网页 文件系统 » 1\_scopes.html x

1 \_scopes.html x

1 <!DOCTYPE html>  
2 <html lang="en">  
3  
4 <head>  
5 <meta charset="UTF-8">  
6 <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />  
7 <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
8 <title>Document</title>  
9 </head>  
10 <body>  
11 <script>  
12     var a = 10;  
13     function fun() {  
14         var a = 100;  
15     }  
16     // <![CDATA[ <>-- For SVG :  
17     if ('WebSocket' in window) {  
18         (function () {  
19             function refreshSheets() {  
20                 var sheets = document.styleSheets;  
21                 var head = document.head || document.documentElement;  
22                 for (var i = 0; i < sheets.length; i++) {  
23                     var elem = sheets[i].rules[0];  
24                     var parent = elem.parentRule;  
25                     parent.removeRule(elem);  
26                     var rel = elem.getAttribute('rel');  
27                     if (elem.href) {  
28                         var url = elem.href;  
29                         elem.href = url + '?t=' + new Date().getTime();  
30                     }  
31                     parent.appendChild(elem);  
32                 }  
33             parent.appendChild(sheet);  
34         }  
35         refreshSheets();  
36         setInterval(refreshSheets, 1000);  
37     }  
38     }  
39     parent.appendChild(sheet);  
40     }  
41     }  
42     var protocol = window.location.protocol;  
43     var address = protocol + '//' + window.location.host;  
44     var socket = new WebSocket(address);  
45     socket.onmessage = function (event) {  
46         var message = event.data;

我们定义的全局变量和全局函数都是window对象的成员

① 已在断点暂停

▶ 监视

▶ 断点

▼ 作用域

▶ 全局

a: undefined

▶ alert: f alert()  
▶ atob: f atob()  
▶ blur: f blur()  
▶ btoa: f btoa()  
▶ caches: CacheStorage {}  
▶ cancelAnimationFrame: f cancelAnimationFrame()  
▶ cancelIdleCallback: f cancelIdleCallback()  
▶ captureEvents: f captureEvents()  
▶ chrome: {loadTimes: f, csi: f}  
▶ clearInterval: f clearInterval()  
▶ clearTimeout: f clearTimeout()  
▶ clientInformation: Navigator {vendorSub: '...', productSub: '20030107', vendor: 'Google Inc.', maxTouchPoints: 0, userActivation: UserActivationState.ACTIVE}  
▶ close: f close()  
▶ closed: false  
▶ confirm: f confirm()  
▶ cookieStore: CookieStore {onchange: null}  
▶ createImageBitmap: f createImageBitmap()  
▶ crossOriginIsolated: false  
▶ crypto: Crypto {subtle: SubtleCrypto}  
▶ customElements: CustomElementRegistry {}  
▶ defaultStatus: ""  
▶ defaultStatusText: ""  
▶ devicePixelRatio: 1.75  
▶ document: document  
▶ external: External {}  
▶ fetch: f fetch()  
▶ find: f find()  
▶ focus: f focus()  
▶ frameElement: null  
▶ frames: Window {window: Window, self: Window, document: document, name: "", location: Location, ...}  
▶ fun: f fun()  
▶ getComputedStyle: f getComputedStyle()  
▶ getSelection: f getSelection()

又因为程序开始执行时先执行声明提前操作暂时未赋值所以，暂时有a变量和fun函数。但是，a变量值暂时为undefined



“**总结: 全局作用域**  
**其实是一个名为window的对象**  
**所有全局变量和全局函数都是**  
**window对象的成员。**”

# JS中，作用域和作用域链都是对象结构



- 第5步: 展开fun函数, 看scopes作用域链部分, 你又发现了什么?

## ▼ 全局

Window

```
a: undefined
▶ alert: f alert()
▶ atob: f atob()
▶ blur: f blur()
▶ btoa: f btoa()
▶ caches: CacheStorage {}
▶ cancelAnimationFrame: f cancelAnimationFrame()
▶ cancelIdleCallback: f cancelIdleCallback()
▶ captureEvents: f captureEvents()
▶ chrome: {loadTimes: f, csi: f}
▶ clearInterval: f clearInterval()
▶ clearTimeout: f clearTimeout()
▶ clientInformation: Navigator {vendorSub: '', productSub: '20030107', vendor: 'Google Inc.', maxTouchPoints: 0, userActivation: UserActivation, ...}
▶ close: f close()
closed: false
▶ confirm: f confirm()
▶ cookieStore: CookieStore {onchange: null}
▶ createImageBitmap: f createImageBitmap()
crossOriginIsolated: false
▶ crypto: Crypto {subtle: SubtleCrypto}
▶ customElements: CustomElementRegistry {}
defaultStatus: ""
defaultstatus: ""
devicePixelRatio: 1.75
▶ document: document
▶ external: External {}
▶ fetch: f fetch()
▶ find: f find()
▶ focus: f focus()
frameElement: null
▶ frames: Window {window: Window, self: Window, document: Document}
▶ fun: f fun()
arguments: null
caller: null
length: 0
name: "fun"
▶ prototype: {constructor: f}
[[FunctionLocation]]: 1_scopes.html:14
▶ [[Prototype]]: f ()
▶ [[Scopes]]: Scopes[1]
▶ 0: Global {window: Window, self: Window, document: document, name: '', location: Location, ...}
```

因为js引擎还未调用fun函数  
所以fun函数的作用域链上暂时只  
有一个作用域Global, 其实就是w  
全局作用域对象window



# JS中，作用域和作用域链都是对象结构



- 第6步: 点2下左上角蓝色向右三角, 让程序运行到fun函数内第一条语句位置, 此时js引擎已经开始调用fun函数
- 收起右侧全局作用域对象, 发现什么?

已在调试程序... | 元素 | 控制台 | 来源 | 网络 | 性能 | 内存 | 应用 | 安全 | Lighthouse | : | X

网页 » : 1\_scopes.html x

top

127.0.0.1: 1\_scope

13 var a = 10;  
14 function fun() {  
15 var a = 100;  
16 a++;  
17 console.log(a);  
18 }  
19 fun();  
20 console.log(a);

在遇到异常时暂停

监视

断点

作用域

本地

- this: Window
- a: undefined

全局

- a: 10
- alert: f alert()
- atob: f atob()
- blur: f blur()
- btoa: f btoa()
- caches: CacheStorage {}
- cancelAnimationFrame: f cancelAnimationFrame()
- cancelIdleCallback: f cancelIdleCallback()
- captureEvents: f captureEvents()

live-server  
vascript  
For SVG  
n window)

• 因为js引擎进入函数内部开始调用函数  
• 所以，作用域链中变为两级作用域。  
• 临时新增了一级"本地(local)"作用域。  
• 其实就是函数作用域。

15 行, 第 15 列 覆盖率: 不适用

# JS中，作用域和作用域链都是对象结构



- 第7步: 展开右侧本地(函数)作用域, 你看到了什么?

网页 > : 1\_scopes.html

- 函数作用域其实是一个js引擎临时创建的对象，只不过没有对象名。
- 函数作用域对象中保存着函数内所有局部变量。
- this，竟然也是保存在函数作用域对象中的。（后边面向对象课程会用）
- 而且，this->全局作用域对象window

{ } 第 15 行, 第 15 列 覆盖率: 不适用

```
9  </head>
10
11 <body>
12   <script>
13     var a = 10;
14     function fun() {
15       var a = 100;
16       a++;
17       console.log(a);
18     }
19     fun();
20   <script>
```

在遇到异常时暂停

监视

断点

作用域

本地

- this: Window
- a: undefined

全局

- a: 10
- alert: f alert()
- atob: f atob()
- blur: f blur()
- btoa: f btoa()
- caches: CacheStorage {}
- cancelAnimationFrame: f cancelAnimationFrame()
- cancelIdleCallback: f cancelIdleCallback()
- captureEvents: f captureEvents()

# JS中，作用域和作用域链都是对象结构



- 第8步：连续点击左上角蓝色向右箭头，让调试工具运行到整段程序组后一句，观察作用域的变化和fun函数对象的scopes作用域链的变化。

```
var a=10;  
function fun(){  
    var a=100;  
    a++;  
    console.log(a);  
}  
fun();  
console.log(a);
```



已在调试程序... | ⏪ ⏹

元素 控制台 来源 网络 性能 内存 应用 安全 Lighthouse | ⚙️ ⋮ X

网页 » : 1\_scopes.html x

top  
127.0.0.1:  
1\_scope

```
9 </head>
10
11 <body>
12   <script>
13     var a = 10;
14     function fun() {
15       var a = 100; a = 100
16       a++;
17       console.log(a);
18     }
19     fun();
20     console.log(a);
21   </script>
22   
23   <script type="text/javascript">
24     // <![CDATA[ <-- For SVG
25     if ('WebSocket' in window)
26       (function () {
27         function refreshCS
28           var sheets = [
```

第 16 行, 第 7 列 覆盖率: 不适用

▶ 在遇到异常时暂停

▶ 断点

▼ 作用域

▼ 本地

- ▶ this: Window
- ▶ a: 100

▶ 全局

▶ 调用堆栈

- ▶ fun 1\_scopes.html:16
- ▶ (匿名) 1\_scopes.html:19

▶ XHR/提取断点

▶ DOM 断点

▶ 全局监听器

▶ 事件监听器断点

▶ CSP 违规断点

已在调试程序... | ⏪ ⏹ | 元素 | 控制台 | 来源 | 网络 | 性能 | 内存 | 应用 | 安全 | Lighthouse | ⚙ | ⋮ | X

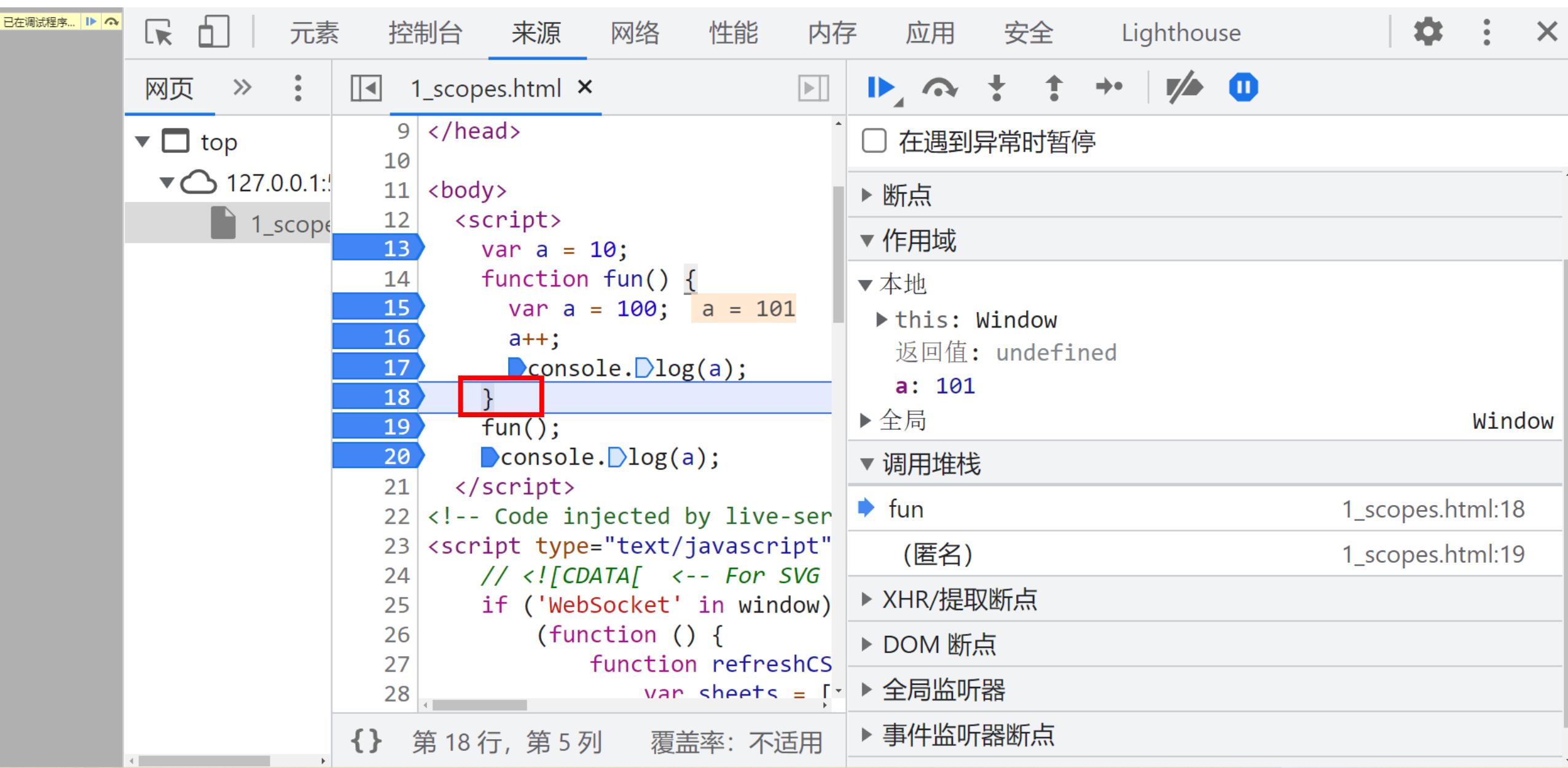
网页 » : 1\_scopes.html x

▶ top  
127.0.0.1:  
1\_scope

9 </head>  
10  
11 <body>  
12 <script>  
13 var a = 10;  
14 function fun() {  
15 var a = 100; a = 101  
16 a++;  
17 console.log(a);  
18 }  
19 fun();  
20 console.log(a);  
21 </script>  
22 <!-- Code injected by live-server -->  
23 <script type="text/javascript"  
24 // <![CDATA[ <-- For SVG  
25 if ('WebSocket' in window)  
26 (function () {  
27 function refreshCS  
28 var sheets = [

▶ 在遇到异常时暂停  
▶ 断点  
▼ 作用域  
▼ 本地  
▶ this: Window  
▶ a: 101  
▶ 全局  
▶ 调用堆栈  
▶ fun 1\_scopes.html:17  
▶ (匿名) 1\_scopes.html:19  
▶ XHR/提取断点  
▶ DOM 断点  
▶ 全局监听器  
▶ 事件监听器断点  
▶ CSP 违规断点

{ } 第 17 行, 第 7 列 覆盖率: 不适用



已在调试程序... | ▶ ⏪ ⏴ 元素 控制台 来源 网络 性能 内存 应用 安全 Lighthouse | ⚙ : X

网页 文件系统 替换 » :

1\_scopes.html x

top

127.0.0.1:5500

1\_scopes.html

```
9 </head>
10
11 <body>
12   <script>
13     var a = 10;
14     function fun() {
15       var a = 100;
16       a++;
17       console.log(a);
18     }
19     fun();
20     console.log(a);
21   </script>
22 <!-- Code injected by live-server -->
```

在遇到异常时暂停

作用域

全局

a: 10

Window

alert: f alert()

atob: f atob()

blur: f blur()

btoa: f btoa()

caches: CacheStorage {}

cancelAnimationFrame: f cancelAnimationFrame()

cancelIdleCallback: f cancelIdleCallback()

captureEvents: f captureEvents()

chrome: {loadTimes: f, csi: f}

clearInterval: f clearInterval()

clearTimeout: f clearTimeout()

clientInformation: Navigator {vendorSub: '', productSub: '20030107', vendor: 'Google Inc.', maxTouchPoints: 0, userActivation: UserActivation, ...}

close: f close()

closed: false

confirm: f confirm()

cookieStore: CookieStore {onchange: null}

createImageBitmap: f createImageBitmap()

crossOriginIsolated: false

crypto: Crypto {subtle: SubtleCrypto}

customElements: CustomElementRegistry {}

defaultStatus: ""

defaultstatus: ""

devicePixelRatio: 1.75

document: document

external: External

fetch: f fetch()

find: f find()

focus: f focus()

frameElement: null

frames: Window {window: Window, self: Window, document: document, name: '', location: Location, ...}

fun: f fun()

arguments: null

caller: null

length: 0

name: "fun"

prototype: {constructor: f}

[[FunctionLocation]]: 1\_scopes.html:14

[[DontType]]: f ()

[[Scopes]]: Scopes[1]

0: Global {window: Window, self: Window, document: document, name: '', location: Location, ...}

getComputedStyle: f getComputedStyle()

getSelection: f getSelection()

history: History {length: 1, scrollRestoration: 'auto', state: null}

第 20 行, 第 5 列 覆盖率: 不适用

• 当js引擎调用完fun函数，重新回到全局范围开始执行时

• 函数作用域竟然没了！

• scopes中只留下全局作用域对象window。



“**总结: 函数作用域**  
**其实是js引擎在调用函数时才临时创建的一个作用域对象。其中保存函数的局部变量。而函数调用完，函数作用域对象就释放了。**”



所以:

“

JS中**函数作用域对象**, 还有个别名  
——”**活动的对象(Actived Object)**”

简称, AO。

”

所以, 局部变量不可重用。



“ 其实：  
一个函数的调用过程  
就像我们在家做一顿饭一样。  
问题：做饭分那三大步？ ”





# 函数调用过程 vs 做饭三步曲

- 第一步: 厨师准备食材:



程序中的js引擎  
也是先创建函数作用域对象  
保存局部变量

```
function fun() {  
    var a = 100;  
    a++;  
    console.log(a);  
}  
fun();
```



```
▶ 变量  
▼ 作用域  
    ▼ 本地  
        ► this: Window  
        a: undefined
```



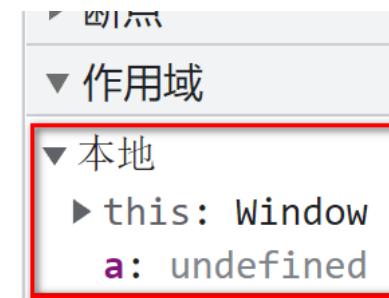
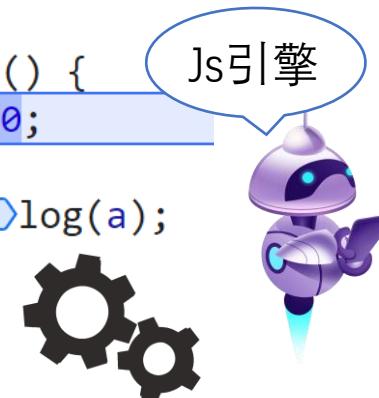
# 函数调用过程 vs 做饭三步曲

- 第二步：厨师按所学的菜谱上的步骤做菜



程序中的js引擎  
也是按函数体规定的步骤执行每一项操作

```
function fun() {  
    var a = 100;  
    a++;  
    console.log(a);  
}  
fun();
```





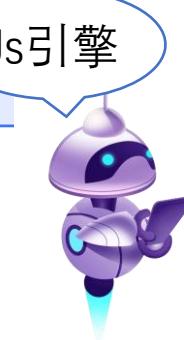
# 函数调用过程 vs 做饭三步曲

- 第三步：收拾厨房，保持干净整洁



程序中的js引擎  
也会在调用函数后自动释放临时创建的  
函数作用域对象

```
function fun() {  
    var a = 100;  
    a++;  
    console.log(a);  
}  
fun();
```





“ 所以，  
函数调用过程=做菜三部曲  
备料，做菜，收拾厨房 ”



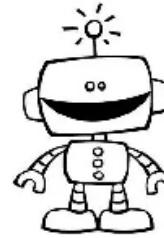


“本节最后，  
我们完整看一遍  
一个函数调用的过程... ...”



程序刚开始执行时

window

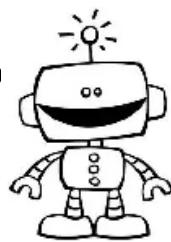


```
var a=10;  
function fun1(){  
    var a=100;  
    a++;  
    console.log(a);//101  
}  
fun1();  
console.log(a);//10
```

程序刚开始执行时，先创建全局变量

window

a : 10



```
function fun1(){  
    var a=100;  
    a++;  
    console.log(a);//101  
}  
fun1();  
console.log(a);//10
```

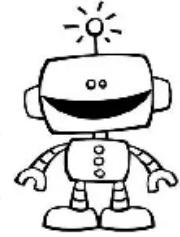
## 再创建函数对象，保存函数体

window

a : 10

底层自动翻译为: 

```
function fun1(){  
    var a=100;  
    a++;  
    console.log(a);//101  
}  
fun1();  
console.log(a);//10
```



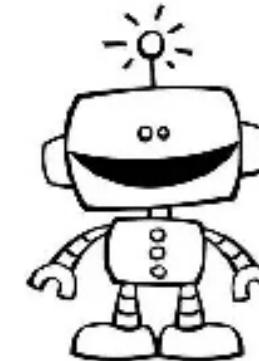
再创建函数对象，保存函数体

window

a : 10

fun1 : 0x1234

地址: 0x1234  
new Function()  
function fun1(){  
 var a=100;  
 a++;  
 console.log(a);  
}



fun1();  
console.log(a); // 10

每个函数对象身上都有一个作用域链，  
暂时只保存全局作用域对象window

window

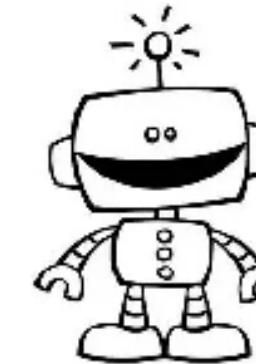
a : 10

fun1 : 0x1234

地址: 0x1234  
new Function()  
function fun1(){  
 var a=100;  
  
 a++;  
  
 console.log(a);  
}  
} 作用域链

空

window



fun1();  
console.log(a); // 10

调用函数时:

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun1(){

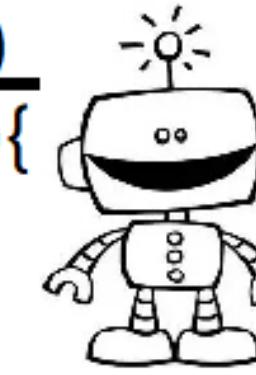
var a=100;

a++;

console.log(a);

} 作用域链

window



fun1().  
console.log(a);

做菜三部曲:

调用函数时:

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

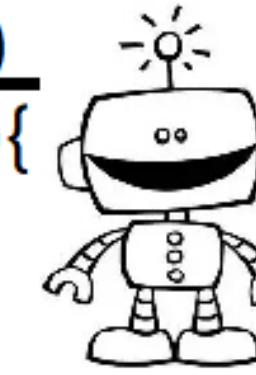
function fun1(){

var a=100;

a++;

console.log(a);

} 作用域链



fun1().  
console.log(a);

做菜三部曲:

1. 备料

window

调用函数时:

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun1(){

var a=100;

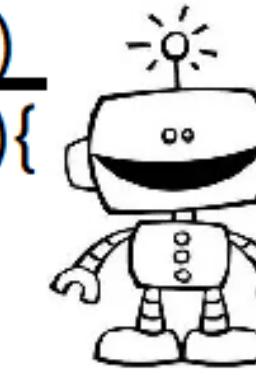
a++;

console.log(a);

作用域链

}

window



fun1().  
console.log(a);

做菜三部曲:

1. 备料  
临时创建

函数作用域对象

调用函数时:

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun1(){

var a=100;

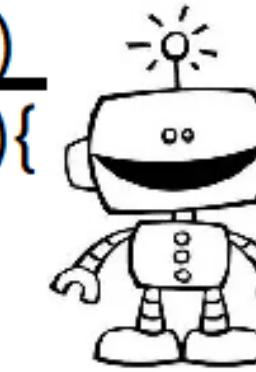
a++;

console.log(a);

作用域链

}

window



fun1().  
console.log(a);

做菜三部曲:

1. 备料  
临时创建

函数作用域对象  
地址: 0x9091

调用函数时:

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun1(){

var a=100;

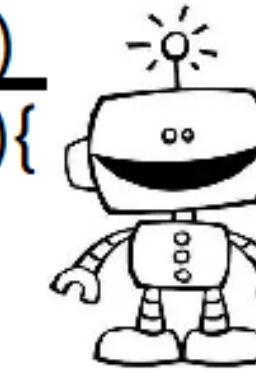
a++;

console.log(a);

作用域链

}

window



fun1().  
console.log(a);

做菜三部曲:

1. 备料  
临时创建

函数作用域对象  
地址: 0x9091

a=100;

调用函数时:

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()  
function fun1(){

var a=100;

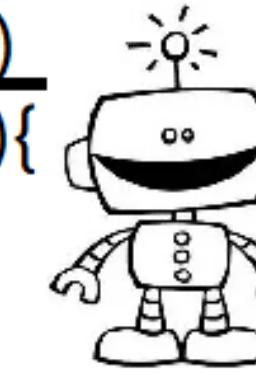
a++;

console.log(a);

作用域链

0x9091

window



fun1().  
console.log(a);

做菜三部曲:

1. 备料  
临时创建

函数作用域对象  
地址: 0x9091

a=100;

# 函数调用过程中

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun1(){

var a=100;

a++;

console.log(a);

} 作用域链



fun1();

console.log(a);

做菜三部曲:  
2. 按菜谱执行操作

0x9091

window

地址: 0x9091

a = 100



# 函数调用过程中

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun1(){

var a=100;

a++;

console.log(a);

} 作用域链



0x9091

window

fun1();

console.log(a);

做菜三部曲:

2. 按菜谱执行操作

地址: 0x9091

a = 100 a++;



# 函数调用过程中

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun1(){

var a=100;

a++;

console.log(a);

} 作用域链



0x9091

window

fun1();

console.log(a);

做菜三部曲:

2. 按菜谱执行操作

地址: 0x9091

a=101



# 函数调用过程中

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun1(){

var a=100;

a++;

console.log(a);

作用域链

0x9091

window



fun1(); 101

console.log(a);

做菜三部曲:

2. 按菜谱执行操作

地址: 0x9091

a=101



# 函数调用后

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun1(){

var a=100;

a++;

console.log(a);

作用域链

}



0x9091

window

fun1(); 101

console.log(a);

做菜三部曲:

3. 收拾厨房

地址: 0x9091

a=101



# 函数调用后

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun10{

var a=100;

a++;

console.log(a);

作用域链

空

window

console.log(a);

做菜三部曲:

3. 收拾厨房



释放函数

作用域对象

局部变量紧跟着释放了

所以，局部变量在函数调用后，就不存在了

## 函数调用后

window

a : 10

fun1 : 0x1234

地址: 0x1234 (菜谱)

new Function()

function fun10{

    var a=100;

    a++;

console.log(a);

    作用域链

}



window



console.log(a);//10

内存恢复到函数调用之前的而样子



“  
问题: 所有函数在定义时,  
作用域链scopes中  
都只有一个全局作用域对象  
window吗?  
”





“

**往后学就知道了... ...**

”





# Q4: 闭包

# 需求: 定义一个函数帮小孩儿管理压岁钱



- 1. 假设小孩儿共有1000元压岁钱，存哪儿？
- 2. 小孩儿每花一笔钱，就从总钱数里减去一笔。





```
var total=1000;
function pay(money){
    total-=money
    console.log(`花了${money}还剩${total}元`)
}
pay(100); //应该剩900
pay(100); //应该剩800
```

The screenshot shows the Chrome DevTools interface with the '控制台' (Console) tab selected. The console output displays two log messages from the executed JavaScript code:

- 花了100还剩900元
- 花了100还剩800元

The messages are timestamped as "3 closure.html:14".



“ 问题：  
假设别人的程序中也用到了total变量，  
并在你调用pay()之前修改了total变量值，  
结果会怎样？ ”





```
var total=1000;
function pay(money){
    total-=money
    console.log(`花了${money}还剩${total}元`)
}
pay(100); //应该剩900
total=0; //别人程序执行了代码
pay(100); //应该剩800
```

The screenshot shows the Chrome DevTools interface with the '控制台' (Console) tab selected. The console output displays two entries:

- "花了100还剩900元" (Logged at 3\_closure.html:14)
- "花了100还剩-100元" (The second entry is highlighted with a red box around its value, Logged at 3\_closure.html:14)



“ 全局变量特点:

优: 可重用

缺: 极易被污染

”





“

# 解决：换局部变量

”



```
// var total=1000;//易被污染
function pay(money){
    var total=1000;//改为局部变量
    total-=money
    console.log(`花了${money}还剩${total}元`)
}
pay(100); //应该剩900
total=0; //别人程序执行了代码
pay(100); //应该剩800
```



A screenshot of the Chrome DevTools interface. The top navigation bar has tabs for '元素' (Elements), '控制台' (Console), '来源' (Sources), '网络' (Network), and '性能' (Performance). The '控制台' tab is currently selected and underlined. Below the tabs is a toolbar with icons for play/pause, stop, and filters. The main area shows two log entries: '花了100还剩900元' and '花了100还剩900元'. The second entry is highlighted with a red rectangular border. On the right side of the interface, there are buttons for '默认级别' (Default Level) and '无问题' (No Issues), along with a settings gear icon.

花了100还剩900元

花了100还剩900元

3\_closure.html:15

3\_closure.html:15



“局部变量特点：

优：不会被污染

缺：用完就释放！不可重用！，”



“

**完啦！BBQ啦！**

**全局也不行！局部也不行！**

”



“解决： b-box



”



“

**错了... ...重来！**

”





“

**解决：闭包(closure)**

”



# 什么是闭包

---

- 浅显的从用法上理解
- 既重用变量又保护变量不被污染的一种编程方法。
- 今后：只要希望给一个函数，保存一个即可反复使用，又不会被外界污染的专属局部变量时，就用闭包

# 如何使用闭包：3步

- (1). 用外层函数包裹要保护的变量和使用变量的内层函数
- (2). 在外层函数内部返回内层函数对象。
- (3). 调用外层函数用变量接住返回的内层函数对象。

```
//第1步：用外层函数包裹要保护的变量和内层函数
function mother(){
  var total=1000;
  //第2步：返回内层函数对象
  function pay(money){
    total-=money
    console.log(`花了${money}还剩${total}元`)
  }
}
//第3步：调用外层函数，用变量接住内层函数对象
var pay=mother();
//pay接住的就是mother()返回出来的内层函数对象
pay(100); //应该剩900
total=0; //别人程序执行了代码
pay(100); //应该剩800
```



“

**试试：**

”





元素 控制台 来源 网络 性能 » | ⚙️ : X

▶ 🔍 top ▼ ⚡ 过滤 默认级别 ▼ 无问题 | ⚙️

花了100还剩900元	3_closure.html:17
花了100还剩800元	3_closure.html:17

能够连续减，说明total变量被反复使用了！



花了100还剩900元

花了100还剩800元

这里的total=0没有影响后边的total计算  
说明，total被保护起来，只能pay函数使用。

3\_closure.html:17

3\_closure.html:17

元素 控制台 来源 网络 性能 » | ⚙️ : X

▶ ⚡ top ▾ ⚡ 过滤 默认级别 ▾ 无问题 ⚙️



“ 所以：使用闭包三步  
可以为函数保存一个  
既可重用，又不会被污染的  
专属局部变量。 ”





“ 小问题：  
**内层函数，将来迟早要被别的变量接住**  
所以，起不起名，结果是一样的！  
那干脆我们就**不起函数名了！**  
将来谁接住内层函数，谁负责起名。  
就像周星驰进了华府... ... ”



# 如何使用闭包：3步

- (1). 用外层函数包裹  
要保护的变量和使用变量的内层函数
- (2). 在外层函数内部  
返回内层函数对象。
- (3). 调用外层函数  
用变量接住返回的内层函数对象。

```
//第1步：用外层函数包裹要保护的变量和内层函数
function mother(){
    var total=1000;
    //第2步：返回内层函数对象
    return function (money){
        total-=money
        console.log(`花了${money}还剩${total}元`)
    }
}
//第3步：调用外层函数，用变量接住内层函数对象
var pay=mother();
//pay接住的就是mother()返回出来的内层函数对象
pay(100); //应该剩900
total=0; //别人程序执行了代码
pay(100); //应该剩800
```



“但是：  
为什么三步就可以实现？  
看原理... ...”



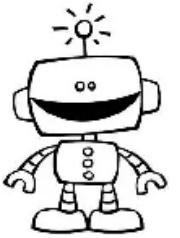
## (1). 定义外层函数时

window

var fun1=new Function(...)

底层自动翻译为:

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    }  
}  
  
var pay=mother();  
pay(100);  
total=0;  
pay(100);
```



定义函数时

window

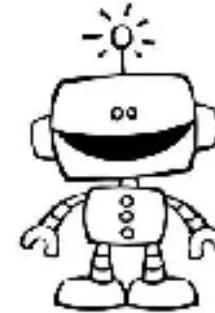
全局作用域对象

mother:0x9091

地址: 0x9091

new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    }  
}
```



```
var pay=mother();  
pay(100);  
total=0;  
pay(100);
```

定义函数时  
window

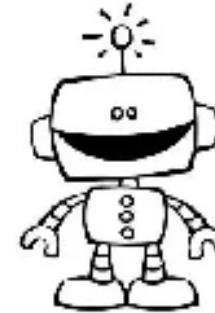
全局作用域对象  
mother:0x9091

地址: 0x9091

new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    } } 作用域链
```

} 空  
window



```
var pay=mother();  
pay(100);  
total=0;  
pay(100);
```

执行var pay=mother()

window

全局作用域对象

mother:0x9091

地址: 0x9091

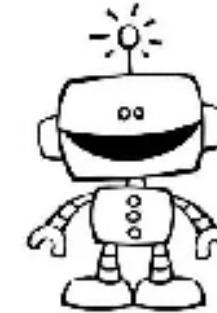
new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    } } 作用域链
```

}

空

window



```
var pay=mother();  
pay(100);  
total=0;  
pay(100);
```

执行var pay=mother()

window

全局作用域对象

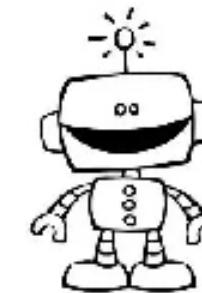
mother:0x9091

地址: 0x9091

new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    } } 作用域链
```

window



```
mother();  
var pay=pay(100);  
total=0;  
pay(100);
```

执行var pay=mother()

window

全局作用域对象

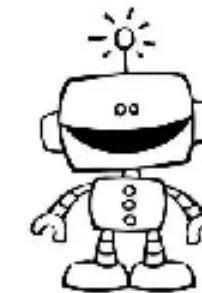
mother:0x9091

地址: 0x9091

new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    } } 作用域链
```

window



```
mother();  
var pay=pay(100);  
total=0;  
pay(100);
```

执行var pay=mother()

window

全局作用域对象

mother:0x9091

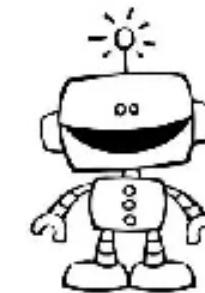
地址: 0x9091

new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    } } 作用域链
```

pay:undefined

window



```
mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);
```

然后，调用mother()

window

全局作用域对象

mother:0x9091

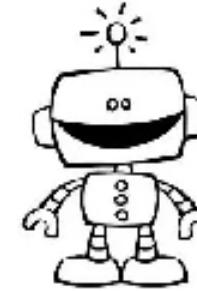
地址: 0x9091

new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    } } // 作用域链
```

pay:undefined

window



mother();

var pay=

pay(100);

total=0;

pay(100);

调用函数时

window

全局作用域对象

mother:0x9091

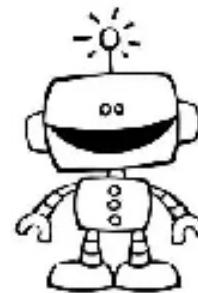
地址: 0x9091

new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    } } 作用域链
```

pay:undefined

window



mother();

var pay=

pay(100);

total=0;

pay(100);

做菜三部曲:

1. 备料

调用函数时

window

全局作用域对象

mother:0x9091

地址: 0x9091

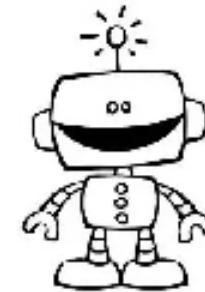
new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    } } 作用域链
```

pay:undefined

0x1234

window



mother();

var pay=

pay(100);

total=0;

pay(100);

做菜三部曲:

1. 备料

临时创建函数作用域对象

地址: 0x1234

调用函数时

window

全局作用域对象

mother:0x9091

地址: 0x9091

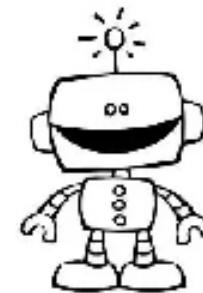
new Function() 函数对象

```
function mother(){  
    var total=1000;  
    return function(money){  
        total-=money;  
        console.log(`还剩${total}`);  
    } } 作用域链
```

pay:undefined

0x1234

window



mother();

var pay=

pay(100);

total=0;

pay(100);

做菜三部曲:

1. 备料

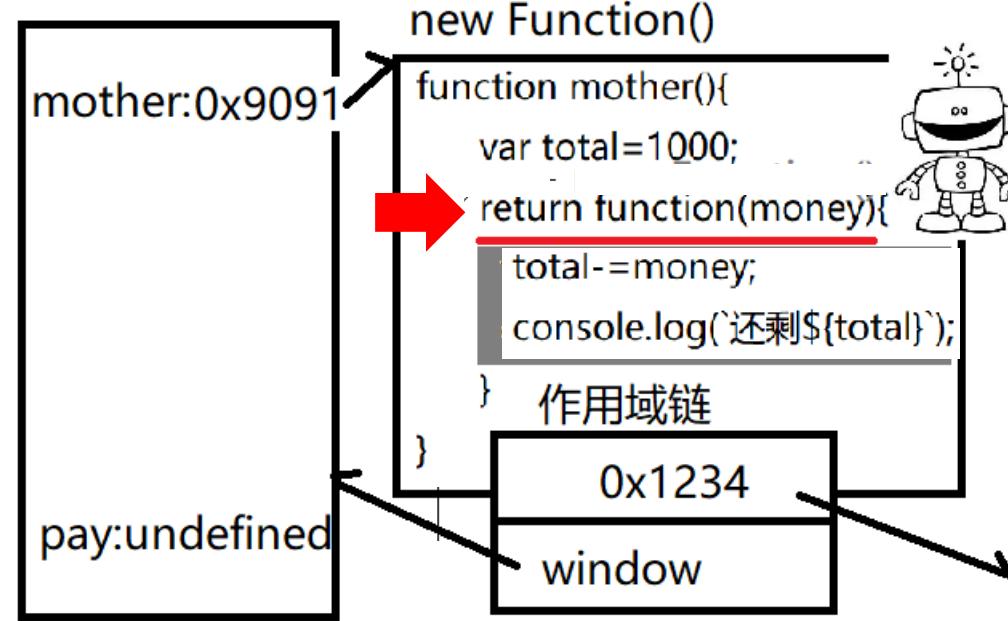
临时创建函数作用域对象

地址: 0x1234

total=1000;



调用函数时  
window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:  
2. 看菜谱做菜  
临时创建函数作用域对象  
地址: 0x1234

total=1000;



调用函数时  
window

mother:0x9091  
  
pay:undefined

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()

return function(money){

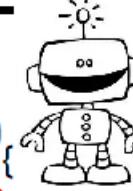
total-=money;

console.log(`还剩\${total}`);

} 作用域链

}

0x1234  
window



mother();  
var pay=  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;



调用函数时  
window

mother:0x9091  
  
pay:undefined

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

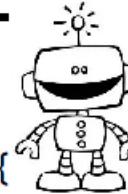
console.log(`还剩\${total}`);

}

作用域链

0x1234

window



mother();  
  
var pay=  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;

地址: 0x2345

new Function()  
function(money){

total-=money;

console.log(`还剩\${total}`);

}



调用函数时  
window

mother:0x9091  
  
pay:undefined

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

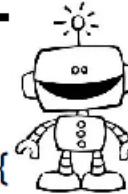
console.log('还剩\${total}');

}

作用域链

}

0x1234  
window



mother();  
var pay=  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;

地址: 0x2345

new Function()

function(money){

total-=money;

console.log('还剩\${total}');

}

但是！暂时不执行函数体的内容！  
函数！只有加()被调用时，才执行函数体！



调用函数时  
window

mother:0x9091

pay:undefined

地址: 0x9091  
new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

0x1234

window

地址: 0x2345

new Function()  
function(money){

total-=money;

console.log('还剩\${total}');

}

但是！暂时不执行函数体的内容！  
函数！只有加()被调用时，才执行函数体！

最内层函数，由内向外  
共经过几级作用域？

mother();

var pay=

pay(100);

total=0;

pay(100);

做菜三部曲：

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;



调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

}

0x1234

window

三级: 自己、妈妈、全局

mother();

var pay=

pay(100);

total=0;

pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;

地址: 0x2345

new Function()

function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

自己空

window

所以，闭包结构内层函数的作用域链，  
比一般函数作用域链，多一级作用域



“

**耳听为虚，眼见为实... ...**

”



# 闭包结构，内层函数作用域链包括几级作用域？

- 控制台执行: console.dir(pay); 看结果

```
花了100还剩900元
花了100还剩800元
: console.dir(pay)
VM462:1
▼ f pay(money) ⓘ
  arguments: null
  caller: null
  length: 1
  name: "pay"
▶ prototype: {constructor: f}
  [[FunctionLocation]]: 3_closure.html:15
▶ [[Prototype]]: f ()
▼ [[Scopes]]: Scopes[2]
  ▼ 0: Closure (mother)
    total: 800
    ▶ [[Prototype]]: Object
  ▶ 1: Global {window: Window, self: Window, document: document, name: '', ...}
```

这都说明了什么?



“

继续...

”

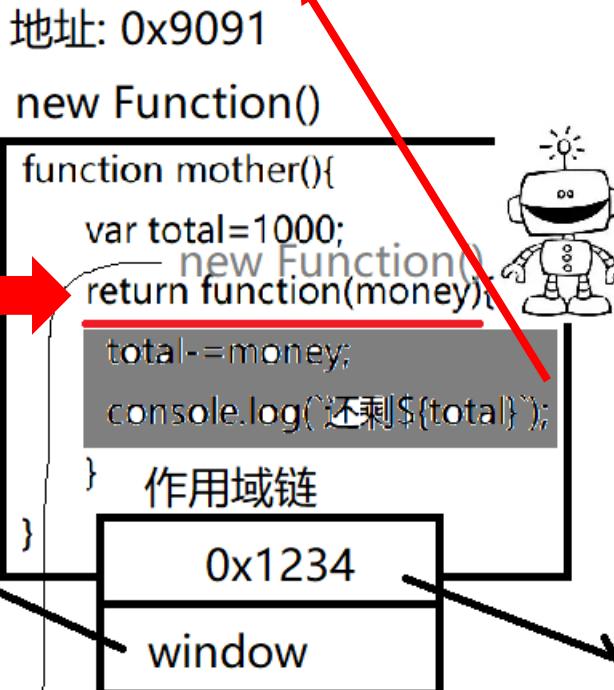




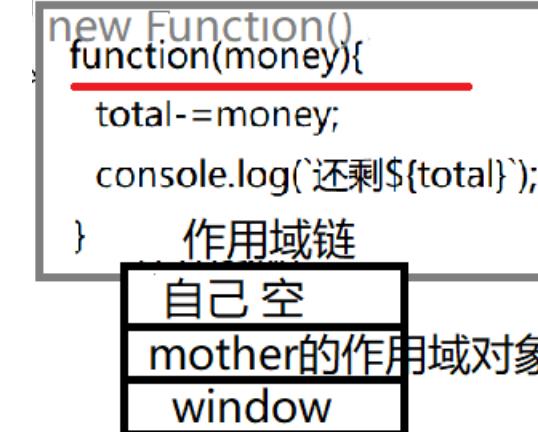
调用函数时  
window

mother:0x9091

pay:undefined



地址: 0x2345



mother();  
var pay=  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;

多出的空位指向外层函数mother的作用域对象



调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

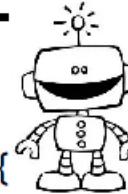
console.log('还剩\${total}');

}

作用域链

0x1234

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲: 到此, 外层函数mother调用完成

3. 收拾厨房

地址: 0x1234

total=1000;

地址: 0x2345

new Function()  
function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

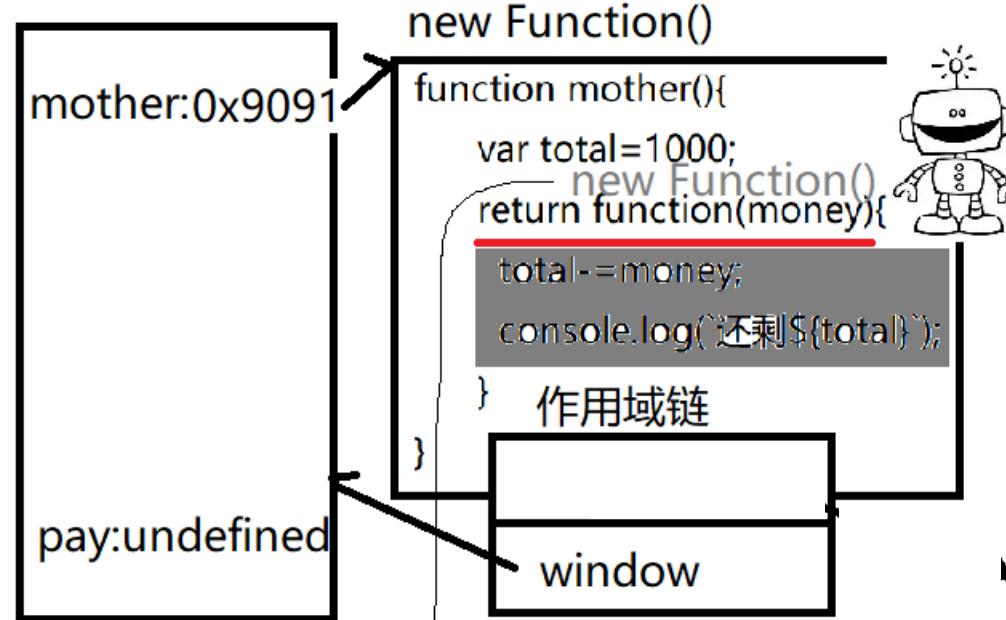
自己空

mother的作用域对象

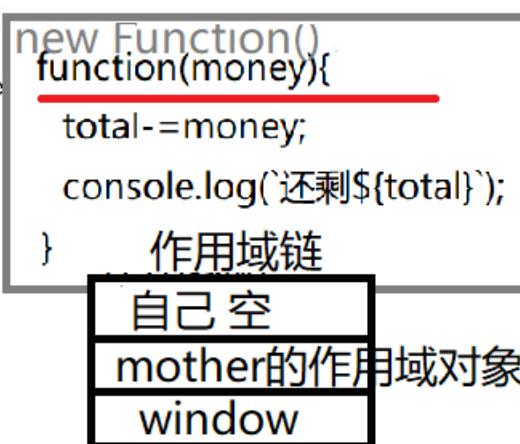
window



调用函数时  
window



地址: 0x2345



mother();  
var pay=  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

3. 收拾厨房

外层函数mother只清空作用域链中离自己近的一个作用域

地址: 0x1234

```
total=1000;
```



调用函数后

window

mother: 0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()

return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

空

window

pay: 0x2345

pay(100);

total=0;

pay(100);

地址: 0x2345

new Function()

function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

自己空

mother的作用域对象

window

但是，因为mother的作用域对象  
被内层函数的作用域链引用着，无法释放，  
就侥幸存活下来！

而且，只有内层函数pay指导total变量的存  
储位置。——专属私密

地址: 0x1234

total=1000;

mother的

作用域对象



“其实

**外层函数：怀了小宝宝的妈妈**

**内层函数：妈妈肚子里的小宝宝，”**

调用函数时

window

全局作用域对象

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

return function(money){

total-=money;

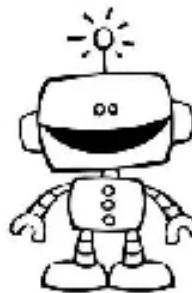
console.log(`还剩\${total}`);

}

作用域链

}

pay:undefined



mother();

var pay=

pay(100);

total=0;

pay(100);

做菜三部曲:

1. 备料



调用外层函数妈妈时  
“  
总是会自动创建外层函数妈妈的函数作用域对象  
其中，保存着**外层函数局部变量**，比如**total=1000**  
**就像妈妈给即将出生的小宝宝准备的一个红包**  
”

调用函数时

window

全局作用域对象

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

return function(money){

total-=money;

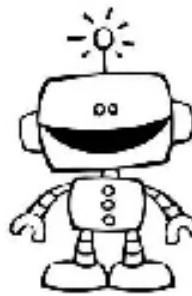
console.log(`还剩\${total}`);

}

作用域链

}

pay:undefined



mother();

var pay=

pay(100);

total=0;

pay(100);

做菜三部曲:

1. 备料

调用函数时

window

全局作用域对象

mother:0x9091

pay:undefined

地址: 0x9091

new Function()

function mother(){

var total=1000;

return function(money){

total-=money;

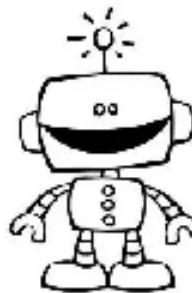
console.log(`还剩\${total}`);

}

作用域链

0x1234

window



mother();

var pay=

pay(100);

total=0;

pay(100);

做菜三部曲:

1. 备料

临时创建函数作用域对象

地址: 0x1234



调用函数时

window

全局作用域对象

mother:0x9091

pay:undefined

地址: 0x9091

new Function()

function mother(){

var total=1000;

return function(money){

total-=money;

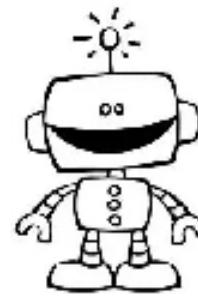
console.log(`还剩\${total}`);

}

作用域链

0x1234

window



mother();

var pay=

pay(100);

total=0;

pay(100);

做菜三部曲:

1. 备料

临时创建函数作用域对象

地址: 0x1234

total=1000;





调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

return function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

}

0x1234

window



mother();  
var pay=pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;





“  
内层函数function等于new Function()  
相当于在外层函数内部创建一个新的内层函数对象  
——小宝宝  
return  
负责将内层函数对象(小宝宝)返回到外层函数外部  
——妈妈把小宝宝生出来  
”





调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

return function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

}

0x1234

window



mother();  
var pay=pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

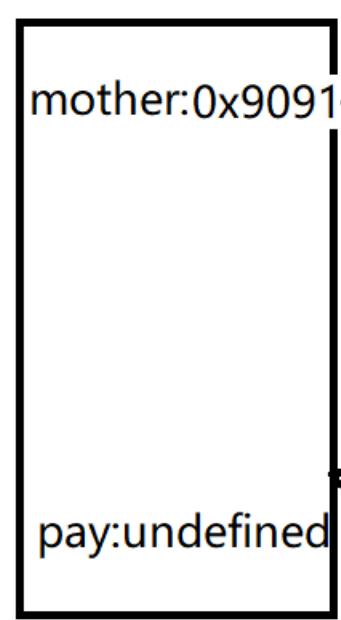
地址: 0x1234

total=1000;





调用函数时  
window



地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

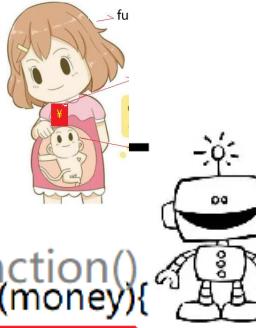
console.log(`还剩\${total}`);

} 作用域链

}

0x1234

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;





调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

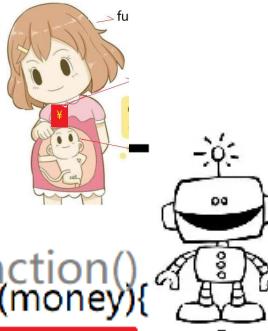
console.log(`还剩\${total}`);

}

作用域链

0x1234

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;



地址: 0x2345

new Function()

function(money){

total-=money;

console.log(`还剩\${total}`);

}





“强调：

因为内层函数只是定义，未加()调用。

所以，内层函数中的代码不执行！”



调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

console.log(`还剩\${total}`);

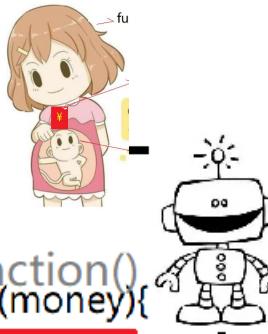
}

作用域链

}

0x1234

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;



地址: 0x2345

new Function()

function(money){

total-=money;

console.log(`还剩\${total}`);

}





调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

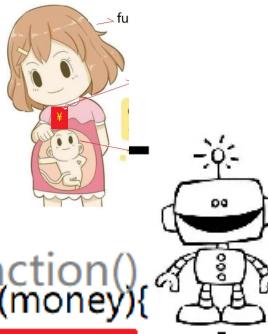
console.log('还剩\${total}');

}

作用域链

0x1234

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;



地址: 0x2345

new Function()

function(money){

total-=money;

console.log('还剩\${total}');

}





“ 因为**内层函数是在外层函数内部创建的**  
所以**内层函数的作用域链是3个格子，依次是：**

- 内层函数自己的作用域对象
- 外层函数的作用域对象
- 全局作用域对象

”



调用函数时  
window

mother:0x9091

地址: 0x9091  
new Function()

function mother(){

var total=1000;  
new Function()  
return function(money){

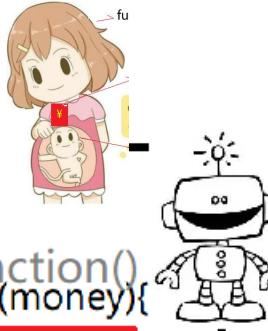
total-=money;  
console.log(`还剩\${total}`);

}

作用域链

0x1234

window



mother();  
var pay=  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象  
地址: 0x1234

total=1000;



地址: 0x2345

new Function()  
function(money){  
total-=money;  
console.log(`还剩\${total}`);  
}





调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

0x1234

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234

total=1000;



地址: 0x2345

new Function()

function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

自己空

window





调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

0x1234

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

2. 看菜谱做菜

临时创建函数作用域对象

地址: 0x1234



total=1000;

地址: 0x2345

new Function()

function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

自己空

mother的作用域对象

window





调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

0x1234

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

3. 收拾厨房

地址: 0x1234

total=1000;



地址: 0x2345

new Function()  
function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

自己空

mother的作用域对象

window





调用函数时  
window

mother:0x9091

地址: 0x9091  
new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

}

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

3. 收拾厨房

地址: 0x1234

total=1000;



地址: 0x2345

new Function()  
function(money){

total-=money;

console.log('还剩\${total}');

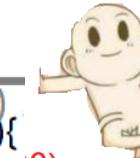
}

作用域链

自己空

mother的作用域对象

window





调用函数后

window

mother:0x9091

地址: 0x9091

new Function()

function mother(

var total=1000;

new Function()

return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

空

window

pay(100);

total=0;

pay(100);

问: 从此, 内层函数pay和外层函数mother, 还有关系吗?

地址: 0x2345

new Function()  
function(money)

total-=money;

console.log('还剩\${total}');

}

作用域链

自己空

mother的作用域对象

window

地址: 0x1234

total=1000;

mother的

作用域对象





结果：

“ 结果：内层函数宝宝  
一降生，就得到了外层函数妈妈给包的专属红包  
从此自立门户  
与外层函数妈妈再无关系！ ”



“内层函数自立门户  
后是如何执行的？”





“

**还是做菜三步曲**

”





“

**第一步——备料**

”





调用函数后

window

mother:0x9091

地址: 0x9091

new Function()

function mother(

var total=1000;

new Function()  
return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

}

空

window

pay(100);

total=0;

pay(100);

pay: 0x2345

地址: 0x2345

new Function()  
function(money)

total-=money;

console.log('还剩\${total}');

}

作用域链

自己空

mother的作用域对象

window

地址: 0x1234

total=1000;

mother的  
作用域对象



pay: 0x2345

window

地址: 0x2345

```
new Function()  
function(money){  
    total-=money; ①  
    console.log(`还剩${total}`);  
}  
作用域链
```

自己

mother的作用域对象

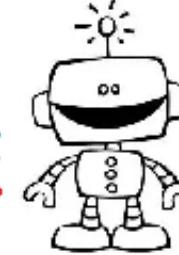
window



pay(100);

total=0;

pay(100);



地址: 0x1234

total=1000;  
mother的

作用域对象



pay: 0x2345

window

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}  
作用域链
```



自己

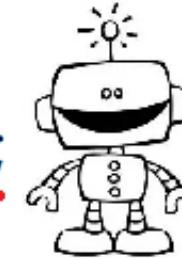
mother的作用域对象

window

pay(100);

total=0;

pay(100);



地址: 0x1234

total=1000;

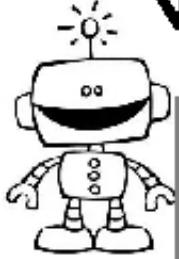
mother的

作用域对象



pay: 0x2345

window



地址: 0x2345

new Function()  
function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

自己 0x6789

mother的作用域对象

window



pay(100);

total=0;

临时创建  
pay的作用域对象

地址: 0x6789

地址: 0x1234

total=1000;

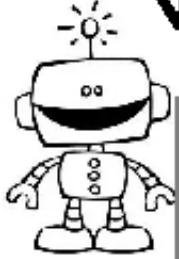
mother的

作用域对象



pay: 0x2345

window



地址: 0x2345

new Function()  
function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

自己 0x6789

mother的作用域对象

window



pay(100);

total=0;

临时创建  
pay的作用域对象

地址: 0x6789

地址: 0x1234

total=1000;

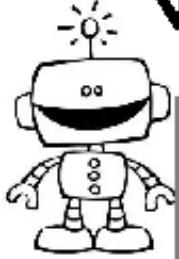
mother的

作用域对象



pay: 0x2345

window



地址: 0x2345

new Function()  
function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

自己 0x6789

mother的作用域对象

window



pay(100);

total=0;

临时创建  
pay的作用域对象

地址: 0x6789

money: 100

地址: 0x1234

total=1000;

mother的

作用域对象



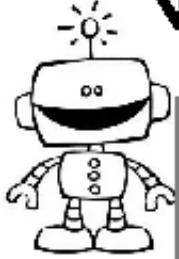


“**第二步**  
——按菜谱做菜”



pay: 0x2345

window



地址: 0x2345

new Function()  
function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

自己 0x6789

mother的作用域对象

window



临时创建

pay(100);

total=0;

pay(100);

pay的作用域对象

地址: 0x6789

money: 100

地址: 0x1234

total=1000;

mother的

作用域对象



pay: 0x2345

window

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}
```

作用域链

自己 0x6789

mother的作用域对象

window

pay(100);

total=0;

pay(100);

临时创建

pay的作用域对象

地址: 0x6789

money: 100

地址: 0x1234

total=1000;

mother的

作用域对象



pay: 0x2345

window

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}
```

作用域链

自己 0x6789

mother的作用域对象

window

pay(100);

total=0;

pay(100);

临时创建

pay的作用域对象

地址: 0x6789

money: 100

地址: 0x1234

total=1000;

mother的

作用域对象



pay: 0x2345

window

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}
```

作用域链

自己 0x6789

mother的作用域对象

window

pay(100);

total=0;

pay(100);

临时创建

pay的作用域对象

地址: 0x6789

money: 100

地址: 0x1234

total=1000;

mother的

作用域对象

pay: 0x2345

window

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}
```

作用域链

自己 0x6789

mother的作用域对象

window

pay(100);

total=0;

pay(100);

临时创建

pay的作用域对象

地址: 0x6789

money: 100

地址: 0x1234

total=1000; -=100  
mother的

作用域对象



pay: 0x2345

window

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}
```

作用域链

自己 0x6789

mother的作用域对象

window

pay(100);

total=0;

pay(100);

临时创建

pay的作用域对象

地址: 0x6789

money: 100

地址: 0x1234

total= 900

mother的

作用域对象

pay: 0x2345

window

地址: 0x2345

`new Function()`  
function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

自己 0x6789

mother的作用域对象

window

pay(100); 900

total=0;

pay(100);

临时创建

pay的作用域对象

地址: 0x6789

money: 100

地址: 0x1234

total= 900

mother的

作用域对象



“**第三步**  
——收拾厨房”



pay: 0x2345

window

地址: 0x2345

`new Function()`  
function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

自己 0x6789

mother的作用域对象

window

pay(100); 900

total=0;

pay(100);

临时创建

pay的作用域对象

地址: 0x6789

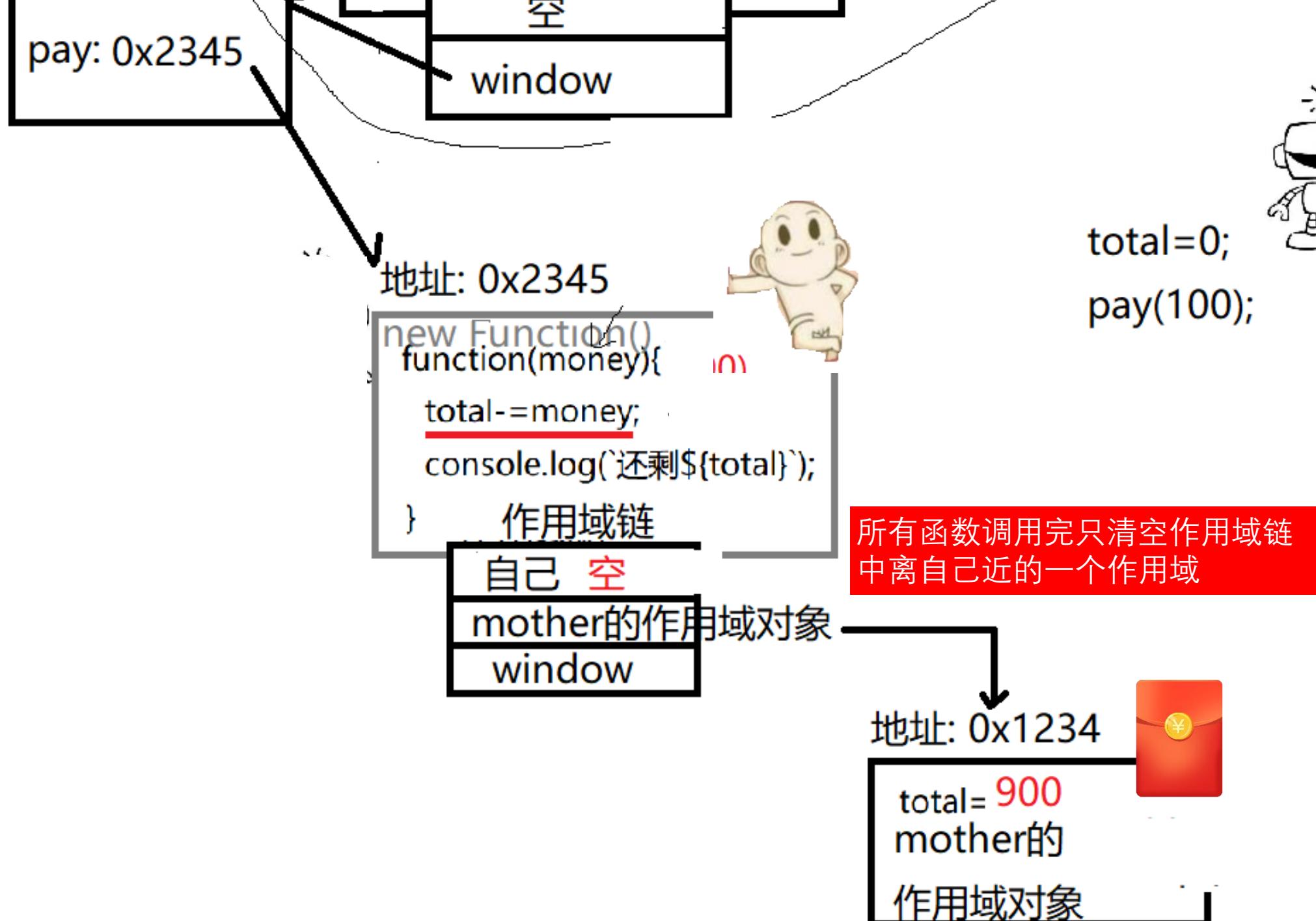
money: 100

地址: 0x1234

total= 900

mother的

作用域对象





执行total=0

window

mother:0x9091

地址: 0x9091

new Function()

```
function mother(){  
    var total=1000;  
    new Function()  
    return function(money){
```

total-=money;

console.log('还剩\${total}');

}

作用域链

空

window

pay: 0x2345

地址: 0x2345

new Function()

```
function(money){  
    total-=money; in
```

```
    console.log('还剩${total}');  
}
```

作用域链

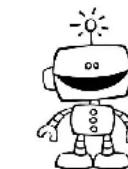
自己 空

mother的作用域对象

window

total=0;

pay(100);



地址: 0x1234

total= 900

mother的

作用域对象



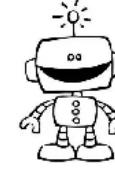


执行total=0

window

mother:0x9091

自动在  
全局创建  
total=0;



pay: 0x2345

地址: 0x9091

new Function()

```
function mother(){  
    var total=1000;  
    new Function()  
    return function(money){
```

total-=money;

console.log('还剩\${total}');

}

} 作用域链

空

window

地址: 0x2345

new Function()

```
function(money){
```

total-=money;

console.log('还剩\${total}');

}

} 作用域链

自己 空

mother的作用域对象

window

total=0;  
pay(100);

给从未声明过的变量赋值  
自动在全局创建该变量

地址: 0x1234

total= 900  
mother的  
作用域对象





“

# 再次调用内层函数

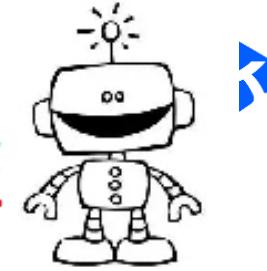
”



pay: 0x2345

window

pay(100);



地址: 0x2345

```
new Function()  
function(money){  
    total-=money; ①  
    console.log(`还剩${total}`);  
}  
② 作用域链
```

自己

mother的作用域对象

window

地址: 0x1234

total=900  
mother的

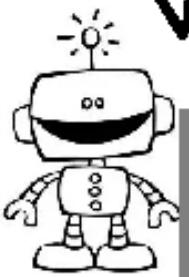
作用域对象

pay: 0x2345

window



pay(100);



地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}  
作用域链
```

自己

mother的作用域对象

window

临时创建

pay的作用域对象  
地址: 0x6789

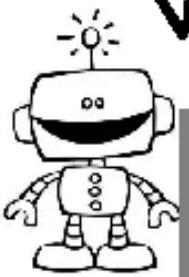
地址: 0x1234

total=900  
mother的  
作用域对象

pay: 0x2345

window

pay(100);



地址: 0x2345

~~new Function()~~  
function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

自己 0x6789

mother的作用域对象

window

临时创建

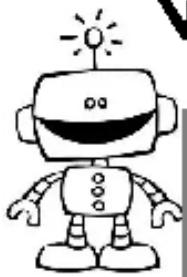
pay的作用域对象  
地址: 0x6789

地址: 0x1234

total=900  
mother的  
作用域对象

pay: 0x2345

window



地址: 0x2345

`new Function()  
function(money){`

total-=money;

`console.log(`还剩${total}`);`

} 作用域链

自己 0x6789

mother的作用域对象

window

pay(100);

临时创建

pay的作用域对象  
地址: 0x6789

money: 100

地址: 0x1234

total=900  
mother的  
作用域对象

pay: 0x2345

window

pay(100);

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}  
作用域链
```

自己 0x6789

mother的作用域对象

window

临时创建

pay的作用域对象  
地址: 0x6789

money: 100

地址: 0x1234

total=900;  
mother的  
作用域对象

pay: 0x2345

window

pay(100);

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}  
作用域链
```

自己 0x6789

mother的作用域对象

window

临时创建

pay的作用域对象  
地址: 0x6789

money: 100

地址: 0x1234

total=900;  
mother的  
作用域对象

pay: 0x2345

window

pay(100);

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}  
作用域链
```

自己 0x6789

mother的作用域对象  
window

临时创建

pay的作用域对象  
地址: 0x6789

money: 100

地址: 0x1234

total=900;  
mother的  
作用域对象

pay: 0x2345

window

pay(100);

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}  
作用域链
```

自己 0x6789

mother的作用域对象

window

临时创建

pay的作用域对象  
地址: 0x6789

money: 100

地址: 0x1234

total=900; -=100  
mother的  
作用域对象



pay: 0x2345

window

pay(100);

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}  
作用域链
```

自己 0x6789

mother的作用域对象

window

临时创建

pay的作用域对象  
地址: 0x6789

money: 100

地址: 0x1234

total= 800  
mother的  
作用域对象



pay: 0x2345

window

pay(100); 800

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}  
作用域链
```

自己 0x6789

mother的作用域对象  
window

临时创建  
pay的作用域对象  
地址: 0x6789

money: 100

地址: 0x1234

total= 800  
mother的  
作用域对象



“

**第三步**

**——收拾厨房**

”



pay: 0x2345

window

pay(100); 800

地址: 0x2345

```
new Function()  
function(money){  
    total-=money;  
    console.log(`还剩${total}`);  
}  
作用域链
```

自己 0x6789

mother的作用域对象

window

临时创建  
pay的作用域对象  
地址: 0x6789

money: 100

地址: 0x1234

total= 800  
mother的  
作用域对象

pay: 0x2345

window

地址: 0x2345

```
new Function()  
function(money)
```

total-=money;

console.log(`还剩\${total}`);

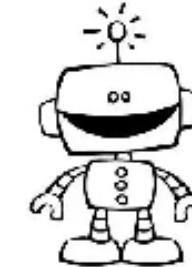
}

作用域链

自己 空

mother的作用域对象

window



地址: 0x1234

total= 800

mother的

作用域对象



“先把书读厚  
再把书读薄...”





嘿，你好  
小石头~



- 外层函数是妈妈
- 内层函数是宝宝
- 外层函数的局部变量是妈妈给宝宝包的专属红包
- 调用外层函数妈妈，生了小宝宝，小宝宝拿着属于自己的红包，自立门户！



嘿，你好  
小石头~



function mother(){

var total=1000

嘿，很高  
兴认识你

return function(money){

var pay=





嘿，你好  
小石头~



function mother(){

var total=1000

嘿，很高  
兴认识你

return function(money){

var pay=



pay(100)

total-=100



嘿，你好  
小石头~

function mother(){



var total=1000

嘿，很高  
兴认识你

return function(money){

var pay=



pay(100)

-100

total-=100

900



嘿，你好  
小石头~



function mother(){

var total=1000

嘿，很高兴认识你

return function(money){

var pay=



pay(100)

1000

total-=100

-100

pay(100)

900

total-=100



嘿，你好  
小石头~



function mother(){

var total=1000

嘿，很高  
兴认识你

return function(money){

var pay=



1000

pay(100) -100

total-=100 900

pay(100) -100

total-=100 800



“

# 到底什么是闭包？

”





# 到底什么是闭包:

- 闭包也是一个对象
- 闭包就是每次调用外层函数时，临时创建的函数作用域对象。
- 为什么外层函数作用域对象能留下来？因为被内层函数对象的作用域链引用着，无法释放。

```
▼ f pay(money) ⓘ  
  arguments: null  
  caller: null  
  length: 1  
  name: "pay"  
► prototype: {constructor: f}  
  [[FunctionLocation]]: 3\_closure.html:15  
► [[Prototype]]: f ()  
▼ [[Scopes]]: Scopes[2]  
  ▼ 0: Closure (mother) ← [Red box]  
    total: 800  
  ► [[Prototype]]: Object  
► 1: Global {window: Window, self: Window, document: document, name: '', ...}
```

这都说明了什么？



## 一句话概括：闭包如何形成

“  
外层函数调用后，  
外层函数的作用域对象，  
被返回的内层函数的作用域链引用着，  
无法释放，  
就形成了闭包对象”





调用函数时  
window

mother:0x9091

地址: 0x9091  
new Function()

function mother(){

var total=1000;  
new Function()  
return function(money){

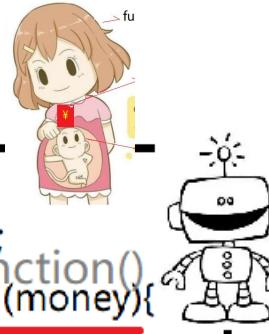
total-=money;  
console.log('还剩\${total}');

}

作用域链

0x1234

window



mother();  
var pay=  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

3. 收拾厨房

地址: 0x1234

total=1000;



地址: 0x2345

new Function()  
function(money){

total-=money;  
console.log('还剩\${total}');

}

作用域链

自己空

mother的作用域对象

window





调用函数时  
window

mother:0x9091

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()  
return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

}

window



mother();  
var pay=←  
pay(100);  
total=0;  
pay(100);

做菜三部曲:

3. 收拾厨房

地址: 0x1234

total=1000;



地址: 0x2345

new Function()

function(money){

total-=money;

console.log('还剩\${total}');

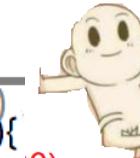
}

作用域链

自己空

mother的作用域对象

window





调用函数后

window

mother: 0x9091

地址: 0x9091

new Function()

function mother(

var total=1000;

new Function()

return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

}

空

window



pay(100);

total=0;

pay(100);

做菜三部曲:

3. 收拾厨房

pay: 0x2345

地址: 0x2345

new Function()  
function(money)

total-=money;

console.log('还剩\${total}');

}

作用域链

自己空

mother的作用域对象

window



地址: 0x1234

total=1000;

mother的  
作用域对象





调用函数后

window

mother:0x9091

地址: 0x9091

new Function()

function mother(

var total=1000;

new Function()

return function(money){

total-=money;

console.log('还剩\${total}');

}

作用域链

}

空  
window



pay(100);

total=0;

pay(100);

做菜三部曲:

3. 收拾厨房

pay: 0x2345

地址: 0x2345

new Function()  
function(money)

total-=money;

console.log('还剩\${total}');

}

作用域链

自己空

mother的作用域对象

window



闭包对象

地址: 0x1234

total=1000;

mother的

作用域对象





“ 闭包缺点：  
由于闭包藏得很深  
几乎找不到  
所以，极容易造成**内存泄漏**！ ”





“解决：  
及时释放不用的闭包  
”





“如何释放不用的闭包对象？

将保存内层函数对象的变量赋值为null

导致函数名变量与内层函数对象分开

”

pay: 0x2345

window

地址: 0x2345

```
new Function()  
function(money)
```

total-=money;

console.log(`还剩\${total}`);

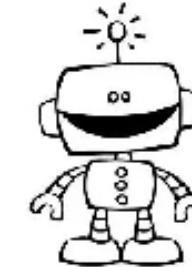
}

作用域链

自己 空

mother的作用域对象

window



地址: 0x1234

total= 800

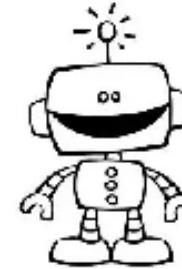
mother的

作用域对象

pay: null

window

pay=null



地址: 0x2345

```
new Function()  
function(money)  
  total-=money;  
  console.log(`还剩${total}`);  
}
```

作用域链

自己 空

mother的作用域对象

window

- 内层函数对象没人要了，会被js引擎自动释放

地址: 0x1234

total= 800

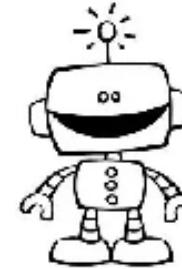
mother的

作用域对象

pay: null

window

pay=null



- 内层函数引用的闭包对象，也没人要了
- 闭包对象就被释放了

地址: 0x1234

total= 800

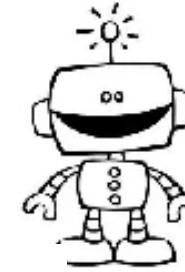
mother的

作用域对象

pay: null

window

pay=null





“最后，内存恢复干  
净的初始状态”





pay调用后  
window

mother:0x9091

total=0;

pay: null

pay=null

地址: 0x9091

new Function()

function mother(){

var total=1000;

new Function()

return function(money){

total-=money;

console.log(`还剩\${total}`);

}

作用域链

空

window



# Q5：闭包答题技巧

## 两大类题：

1. 看程序，判断输出结果
2. 根据需求，使用闭包实现功能（课后作业）



“ 找3种东西：  
1. 外层函数——妈妈  
2. 外层函数的局部变量——红包  
3. 内层函数(多个)——孩子(们) ”



## 外层函数返回内层函数的方法：

“

3种：

1. return

2. 强行赋值为全局变量

”

3. 将函数包裹在对象或数组中返回。



## 二种情况：

- (1). 妈妈一次生多个孩子：多个孩子共用同一个红包！
- (2). 妈妈反复生多次孩子：多次出生的孩子，拥有各自独立的红包，互不影响。



```
function fun(){//妈妈
    var i=999;//红包
    //给从未声明过的变量赋值，自动在全局创建该变量
    //剖腹产
    nAdd=function(){i++};//孩子1
    //妈妈顺产生了一个孩子2
    return function(){//孩子2
        console.log(i)
    }
}
var getN=fun();
//妈妈包了一个红包，生了2个孩子
//结果：两个孩子共用同一个红包！
getN();//红包=999
nAdd();// (红包+=1) 红包=1000
getN();//红包=999
```



```
function mother(){//妈妈
    var i=0;//红包
    return function(){//孩子
        i++;
        console.log(i);
    }
}
var get1=mother();//妈妈生老大，给老大包一个红包
get1();//老大的红包是1
var get2=mother();//妈妈又生了老二，又给老二包一个新的红包
get2();//老二的红包是1
get1();//老大的红包+1，变成2
get2();//老二的红包+1，变成2
```



```
function fun(){//妈妈
    arr=[];//不算红包，因为内层函数中没有用到!
    //但是arr是给未声明的变量强行赋值
    //自动在全局创建该变量
    for(var i=0/*红包*/;i<3;i++){//循环三次
        //强行给外部全局变量中添加新函数—破腹产
        //      new Function()
        arr[i]=function()//反复创建了3个孩子
            console.log(i);//因为只是创建函数，所以这里暂时不执行。
    }
}
//for结束后，i=3
}
fun();
//妈妈一次刨妇产生了3个孩子，包了一个共用的红包，红包里是3块钱
//三个兄弟，输出的结果是一样的，都是3
arr[0]();//3
arr[1]();//3
arr[2]();//3
```



Q6: 作业  
课后发闭包相关笔试题和讲解视频



我们的口号是：  
早学早面试卷别人  
晚学晚面试被人卷

Balkeba  
开课吧