闭包

2020年03月18日, 星期三 14:49

思考：

函数内的局部变量，是否能在函数外得到？ 不能

有什么方法能读写函数内部的局部变量？ **闭包**

```
function f1() {
    var n = 999;

    function f2() {
        console.log(n++);
    }
    return f2;
}
var result = f1();
result();
result();
```

1.闭包（closure）

- 函数与对其词法环境的引用共同构成闭包。
- 闭包是由函数以及创建该函数的词法环境组合而成。这个环境包含了这个闭包创建时所需访问的所有局部变量。

```
function f1(  ) {
    var x = 1;

    function f2() {
        return x++;
    }
    return f2;
}
var f3 = f1();
```

2.闭包的作用

可通过闭包访问函数作用域内的局部变量

使函数中的变量被保存在内存中不被释放

3.闭包的缺点

由于闭包会使得函数中的变量都被保存在内存中，内存消耗大

闭包会在父函数外部，改变父函数内部变量的值

4.闭包的应用——IIFE


```
for(var i = 0; i < tabs.length; i++) {
    ! function(i) {//IIFE start
        tabs[i].onclick = function() {
            for(var j = 0; j < tabs.length; j++) {
                tabs[j].className = "";
            }
            this.className = "active";
            contents.innerHTML = "点击了" + i;
        }
    }(i);//IIFE end
}
```

IIFE —— 解决变量共享、变量污染问题

5.闭包的常见形式—— 以普通函数形式返回

```
var tmp = 100;

function foo() {
    var tmp = 3;
    return function(y) {
        console.log(x + y + (++tmp));
    }
}
var fee = foo(2);
fee(10);
fee(10);
fee(10);
```

```
var outer;

function foo() {
    var b = "local";

    function inner() {
        return b;
    }

    outer = inner;
}
foo();
console.log(outer());
```

```
function outer(fn) {
    console.log(fn());
}

function foo() {
    var b = "local";

    function inner() {
        return b;
    }

    outer(inner);
}
foo();
```

6.闭包的常见形式—— 作为对象的方法返回

```
function counter(  ) {

    var n = 0;
    return {
        count: function() { return ++n;},
        reset: function() { n = 0;return n;}
    }
}
```

```
var c = counter();
var d = counter();
console.log(c.count());
console.log(d.count());
console.log(c.reset());
console.log(c.count());
console.log(d.count());
```