[**js 函数闭包内部返回函数体调用方法难点解答**](http://www.cnblogs.com/MrZouJian/p/6068400.html)

今天在网上，看到一篇关于js函数难点的文章，[js函数的一些难点](https://segmentfault.com/a/1190000005687207)。在那上面提了一下，关于js函数返回另一个函数的问题，并附上了一道面试题：



var add = function(x){

var sum = 1;

var tmp = function(x){

sum = sum + x;

return tmp;

}

tmp.toString = function(){

return sum;

}

return tmp;

} // alert(add(1)(2)(3)) --> 6



接下来，就来详细的解读返回另一个函数的问题。

之所以写这篇文章是因为，在那里面有一点让我感到奇怪，那就是最后的调用方式

add(1)(2)(3)

由于在java中，我没有见到过这样的函数调用方式，所以引起了我的注意，我决定去研究研究；下面就将我的研究分享出来，当然如果你对此已经有了深刻的认识，你可以选择跳过，或者对于不足的地方，给出指点。好了闲话不多说，进入正题。



我们来看一个最简单的例子：



function create1(pro) {

console.log("pro : " + pro);

return function(obj1, obj2){

console.log(obj1 + " -- " + obj2);

return obj1 + obj2;

}

}



我构建了一个简单的函数create1，并且有一个返回值，返回值是一个内部函数。函数构建完了，接下来进行调用：

var c1 = create1("pro"); // 创建函数

如果按照我之前的理解，当我调用了这个方法后，应该会打印出 pro : pro，接着然后报错的。如果你看完过后，也跟我有一样的想法，那恭喜你想多了或者有了固型思维



。真实的是当我们通过上面的代码调用的时候，日志是打印出了  pro : pro ,但是并没有报错，并且我们反复来回的调用过后，也只是来回的打印相同的日志。这也就说明这个时候，只是进入了create1()方法，并没有进入到该函数的内部函数内。通过面试题的启发，我在试着调用了一次，发现打印出了后续的。

c1(1, 2); // 调用函数

这样就打印出了下面的日志；**这说明其实我们一开始调用方法的时候，其实是并没有进入到里层的函数的，只是进入了外层函数体，**

**我们只有再调用才能进入里层函数体，并且这个时候，我们重复上面的调用，他只会是调用里层的函数体，并没有外面的函数体。**

类似这种函数返回另一个函数的，我们第一次调用只是构建了一个外层函数体对象，只有有后续的调用，才能调用内层函数体，并且重复调用，只会重复内层函数体。  
不要急，还没有完，后面还有……

接下来，我们看一看另一种情况，我们先声明一个函数，用来做加法运算：

function infun(obj1, obj2) {

console.log(obj1 + " -- " + obj2);

return obj1 + obj2;

}

然后再声明一个函数，在该函数中调用上面声明的函数：

function create2(pro) {

console.log("pro = " + pro);

return infun(obj1, obj2); // 这个时候,会报错

}

最后是调用：

var c1 = create2("pro");

查看日志：



pro = pro

‌Uncaught ReferenceError: obj1 is not defined

会发现，打印出了一条日志后，接着抛出了异常。对方法做一下改动，

function create2(pro) {

console.log("pro = " + pro);

var obj1 = 1, obj2 = 2;

return infun(obj1, obj2); // 这个时候,在内部设了传的变量，则不报错了，事实是调用了外面已经声明过的函数，ifun(obj1,obj2),而obj1与obj2两个传参变量由于是在调用已声明过的ifun函数，所以不能当做声明的传参变量，

　　　　　　　　　　　　　　　　　　//只当传参的字符串变量（准确点应该说，与obj1 obj2这两变量名本身没任何关系了，名字可以为任何变量名，前提是已经声明过赋过值），而函数内部和全局都未定义这两变量的值，所以抛出异常，is not defined   
}



在调用会发现正常运行，并且打印出了两条日志记录。

这说明，类似于这种，在一个函数内返回一个已经声明的函数，其实是调用已经声明的函数，跟上面的情况是不一样的。  
好了，现在回过头来，仔细看看开头的面试题，就会发现一切都明了了：



// 声明一个函数表达式

var add = function(x){

var sum = 1;

// 在函数表达式内部有一个求和的内部函数

var tmp = function(x){

sum = sum + x;// 求和

return tmp;

}

// 构建一个函数体的toString()函数

tmp.toString = function(){

return sum;

}

return tmp; // 返回的是一个函数体,如果该函数体有toString()方法,则会调用函数体的toString()方法

}



然后再来看看调用：

alert(add(1)(2)(3))

**重点：**add(1)(2)(3)　  
1、函数add(1)第一次调用，其实是只声明了　var sum=1;这个变量，然后返回了tmp函数体，用于后面调用tmp函数  
2、函数add(1)(2)第二次调用才真正的把参数传进来使用了，即第一次传的　1　是没地方用的，没意义，第二次传的　2　是给第一次返回的tmp函数体传的参、即用在sum=sum+x上----sum=1+2  
3、函数add(1)(2)(3)第三次调用和第二次一样，由于tmp函数体内部　return tmp  返回了本身，所以后面可以继续调用tmp函数，也就是除第一次调用传参无效外，后面可以调用无数次，sum值会不断累加  
4、toString是tmp函数体附带的属性方法函数，会随着主体函数toString执行一次调用一次

结果为6，至于原因就跟我们第一种讨论的情况一样，接下来，我们反复调用：

// 以下结果输出为：6

alert(add(10)(2)(3))

alert(add(100)(2)(3))

// 下面的结果输出变了

alert(add(1)(3)(3))

alert(add(1)(2)(5))