学习JavaScript闭包

【思考：外面认得return，却认不出return之物，所以得找个子函数来return】

作者： 阮一峰

日期： 2009年8月30日

闭包（closure）是Javascript语言的一个难点，也是它的特色，很多高级应用都要依靠闭包实现。

下面就是我的学习笔记，对于Javascript初学者应该是很有用的。

一、变量的作用域

要理解闭包，首先必须理解Javascript特殊的变量作用域。

变量的作用域无非就是两种：全局变量和局部变量。

Javascript语言的特殊之处，就在于函数内部可以直接读取全局变量。

```
var n=999;
function f1(){
    alert(n);
}
f1(); // 999
```

另一方面，在函数外部自然无法读取函数内的局部变量。

```
function f1(){
    var n=999;
}
alert(n); // error
```

这里有一个地方需要注意，函数内部声明变量的时候，一定要使用var命令。如果不用的话，你实际上声明了一个全局变量！

```
function f1(){
    n=999;
}
f1();
alert(n); // 999
```

二、如何从外部读取局部变量？

出于种种原因，我们有时候需要得到函数内的局部变量。但是，前面已经说过了，正常情况下，这是办不到的，只有通过变通方法才能实现。

那就是在函数的内部，再定义一个函数。

```
function f1(){
    var n=999;
    function f2(){
        alert(n); // 999
    }
}
```

在上面的代码中，函数f2就被包括在函数f1内部，这时f1内部的所有局部变量，对f2都是可见的。但是反过来就不行，f2内部的局部变量，对f1就是不可见的。这就是Javascript语言特有的"链式作用域"结构（chain scope），子对象会一级一级地向上寻找所有父对象的变量。所以，父对象的所有变量，对子对象都是可见的，反之则不成立。

既然f2可以读取f1中的局部变量，那么只要把f2作为返回值，我们不就可以在f1外部读取它的内部变量了吗！

```
function f1(){
    var n=999;
    function f2(){
        alert(n);
    }
    return f2;
}
var result=f1();
result(); // 999
```

三、闭包的概念

上一节代码中的f2函数，就是闭包。

各种专业文献上的"闭包"（closure）定义非常抽象，很难看懂。我的理解是，闭包就是能够读取其他函数内部变量的函数。

由于在Javascript语言中，只有函数内部的子函数才能读取局部变量，因此可以把闭包简单理解成"定义在一个函数内部的函数"。

所以，在本质上，闭包就是将函数内部和函数外部连接起来的一座桥梁。

四、闭包的用途

闭包可以用在许多地方。它的最大用处有两个，一个是前面提到的可以读取函数内部的变量，另一个就是让这些变量的值始终保持在内存中。

怎么来理解这句话呢？请看下面的代码。

```
function f1(){
    var n=999;
    nAdd=function(){n+=1;};
    function f2(){
        alert(n);
    }
    return f2;
}
var result=f1();
result(); // 999
nAdd();
result(); // 1000
```

在这段代码中，result实际上就是闭包f2函数。它一共运行了两次，第一次的值是999，第二次的值是1000。这证明了，函数f1中的局部变量n一直保存在内存中，并没有在f1调用后被自动清除。

为什么会这样呢？原因就在于f1是f2的父函数，而f2被赋给了一个全局变量，这导致f2始终在内存中，而f2的存在依赖于f1，因此f1也始终在内存中，不会在调用结束后，被垃圾回收机制（garbage collection）回收。

这段代码中另一个值得注意的地方，就是"nAdd=function(){n+=1}"这一行，首先在nAdd前面没有使用var关键字，因此nAdd是一个全局变量，而不是局部变量。其次，nAdd的值是一个匿名函数（anonymous function），而这个匿名函数本身也是一个闭包，所以nAdd相当于是一个setter，可以在函数外部对函数内部的局部变量进行操作。

五、使用闭包的注意点

1）由于闭包会使得函数中的变量都被保存在内存中，内存消耗很大，所以不能滥用闭包，否则会造成网页的性能问题，在IE中可能导致内存泄露。解决方法是，在退出函数之前，将不使用的局部变量全部删除。

2）闭包会在父函数外部，改变父函数内部变量的值。所以，如果你把父函数当作对象（object）使用，把闭包当作它的公用方法（Public Method），把内部变量当作它的私有属性（private value），这时一定要小心，不要随便改变父函数内部变量的值。

六、思考题

如果你能理解下面两段代码的运行结果，应该就算理解闭包的运行机制了。

代码片段一。

```
var name = "The Window";
var object = {
    name : "My Object",
    getNameFunc : function(){
        return function(){
            return this.name;
        };
    }
};
alert(object.getNameFunc()());
```

代码片段二。

```
var name = "The Window";
var object = {
    name : "My Object",
    getNameFunc : function(){
        var that = this;
        return function(){
            return that.name;
        };
    }
};
alert(object.getNameFunc()());
```

（完）

思考题答案：The Window、My Object；其实最关键的就是要明白作用域链和闭包所起到的作用，其实闭包说到底就是一个函数，而且函数调用返回后其资源所占用的栈区并没有释放，，所以变量就还保存在内存中，由于作用域链的关系，它会去寻找离其最近的var声明，var声明所在就是该闭包所在的层，从而在这层里面所得到的结果就是它最后的值，于是乎返回值也就是这个最终的值了,因此这个this就不能调用函数体内的全局对象，而是他的局

部对象object.name.产生闭包效果的环境必须是嵌套函数的引用被保存到了一个全局作用域里面.

创建函数时的this与调用函数时的this指向可能不一致，因此需要一个中介变量保存函数创建时的this所指：