[**(转)JavaScript中的匿名函数及函数的闭包**](http://www.cnblogs.com/jenry/archive/2010/12/08/1900591.html)

匿名函数

函数是JavaScript中最灵活的一种对象，这里只是讲解其匿名函数的用途。匿名函数：就是没有函数名的函数。

1、函数的定义，首先简单介绍一下函数的定义，大致可分为三种方式

第一种：这也是最常规的一种

function double( x ){  
 return 2 \* x;   
}

第二种：这种方法使用了Function构造函数，把参数列表和函数体都作为字符串，很不方便，不建议使用。

var double = new Function( 'x' , ' return 2 \* x;');

第三种：

var double = function( x ) { return 2\* x; }

注意 '='右边的函数就是一个匿名函数，创造完毕函数后，又将该函数赋给了变量square。

2、匿名函数的创建

第一种方式：就是上面所讲的定义square函数，这也是最常用的方式之一。

第二种方式：

(function( x , y){  
 alert( x + y);   
})(2 ,3 );

这里创建了一个匿名函数(在第一个括号内)，第二个括号用于调用该匿名函数，并传入参数。

闭包

闭包的英文单词是closure，这是JavaScript中非常重要的一部分知识，因为使用闭包可以大大减少我们的代码量，使我们的代码看上去更加清晰等等，总之功能十分强大。

闭包的含义：闭包说白了就是函数的嵌套，内层的函数可以使用外层函数的所有变量，即使外层函数已经执行完毕（这点涉及[JavaScript作用域链](http://www.cnblogs.com/rainman/archive/2009/04/28/1445687.html)）。

示例一：

function checkClosure(){  
 var str = 'rain-man';  
 setTimeout(  
 function(){ alert( str ); } //这是一个匿名函数  
 , 2000);  
}  
checkClosure();

这个例子看上去十分的简单，仔细分析下它的执行过程还是有许多知识点的：checkClosure函数的执行是瞬间的（也许用时只是0.00001毫秒），在checkClosure的函数体内创建了一个变量str，在checkClosure执行完毕之后str并没有被释放，这是因为setTimeout 内的匿名函数存在这对str的引用。待到2秒后函数体内的匿名函数被执行完毕,str才被释放。

示例二：优化代码

function forTimeout( x, y){  
 alert( x + y );  
}  
function delay( x , y , time ){  
 setTimeout( 'forTimeout(' + x + ',' + y + ')' , time );   
}  
/\*\*  
 \*上面的delay函数十分难以阅读，也不容易编写，但如果使用闭包就可以让代码更加清晰  
  
function delay( x , y , time ){  
 setTimeout(  
 function(){  
 forTimeout( x , y )   
 }   
 , time );   
}  
  
\*/

举例

匿名函数最大的用途是创建闭包（这是JavaScript语言的特性之一），并且还可以构建命名空间，以减少全局变量的使用。

示例三：

var oEvent = {};  
(function(){   
var addEvent = function(){ /\*代码的实现省略了\*/ };  
function removeEvent(){}  
  
oEvent.addEvent = addEvent;  
oEvent.removeEvent = removeEvent;  
})();

在这段代码中函数addEvent和removeEvent都是局部变量，但我们可以通过全局变量oEvent使用它，这就大大减少了全局变量的使用，增强了网页的安全性。

我们要想使用此段代码：oEvent.addEvent( document.getElementById('box') , 'click' , function(){} );

示例四：

var rainman = (function( x , y ){  
 return x + y;  
})( 2 , 3 );  
/\*\*  
 \*也可以写成下面的形式，因为第一个括号只是帮助我们阅读，但是不推荐使用下面这种书写格式。  
var rainman = function( x , y ){  
 return x + y;  
}( 2 , 3 );  
\*/

在这里我们创建了一个变量rainman，并通过直接调用匿名函数初始化为5，这种小技巧有时十分实用。

示例五：

var outer = null;  
  
(function(){  
var one = 1;  
function inner (){  
 one += 1;  
 alert( one );  
}  
outer = inner;  
})();  
  
outer(); //2  
outer(); //3  
outer(); //4

这段代码中的变量one是一个局部变量（因为它被定义在一个函数之内），因此外部是不可以访问的。但是这里我们创建了inner函数，inner函数是可以访问变量one的；又将全局变量outer引用了inner，所以三次调用outer会弹出递增的结果。

注意

一：闭包允许内层函数引用父函数中的变量，但是该变量是最终值。

示例六：

/\*\*  
<body>  
<ul>  
 <li>one</li>  
 <li>two</li>  
 <li>three</li>  
 <li>one</li>  
</ul>  
\*/  
   
var lists = document.getElementsByTagName('li');  
for(var i = 0 , len = lists.length ; i < len ; i++ ){  
 lists[ i ].onmouseover = function(){  
 alert( i );   
 };  
}

你会发现当鼠标移过每一个<li>元素时，总是弹出4，而不是我们期待的元素下标。这是为什么呢？注意事项里已经讲了（最终值）。显然这种解释过于简单，当mouseover事件调用监听函数时，首先在匿名函数（ function(){ alert(i); }）内部查找是否定义了 i，结果是没有定义；因此它会向上查找，查找结果是已经定义了，并且i的值是4（循环后的i值）；所以，最终每次弹出的都是4。

解决方法一：

var lists = document.getElementsByTagName('li');  
for(var i = 0 , len = lists.length ; i < len ; i++ ){  
 (function( index ){  
 lists[ index ].onmouseover = function(){  
 alert( index );   
 };   
 })( i );  
}

解决方法二：

var lists = document.getElementsByTagName('li');  
for(var i = 0 , len = lists.length ; i < len ; i++ ){  
 lists[ i ].$$index = i; //通过在Dom元素上绑定$$index属性记录下标  
 lists[ i ].onmouseover = function(){  
 alert( this.$$index );   
 };  
}

解决方法三：

function eventListener( list , index ){  
 list.onmouseover = function(){  
 alert(index);  
 };  
}  
var lists = document.getElementsByTagName('li');  
for(var i = 0 , len = lists.length ; i < len ; i++ ){  
 eventListener( lists[ i ] , i );  
}