闭包：1、指有权访问另一个函数作用域中的变量的函数

2、能够读取其他函数内部变量的函数

3、定义在一个函数内部的函数（因为函数内部的子函数才能读取局部变量）

将函数内部和函数外部连接起来的一座桥梁

创建闭包的常见方式：在一个函数内部创建另一个函数

闭包是一种函数、一种特殊的函数。

闭包定义在一个外部函数内部，并且能够访问外部函数中自由变量的函数。

从定义上看涉及以下名词：函数、函数内部、作用域、自由变量、内部变量

作用域：全局变量、局部变量

函数内部可以直接读取全局变量

函数内部声明变量一定要使用“var”，如果不使用变量变成全局变量

作用域链：子对象会一级一级地向上寻找所有父对象的变量，直到最后没找到所需要的变量，返回undefined。父对象的所有变量对子对象都是可见的，反之不成立

自由变量：凡是跨了自己的作用域的变量

闭包特性：1、函数嵌套函数

2、函数内部可以引用外部的参数和变量

3、参数和变量不会被垃圾回收机制回收

闭包使用场合：设计私有的方法和变量

闭包缺点：常驻内存，增大内存使用量，使用不当很容易造成内存泄露。

function fun1(){

var num = 999;

num\_Add = function(){num += 1};

function fun2(){

console.log(num);

};

return fun2;

}

var result = fun1();

result(); //输出999

num\_Add(); //num = 1000 说明num常驻内存，没有被回收

result(); //输出1000

//num\_Add没有使用var声明变量，num\_Add是全局变量，它的值是一个匿名函数，而这个匿名函数本身也是一个闭包，所以可以在函数外部对函数内部的局部变量进行操作。

垃圾（内存）回收机制：JS解释器会自动分配和回收内存。

1、一个对象不再被引用，这个对象就会被GC回收

2、如果两个对象互相引用，而不被第3都引用，这两个对象也会被回收。

两个经典的闭包和this指向的栗子：

getNameFunc = function(){}

var getnameFunc = function(){}

var name = "The Window";

var object = {

　　name : "My Object",

　　getNameFunc : function(){

　　　　return function(){

　　　　　　return this.name;

　　　　};

　　}

　};

console.log(object.getNameFunc()());

===> The Window

var name = "The Window";

var object = {

name : "My Object",

getNameFunc : function(){

var that = this; //把this赋值给变量that

//console.log(that);

return function(){

return that.name;

};

}

};

console.log(object.getNameFunc()());

===> My Object

