2016年11月2日 星期三

闭包

每次定义一个函数，都会产生一个作用域链（scope chain）。当JavaScript寻找变量varible时（这个过程称为变量解析），总会优先在当前作用域链的第一个对象中查找属性varible，如果找到，则直接使用这个属性；否则，继续查找下一个对象的是否存在这个属性；这个过程会持续直至找到这个属性或者最终未找到引发错误为止。

```
(function(){
    var hello="hello,world";
    function welcome(hi){
        alert(hi);       //解析到作用域链的第一个对象的属性
        alert(hello);    //解析到作用域链的第二个对象的属性
    }
    welcome("It's easy");
})();
```

分析过程如下：

对于函数welcome()，定义welcome的时候会产生一个作用域链对象，为了表示方便，记作scopechain。scopechain是个有顺序的集合对象。

scopechain的第一个对象：为了方便表示记作sc1， sc1有若干属性，引用本函数的参数和局部变量，如sc1.hi ；

scopechain的第二个对象：为了方便表示记作sc2，sc2有若干属性，引用外层函数的参数和局部变量，如sc2.hello；

...

scopechain的最后一个对象：为了方便表示记作scn，scn引用的全局的执行环境对象，也就是window对象！，如scn.eval()；

2、假闭包

```
function foo(x) {
```

```
        var tmp = 3;
        function bar(y) {
                alert(x + y + (++tmp));
        }
        bar(10);
        }
    foo(2)
```

3、真闭包

闭包是从用户角度考虑的一种设计概念，它基于对上下文的分析，把龌龊的事情、复杂的事情和外部环境交互的事情都自己做了，留给用户一个很自然的接口。

```
    function foo(x) {
        var tmp = 3;
        return function (y) {
                alert(x + y + (++tmp));
        }
    }
    var bar = foo(2); // bar 现在是一个闭包
    bar(10);
```

上面的x是一个字面值（值传递），和JS里其他的字面值一样，当调用foo时，实参x的值被复制了一份，复制的那一份作为了foo的参数x。

那么问题来了，JS里处理object时是用到引用传递的，那么，你调用foo时传递一个object，foo函数return的闭包也会引用最初那个object！

如果一个函数访问了它的外部变量，那么它就是一个闭包。

注意，外部函数不是必需的。通过访问外部变量，一个闭包可以维持（keep alive）这些变量。在内部函数和外部函数的例子中，外部函数可以创建局部变量，并且最终退出；

但是，如果任何一个或多个内部函数在它退出后却没有退出，那么内部函数就维持了外部函数的局部数据。这里有利也有弊，当要维持这个数据且保持神秘性的时候，它是有利的；但不需要后续操作时，它浪费内存。

内存泄漏

内存泄露是指一块被分配的内存既不能使用，又不能回收，直到浏览器进程结束。在C++中，因为是手动管理内存，内存泄露是经常出现的事情。而现在流行的C#和Java等语言采用了自动垃圾回收方法管理内存，正常使用的情况下几乎不会发生内存泄露。浏览器中也是采用自动垃圾回收方法管理内存，但由于浏览器垃圾回收方法有bug，会产生内存泄露。

内存泄露的几种情况

1、当页面中元素被移除或替换时，若元素绑定的事件仍没被移除，在IE中不会作出恰当处理，此时要先手工移除事件，不然会存在内存泄露。

```
<div id="myDiv">
    <input type="button" value="Click me" id="myBtn">
</div>
<script type="text/javascript">
    var btn = document.getElementById("myBtn");
    btn.onclick = function(){
        document.getElementById("myDiv").innerHTML = "Processing...";
    }
</script>
```

怎么解决呢?

```
btn.onclick = function(){
    btn.onclick = null;
    document.getElementById("myDiv").innerHTML = "Processing...";
}
```

或者采用事件委托

```
<div id="myDiv">
    <input type="button" value="Click me" id="myBtn">
</div>
<script type="text/javascript">
    document.onclick = function(event){
        event = event || window.event;
        if(event.target.id == "myBtn"){
            document.getElementById("myDiv").innerHTML = "Processing...";
        }
    }
</script>
```

2、DOM对象循环应用

3、闭包可以维持函数内局部变量，使其得不到释放。

```
function bindEvent()
{
     var obj=document.createElement("XXX");
    obj.onclick=function(){
    //Even if it's a empty function
     }
}
```

解决方案：

a、定义事件回调时，由于是函数内定义函数，并且内部函数—事件回调的引用外暴了，形成了闭包。解决之道，将事件处理函数定义在外部，解除闭包

b、手动释放对象内存

```
    obj.onclick=function(){
        //Even if it's a empty function
    }
    obj=null;
```

4、 a = {p: {x: 1}};

```
b = a.p;
delete a.p;
```

执行这段代码之后b.x的值依然是1.由于已经删除的属性引用依然存在，因此在JavaScript的某些实现中，可能因为这种不严谨的代码而造成内存泄露。所以在销毁对象的时候，要遍历属性中属性，依次删除。

5、自动类型装箱转换

下面的代码在ie系列中会导致内存泄露

```
var s=”lalala”;
alert(s.length);
```

s本身是一个string而非object，它没有length属性，所以当访问length时，JS引擎会自动创建一个临时String对象封装s，而这个对象一定会泄露。这个bug匪夷所思，所幸解决起来相当容易，记得所有值类型做.运算之前先显式转换一下：

```
var s="lalala";

alert(new String(s).length);
```