04 闭包面试真题集中解析

更新时间：2020-03-03 12:20:46

> 不想当将军的士兵，不是好士兵。——拿破仑

经过了两个小节围绕知识点和原理的“厮杀”，相信大家都对闭包及作用域知识有了超出绝大部分竞争对手的理解。

在刚刚过去的讲解中，大家见到的许多示例、问题其实本身已经由面试题演化而来。即便如此，涉及闭包的面试题仍然存在不少值得提点的“变体”。本节我们就围绕闭包来“刷题”，针对面试过程中闭包这块主要的几条面试思路来逐个拆解。

“循环体与闭包”系列

把闭包和循环体结合起来考察，是闭包最为经典的一种命题方式。通过简单的一道题，面试官可以问出他想听的许多东西。

我们来看一个大家可能都已经非常非常熟悉的题目：

```
for (var i = 0; i < 5; i++) {
    setTimeout(function() {
        console.log(i);
    }, 1000);
}

console.log(i);
```

问：上述代码输出结果是什么？（大家先自己脑内跑一下代码，记住自己的答案）

如果你是刚入门的新手，你可能会给出这样的答案：

```
0 1 2 3 4 5
```

给出这样答案的同学，让我来猜猜你在想什么：for 循环逐个输出了 0-4 的 i 值， setTimeout 这个我好像在哪见过，但是印象不深了 / 在这儿好像也不重要 / 我没看清楚，干脆忽视它好了。

于是，你给出了上述答案～

不过如果你基础不错，对 setTimeout 函数的用法特性还有印象，那么不难给出这样的 “进化版” 答案：

```
5 0 1 2 3 4
```

这部分同学是这样想的：for 循环逐个输出了 0-4 的 i 值，但 setTimeout 把输出推迟了，所以最后一行先执行了。这最后一行输出的就是 i 的终态（5），然后 1000ms 后，0-4 才会被逐个输出。

但如果你既对 setTimeout 函数有了解，又有一定的面试经验（或者说充分吸收理解了我们前几节对闭包和作用域的讲解），相信你可以给出一个正确答案：

```
5 5 5 5 5 5
```

为什么是这个答案？

最后一行的 console，大家都能看出来是 5；最后一行最先输出，大家也能理解（setTimeout 内的函数会被推迟执行）；关键是 for 循环里发生了什么，我们得来捋捋：

for 循环里的 setTimeout 执行了 5 次，每次都会将这个函数的执行推迟 1000ms ：

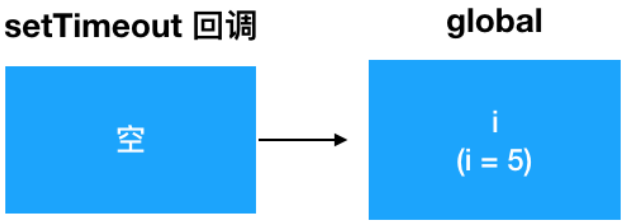
```
function() {
  console.log(i);
}
```

大家看这个函数，它自身的作用域里，是不是压根儿没有 i 这个变量？

那么结合我们前面学过的作用域知识，要想输出 i，咱们是不是得去上层作用域找？

大家想想，这个函数第一次被执行，也是 1000ms 以后的事情了，此时它试图向外、向上层作用域（这里就是全局作用域）去找一个叫 i 的变量。此时 for 循环早已执行完毕， i 也进入了尘埃落定的最终状态 ——5。所以 1000ms 后，当这个函数第一次真正被执行的时候，引用到的 i 值已经是 5 了。函数执行时整体的作用域状态示意如下：

对应的作用域链关系如下：



接下来持续四次，每隔 1000ms，都会有一个一模一样的 setTimeout 回调被执行，它试图输出的也都是这同一个全局的 i，所以说每一次输出的都是 5。

例题延伸：改造它的三种思路

循环了五次，每次却都输出一个值，这显然是 bug 级别的输出效果。如果我们希望让 i 从 0 到 4 依次被输出，那么如何改造上面的写法，才可以达到我们的目的呢？

第一种思路：

我们可以把 setTimeout 函数的第三个参数利用起来。别忘了，setTimeout 从第三个入参位置开始往后，是可以传入无数个参数的。这些参数会作为回调函数的附加参数存在。

在这里我们可以把每一轮循环里 i 的值，存进 setTimout 的第三个参数里：

```
for (var i = 0; i < 5; i++) {
    setTimeout(function(j) {
        console.log(j);
    }, 1000, i);
}
```

第二种思路：

在 setTimeout 外面再套一层函数，利用这个外部函数的入参来缓存每一个循环中的 i 值：

```
var output = function (i) {
    setTimeout(function() {
        console.log(i);
    }, 1000);
};

for (var i = 0; i < 5; i++) {
    // 这里的 i 被赋值给了 output 作用域内的变量 i
    output(i);
}
```

第三种思路：

和第二种思路比较相似，同样是在 setTimeout 外面再套一层函数，只不过这个函数是一个**立即执行函数**。利用立即执行函数的入参来缓存每一个循环中的 i 值：

```
for (var i = 0; i < 5; i++) {
    // 这里的 i 被赋值给了立即执行函数作用域内的变量 j
    (function(j) {
        setTimeout(function() {
            console.log(j);
        }, 1000);
    })(i);
}
```

知一解百 —— 说是真题，全是“变体”：

上面这道题，还请各位引起重视，这是一道母题中的母题。如果能够真正理解这道题的命题用意，你会发现只要闭包和循环体凑一块儿了，全是这么个考察套路。下面，我们就用两道题来考考大家：

（下面两道题目的输出结果是多少？ 为什么？ ）

```
function test (){
    var num = []
    var i

    for (i = 0; i < 10; i++) {
        num[i] = function () {
            console.log(i)
        }
    }

    return num[9]
}

test()()
```

```
var test = (function() {
    var num = 0
    return () => {
        return num++
    }
}())

for (var i = 0; i < 10; i++) {
    test()
}

console.log(test())
```

“复杂作用域”系列

看代码说结果、并追问候选人得出结论的思路，这是一种效率最高的闭包考察方式。这种命题方式主要有两个方向，一个方向是我们刚刚讲完的循环 & 闭包问题，另一个方向就是“复杂作用域”。

复杂作用域考题虽然高频，但是并不复杂。它的套路非常简单粗暴 —— 就是在一道题里面，尽可能地想办法给你折腾出一堆作用域（有时还会杂糅一些较为零碎的 JS 语法知识点），目的就是把你整懵。

怎样才能不被整懵？这里刚好给大家引出一个做题技巧：

回忆下我们之前学习闭包的过程，是不是基本每段代码，我都会给大家用 “方块套方块” 这种形式图示作用域？

画图 —— 这是避免被整懵的一个重要手段。

做这类题的时候，最好是找面试官要张纸要根笔（要不到靠脑补也行，问题不大）。从被执行的那个函数看起，一层一层由内而外地把作用域全局图给画出来，然后**看图说话！**

很多同学光听我这么说没啥感觉，我建议你试试，哪怕就试一次，**你就会发现问题的复杂度其实可以变得很低很低**。只要你的知识点本身没有硬伤（有硬伤就拐回去多复习几遍前两节的内容），画图 + 按图定位变量之后，把这类题做错对你来说会变得非常难。

我们下面就通过一道真题来实践一下我们这个做题方法：

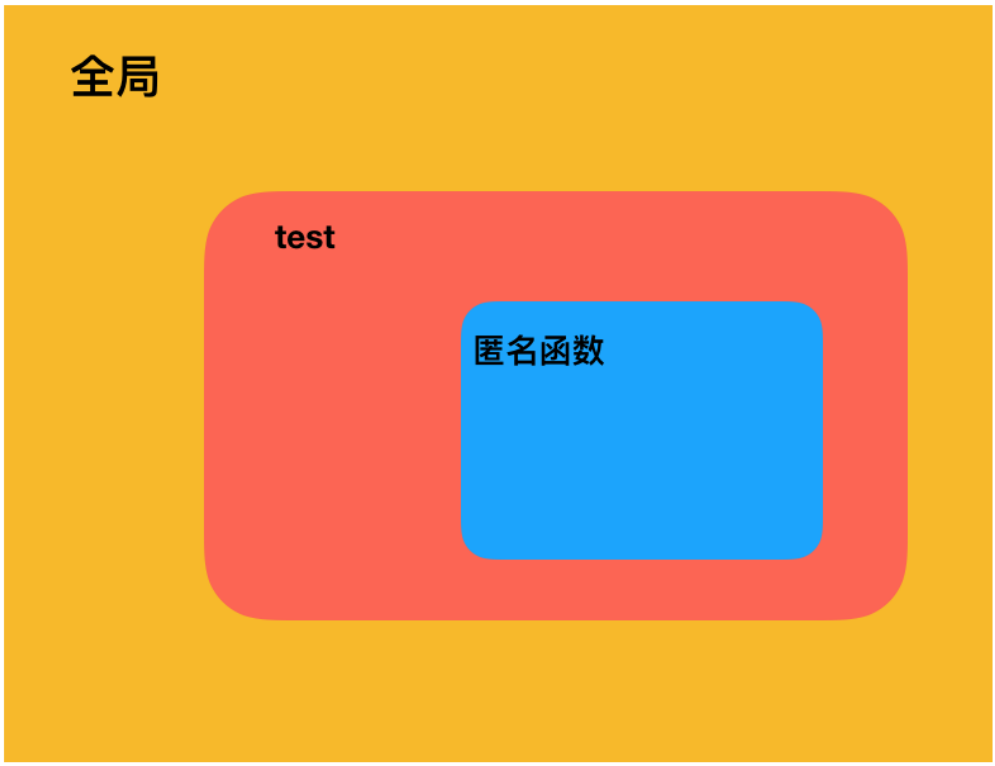
真题（例题）：

step1: 读题

```
var a = 1;
function test(){
    a = 2;
    return function(){
        console.log(a);
    }
    var a = 3;
}
test()();
```

step2：画图（到了做题环节，大家该动手和我一起画啦～）

- 分层：我们从内向外画。最内层，是一个匿名函数。匿名函数的外层，是 test 函数的作用域，再外层，就是全局作用域了，我们先把这个层级关系画出来：



- 找变量：这是这题的难点！全局变量里的 a =1 毫无疑问，最内层匿名函数里的 a 当前作用域不存在也毫无疑问；关键就是 test 函数里这个 a ，是啥情况？

要分析出 test 函数里 a 的值，大家至少有两个知识点不能虚：

1. **变量提升**：test 函数中，a = 2 在先， var a = 3 在后。但是！别忘了，函数作用域内部也是存在变量提升的，这个 var 会被提到函数的顶部，所以 test 函数体其实等价于下面这种情况：

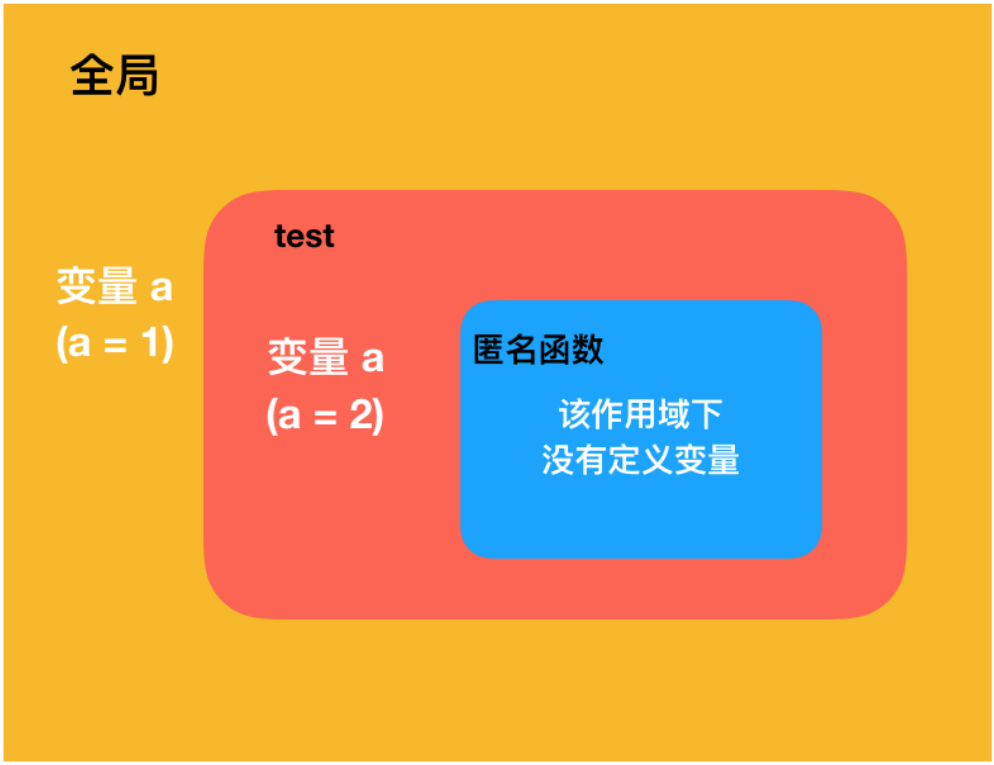
```
function test(){
  // 声明 var 被提前
  var a = 2;
  return function(){
    console.log(a);
  }
  a = 3;
}
```

2. 作用域规则：结合我们 step1 里划分的层级和上一步分析出来的 test 函数中的变量情况，不难看出匿名函数实际拿到的 a 就是 test 作用域里的 a。那么这个 a 到底是 2 还是 3？这里需要大家牢记我们前面讲解中说过的一句话：

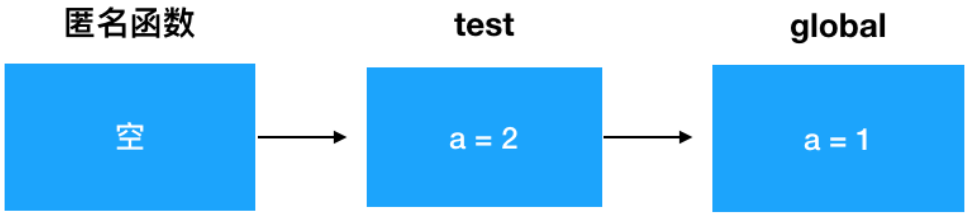
> 我们作用域的划分，是在**书写的过程中，根据你把它写在哪个位置来决定的**。像这样划分出来的作用域，遵循的就是词法作用域模型。

敲黑板！大家一定要对“书写”的时机敏感！这里我们匿名函数被定义的时候 a = 3 的赋值动作还没有发生（只有声明会被提前！），因此它拿到的 a 就是 2!

我们把我们分析出来的变量一个一个填进它们对应的地盘里：



由此可以很清晰地捋出作用域链如下：



大功告成！ 这题的答案 就是 2 ~~

真题（练习题）：

下面段代码，会输出什么？为什么？

```
function foo(a,b){
  console.log(b);
  return {
    foo:function(c){
      return foo(c,a);
    }
  }
}

var func1=foo(0);
func1.foo(1);
func1.foo(2);
func1.foo(3);
var func2=foo(0).foo(1).foo(2).foo(3);
var func3=foo(0).foo(1);
func3.foo(2);
func3.foo(3);
```

}

03 闭包—探索词法作用域模型

05 闭包的应用