|  |
| --- |
| function Foo() {      var i = 0;  **return** function() {          console.log(i++);      }  }    var f1 = Foo(),      f2 = Foo();  f1();0  f1();1  f2();0 |

1. 闭包例题

（1）function是引用类型：保存在堆中，变量f1,f2是保存在栈中；

（2）闭包：一般来说**函数执行完后它的局部变量就会随着函数调用结束被销毁**，但是此题Foo函数返回了一个匿名函数的引用（即一个闭包），它可以访问到Foo()被调用产生的环境，而局部变量i一直处在这个环境中，只要一个环境有可能被访问到，它就不会被销毁，所以说**闭包有延续变量作用域的功能**。这就好理解为什么：f1();//0 f1();//1

(3)   过程：

  第一次f1()  :f1=Foo()中，先执行Foo(): i = 0,return值返回给f1

 (f1指向子函数   f1()=function(){.....},因为子函数没有 定义i，所以向上找到父函数定义的 i:  )并执行子函数 先输出i=0,再自加 i =1(覆盖了父函数Foo 的 i值);

  第二次f1() : 执行的是子函数 Function(){  ..},先输出的是父函数 的 i=1,再自加 i =2;

  第一次f2():同第一次f1(),不同的是 **f2指向堆中一个新的对象 function(){ ...},**所有此i非彼i,输出i=0;如果

   如果再次f2(),那么和第二次f1(),一样输出i=1;

<SCRIPT LANGUAGE="JavaScript">

var bb = 1;

function aa(bb) {

    bb = 2;

    alert(bb);

};

aa(bb);2

alert(bb);1

</SCRIPT>

2.函数体内，bb并没有使用var来定义，按理说这个bb在预处理的时候应该是window的属性。但在这里，函数声明的时候，带了一个参数bb，也就是相当于在函数体内声明了var bb。所以，函数里的bb就是函数活动对象的属性。所以函数执行时会输出2。函数执行完后，函数的活动对象被销毁，也就是局部的这个bb被删除了，执行流进入到window，再输出bb，值就是1了。