维基百科中闭包的描述：

在计算机科学中闭包（也称为词法闭包或函数闭包）是指一个函数或者函数引用，与引用环境绑定在一起，这个引用环境是一个存储该函数每个非局部变量的表。

-闭包不同于一般的函数，它允许一个函数在立即词法作用域外调用时，仍可访问非本地变量。

实际种处处是闭包

如

Function(){

Var loaclData=”localData here”;

Document.addEventListener(

‘click’,function(){

Console.log(loacalDara); 这就是闭包,读取了外部的localdata

});

}();

再如

function f1(){

　　　　var n=999;

　　　　function f2(){  
　　　　　　return n;   
　　　　}

　　　　return f2;

　　}

　　var result=f1();

alert(result); // 999

在外部访问了f1里面的参数，这个f2函数就是闭包。

**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。**

**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。**

**闭包的另一个封装作用**

**如**

**(function(){**

**var \_userId=123;**

**var \_typeID="item";**

**var export={}; //定义一个export对象**

**export.getUserId=function(){**

**return \_userId;**

**}**

**export.getTypeId=function(){**

**return \_TypeId;**

**}**

**window.export=export;**

**}());**

**这样子在外部，export.\_userId //undefined**

**export.\_typeid //undefined**

**但是通过export.getUserId()//123**

**Export.getTypeId()//’titem’**