第二次实验第四题

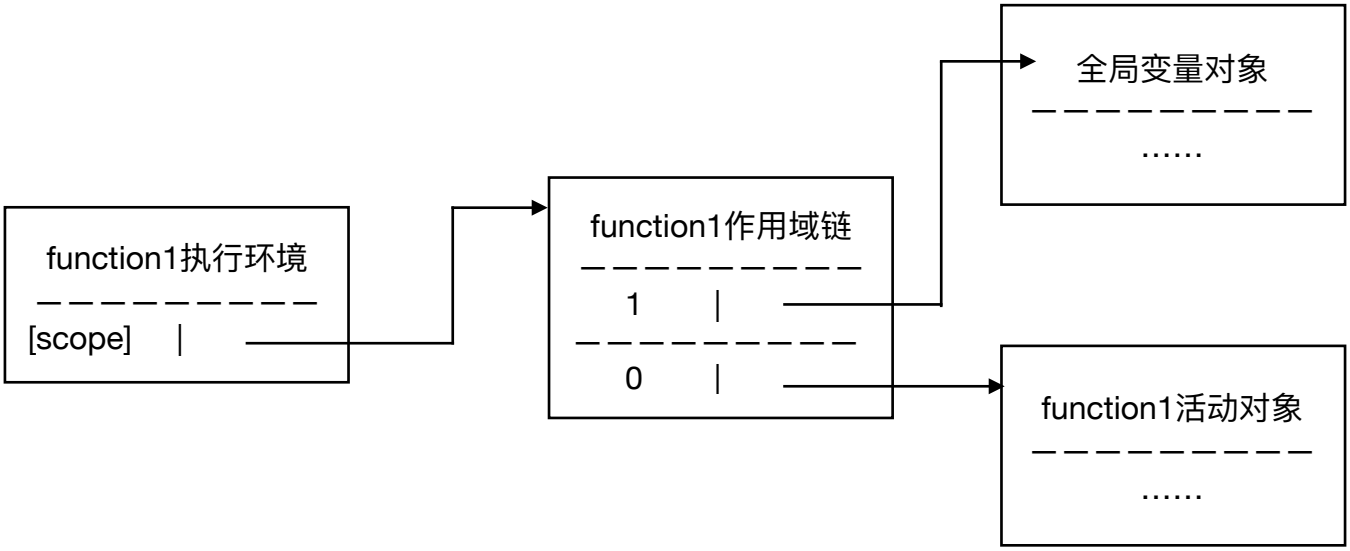
雷怡然　　软件62　　2016013274

一、作用域链

① 当某一个函数在第一次被调用时会产生一个“执行环境”。“执行环境”是一个对象，其中有一个[scope]属性，存储着一个指向“作用域链”的指针。

② 每一个函数在具体执行时，都会产生一个“活动对象”，该对象包含函数实参、函数内变量等各种信息。

③ 每一个函数的“执行环境”中的[scope]属性都指向一个“作用域链”。“作用域链”是一个指向“活动对象”的指针列表，该列表的第0个表项是本函数的“活动对象”，最后一个是“全局变量对象”。当访问某个函数中的某一变量时，从该函数的作用域链的第0项指向的活动对象开始（即本函数自身的活动对象），查找所要找的变量，如果无法找到，则从作用域链的下一项所指的活动对象开始查找，找到即返回，否则按链继续查找，直到找完“全局变量对象”。



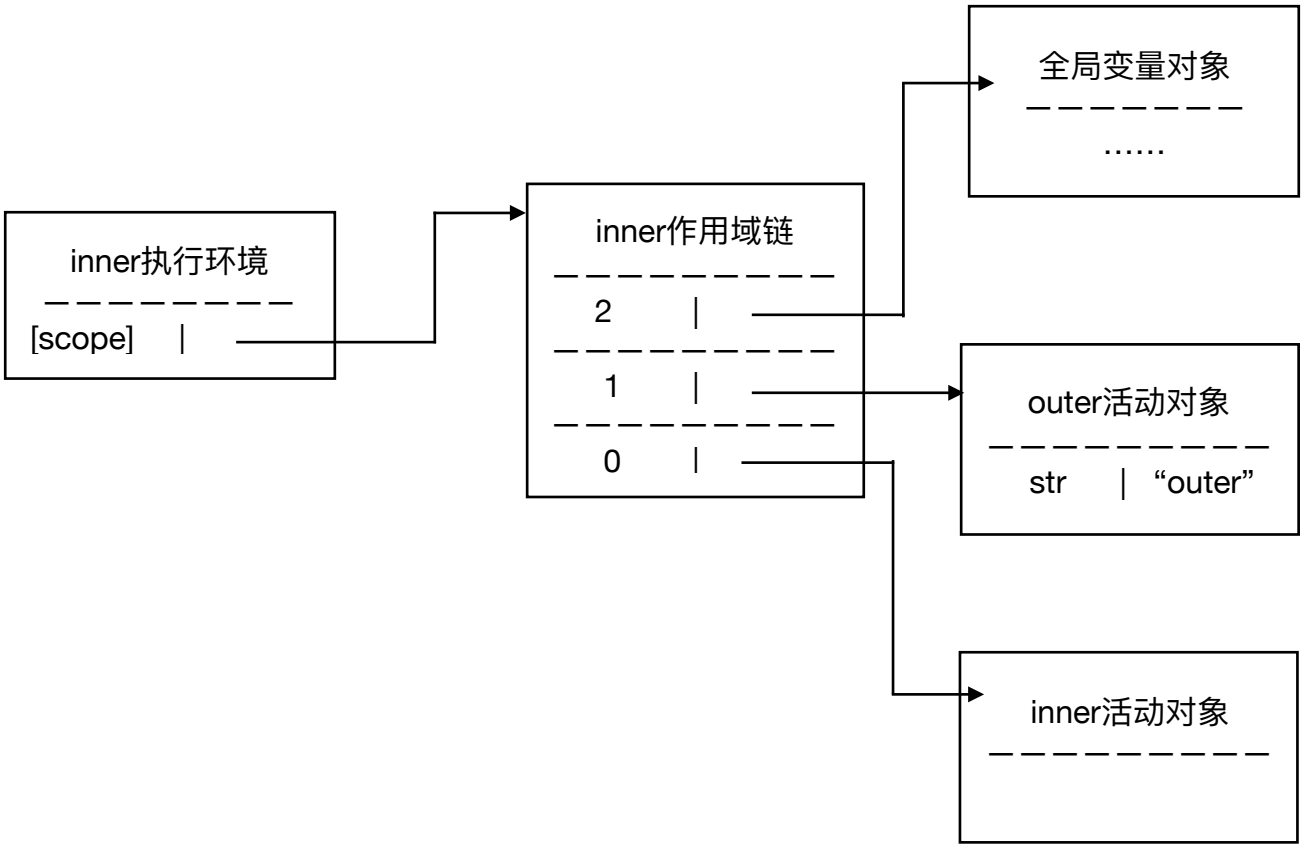
二、闭包

理论上来说，当函数执行完以后，其对应的“执行环境”、“作用域链”和“活动对象”应该被销毁，但是存在一种特殊情况。当一个函数包含“内部函数”（函数内定义的函数）且当父函数调用结束后，该“内部函数”可能由于某种操作仍然可以被调用，那么此时本应该销毁的父函数的“活动对象”由于内部函数的作用域链所指而保留了下来（执行环境和父函数的作用域链被正常销毁），这种现象就叫做闭包，换言之，内部函数的作用域链仍然保持着对父函数活动对象的引用，叫做闭包。“内部函数”的作用域链的建立方法是：复制父函数的作用域链，然后在作用域链的最开头（即第0项）插入自己的活动对象。此时，作用域链的第一项会指向父函数的活动对象，

例如：

```
function outer(){
    var str = "outer";
    function inner(){
        return str;
    }
    return inner;
}
var fn = outer();
console.log(fn());
```

当outer()执行完以后，其内部函数inner作为outer返回值仍然可以被fn调用，outer的活动对象并没有被销毁。在执行inner函数时，inner()函数中并没有变量str，只能通过作用域链向后查找，在其父函数的活动对象中找到变量str,所以inner返回的str = “outer”。



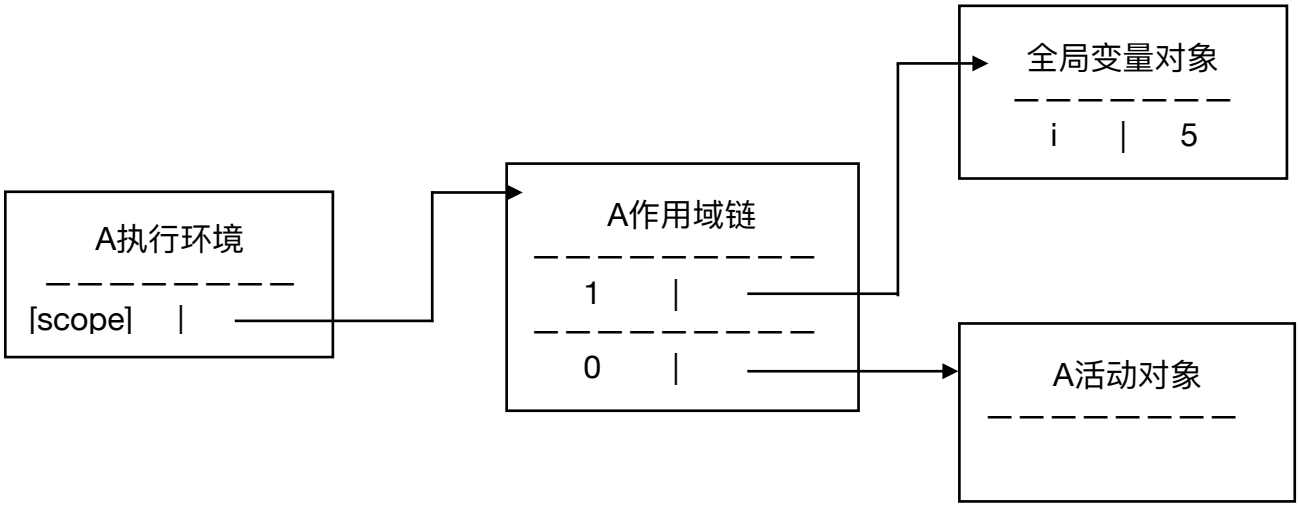
上面这个例子反映了闭包的两个作用：
1.可以读取自身函数外部的变量。
2.让这些外部变量始终保存在内存中。

三、代码的结果

1.

```
5
5
5
5
5
```

setTimeout设置在100毫秒后执行函数，为了方便讨论将待执行的函数叫做A，100毫秒后，循环已经结束,此时i=5，函数A执行时的作用域链如下：

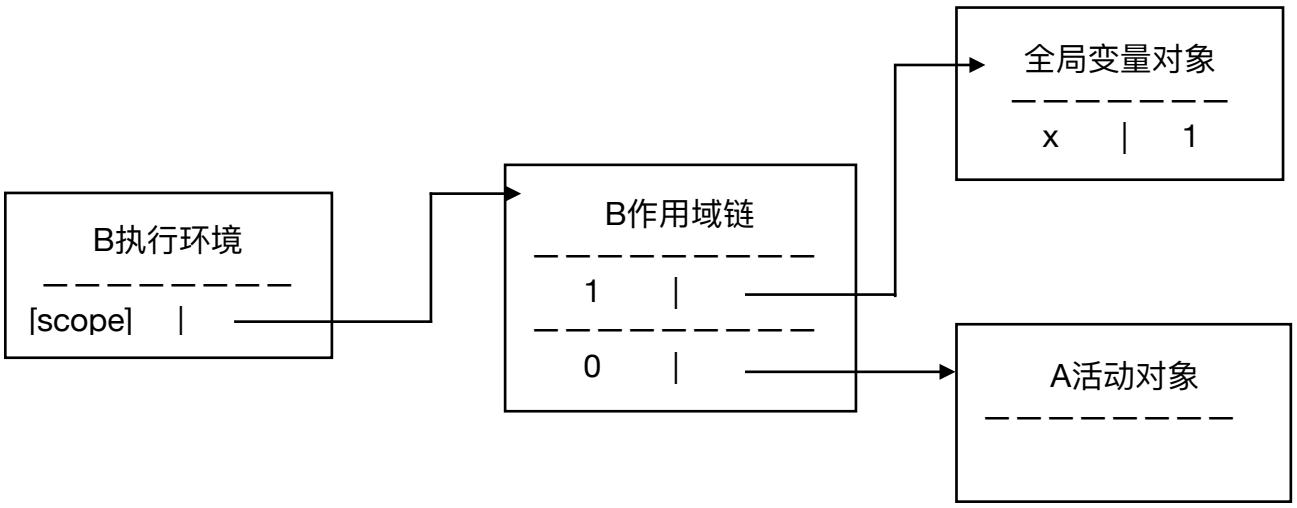


因为i定义是var i,因此i在全局变量区，100毫秒后调用函数A时，函数A中并没有i的定义，只能沿着作用域链向后寻找，在全局对象中找到i，i=5，因此输出5个5。

2.

```
2
3
4
```

为了方便讨论，将add()函数中的内部函数记做函数B，由于add函数的返回值是函数B，const num = add(); 可以看成是函数表达式，使得num就是内部函数B，num()就是调用内部函数B()，在执行num()之前，B的作用域链如下：



现在开始执行num(),由于函数B中并没有变量x,因此沿着作用域链向后查找变量x，可以在全局变量对象中找到x，执行语句console.log(++x),先将x+1，然后打印到控制台，因此x=2,打印出2；继续执行num(),按照上过程依次类推，所以最后输出结果是2 3 4。

四、有人认为作用域链和闭包有关概念的重要性下降。

原因：可能和ES6的“块级作用域”和新的“let”变量定义方式有关。

有了块级作用域后，我觉得可以把作用域链的每一项指针理解为指向一个“块级作用域的活动对象”（原来是“函数作用域的活动对象”），let定义的变量只能在某一个块级作用域的活动对象中。

考虑下面代码：

```
let arr=[];
for(let i=0;i<10;i++){
    arr[i]=function(){
        return i;
    }
}
console.log(arr[3]());//3
```

每个arr[i]()中的i都是块级作用域的i(即那一次循环中的i)，因此返回值都是i。如果把代码段一的var改为let，也会得到类似的结果。

```
for (let i = 0; i < 5; ++i) {
    setTimeout(function() {
        console.log(i + ' ');
    }, 100);
}
```

→

```
0
1
2
3
4
```

这样的结果是很符合“直觉”的，因此有些人可能觉得作用域链和闭包等概念的重要性下降了。

但是我并不同意这个观点，我觉得想要理解上述两段代码的运行结果，为什么被调用的函数内部没有对应变量却不会报错等问题，就必须理解作用域链和闭包的概念。其次，正如上面所提到的，闭包能够在内存中保留内部函数的某些外部变量，如果我们设计系统需要实现这个功能，闭包和作用域链的概念必不可少。所以我拒绝这个观点。