1、什么是闭包、以及闭包所涉及的作用域链这里就不说了。

2、JavaScript垃圾回收机制

     JavaScript不需要手动地释放内存，它使用一种自动垃圾回收机制（garbage collection）。当一个对象无用的时候，即程序中无变量引用这个对象时，就会从内存中释放掉这个变量。

[?](http://www.cnblogs.com/jenry/archive/2010/12/08/1900211.html)

|  |  |
| --- | --- |
| 1  2  3 | var s = [ 1, 2 ,3];      var s = null;      //这样原始的数组[1 ,2 ,3]就会被释放掉了。 |

3、循环引用

     三个对象 A 、B 、C

     ABC ：A的某一属性引用着B，同样C也被B的属性引用着。如果将A清除，那么B、C也被释放。

     ABCB ：这里增加了C的某一属性引用B对象，如果这是清除A，那么B、C不会被释放，因为B和C之间产生了循环引用。

[?](http://www.cnblogs.com/jenry/archive/2010/12/08/1900211.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | var a = {};     a.pro = { a:100 };     a.pro.pro = { b:100 };     a = null ;     //这种情况下，{a:100}和{b:100}就同时也被释放了。       var obj = {};     obj.pro = { a : 100 };     obj.pro.pro = { b : 200 };     var two = obj.pro.pro;     obj = null;     //这种情况下 {b:200}不会被释放掉，而{a:100}被释放了。 |

4、循环引用和闭包

[?](http://www.cnblogs.com/jenry/archive/2010/12/08/1900211.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | function outer(){          var obj = {};          function inner(){              //这里引用了obj对象          }          obj.inner = inner;      } |

这是一种及其隐蔽的循环引用，。当调用一次outer时，就会在其内部创建obj和inner两个对象，obj的inner属性引用了inner；同样inner也引用了obj，这是因为obj仍然在innerFun的封闭环境中，准确的讲这是由于JavaScript特有的“作用域链”。  
因此，闭包非常容易创建循环引用，幸运的是JavaScript能够很好的处理这种循环引用。

5、IE中的内存泄漏

    IE中的内存泄漏有好几种，这里有详细的解释（<http://msdn.microsoft.com/en-us/library/bb250448.aspx>），园子里也有翻译了（<http://www.cnblogs.com/birdshome/archive/2006/05/28/ie_memoryleak.html>）。

    这里只讨论其中一种，即循环引用所造成的内存泄漏，因为，这是一种最普遍的情况。

    当在DOM元素或一个ActiveX对象与普通JavaScript对象之间存在循环引用时，IE在释放这类变量时存在特殊的困难，最好手动切断循环引用，这个bug在IE 7中已经被修复了（<http://www.quirksmode.org/blog/archives/2006/04/ie_7_and_javasc.html>）。

   “IE 6 suffered from [memory leaks](http://www.quirksmode.org/blog/archives/coding_techniques/memory_leaks/index.html) when a circular reference between several objects, among which at least one DOM node, was created. This problem has been solved in IE 7. ”

    如果上面的例子（第4点）中obj引用的不是一个JavaScript Function对象（inner），而是一个ActiveX对象或Dom元素，这样在IE中所形成的循环引用无法得到释放。

[?](http://www.cnblogs.com/jenry/archive/2010/12/08/1900211.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | function init(){          var elem = document.getElementByid( 'id' );          elem.onclick = function(){              alert('rain-man');              //这里引用了elem元素          };      } |

Elem引用了它的click事件的监听函数，同样该函数通过其作用域链也引用回了elem元素。这样在IE中即使离开当前页面也不会释放这些循环引用。

6、解决方法

   基本的方法就是手动清除这种循环引用，下面一个十分简单的例子，实际应用时可以自己构建一个addEvent()函数，并且在window的unload事件上对所有事件绑定进行清除。

[?](http://www.cnblogs.com/jenry/archive/2010/12/08/1900211.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | function outer(){          var one = document.getElementById( 'one' );          one.onclick = function(){};      }      window.onunload = function(){          var one = document.getElementById( 'one' );          one.onclick = null;      }; |

其它方法（by:[Douglas Crockford](http://www.crockford.com/)）

[?](http://www.cnblogs.com/jenry/archive/2010/12/08/1900211.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | /\*\*   \* 遍历某一元素节点及其所有后代元素   \*   \* @param Elem node  所要清除的元素节点   \* @param function func  进行处理的函数   \*   \*/  function walkTheDOM(node, func) {      func(node);      node = node.firstChild;      while (node) {          walkTheDOM(node, func);          node = node.nextSibling;      }  }  /\*\*   \* 清除dom节点的所有引用，防止内存泄露   \*   \* @param Elem node  所要清除的元素节点   \*   \*/  function purgeEventHandlers(node) {      walkTheDOM(node, function (e) {          for (var n in e) {              if (typeof e[n] ===                      'function') {                  e[n] = null;              }          }      }); |