在 javascript 中存在着 this 和 scope 两个概念，如果不细心了解，还真搞不清楚这两个家伙，今天我们就来详细了解一下 this 和 scope 的区别以及它们的作用，最后会附上 code 以加深理解。

一、作用域（scope）

所谓作用域就是：变量在声明它们的函数体以及这个函数体嵌套的任意函数体内都是有定义的。

```
function scope(){
    var foo = "global";
    if(window.getComputedStyle){
        var a = "I'm if";
        console.log("if:"+foo); //if:global
    }
    while(1){
        var b = "I'm while";
        console.log("while:"+foo);//while:global
        break;
    }
    !function (){
        var c = "I'm function";
        console.log("function:"+foo);//function:global
    }();
    console.log(
         foo,//global
         a, // I'm if
         b, // I'm while
         c  // c is not defined
    );
}
scope();
```

（1）scope 函数中定义的 foo 变量，除过自身可以访问以外，还可以在 if 语句、while 语句和内嵌的匿名函数中访问。 因此，foo 的作用域就是 scope 函数体。

（2）在 javascript 中，if、while、for 等代码块不能形成独立的作用域。因此，javascript 中没有块级作用域，只有函数作用域。

但是，在 JS 中有一种特殊情况：

如果一个变量没有使用 var 声明，window 便拥有了该属性，因此这个变量的作用域不属于某一个函数体，而是 window 对象。

```
function varscope(){
```

```
2      foo = "I'm in function";
3      console.log(foo);//I'm in function
4  }
5  varscope();
6  console.log(window.foo); //I'm in function
```

二、作用域链（scope chain）

所谓作用域链就是：一个函数体中嵌套了多层函数体，并在不同的函数体中定义了同一变量， 当其中一个函数访问这个变量时，便会形成一条作用域链（scope chain）。

```
 1 foo = "window";
 2 function first(){
 3     var foo = "first";
 4     function second(){
 5         var foo = "second";
 6         console.log(foo);
 7     }
 8     function third(){
 9         console.log(foo);
10     }
11     second(); //second
12     third();  //first
13 }
14 first();
```

当执行 second 时，JS 引擎会将 second 的作用域放置链表的头部，其次是 first 的作用域，最后是 window 对象，于是会形成如下作用域链：

second->first->window， 此时，JS 引擎沿着该作用域链查找变量 foo，查到的是"second"

当执行 third 时，third 形成的作用域链：third->first->window，因此查到的是："frist"

特殊情况：with 语句

JS 中的 with 语句主要用来临时扩展作用域链，将语句中的对象添加到作用域的头部。with 语句结束后，作用域链恢复正常。

```
 1 foo = "window";
 2 function first(){
 3     var foo = "first";
 4     function second(){
```

```
5         var foo = "second";
6         console.log(foo);
7     }
8     function third(obj){
9         console.log(foo); //first
10        with (obj){
11            console.log(foo); //obj
12        }
13        console.log(foo); //first
14    }
15    var obj = {foo:'obj'};
16    third(obj);
17 }
18 first();
```

在执行 third()时，传递了一个 obj 对象，obj 中有属性 foo，在执行 with 语句时，JS 引擎将 obj 放置在了原链表的头部，于是形成的作用域链如下：

obj->third->first->window，此时查找到的 foo 就是 obj 中的 foo,因此输出的是："obj"，而在 with 之前和之后，都是沿着原来的链表进行查找，从而说明，在 with 语句结束后，作用域链已恢复正常。

三、this

在一个函数中，this 总是指向当前函数的所有者对象，this 总是在运行时才能确定其具体的指向，也才能知道它的调用对象。

这句话总结了关于 this 的一切，切记，切记，切记！(ps:重要的事情说三遍!)

```
1 window.name = "window";
2 function f(){
3     console.log(this.name);
4 }
5 f();//window
6
7 var obj = {name:'obj'};
8 f.call(obj); //obj
```

在执行 f()时，此时 f()的调用者是 window 对象，因此输出"window"

f.call(obj) 是把 f()放在 obj 对象上执行，相当于 obj.f(),此时 f 中的 this 就是 obj,所以输出的是"obj"

四、实战应用

code1：

```
 1 var foo = "window";
 2 var obj = {
 3     foo : "obj",
 4     getFoo : function(){
 5         return function(){
 6             return this.foo;
 7         };
 8     }
 9 };
10 var f = obj.getFoo();
11 f(); //window
```

code2:

```
var foo = "window";
var obj = {
    foo : "obj",
    getFoo : function(){
        var that = this;
        return function(){
            return that.foo;
        };
    }
};
var f = obj.getFoo();
f(); //obj
```

code1 和 code2 是对 this 和 scope 最好的总结，如果对于运行结果有疑惑，说明你还没有理解 this 和 scope，请多看几遍本文

代码解析：

code1:
执行 var f = obj.getFoo()返回的是一个匿名函数，相当于:

```
var f = function(){
     return this.foo;
}
```

f() 相当于 window.f(), 因此 f 中的 this 指向的是 window 对象，this.foo 相当于 window.foo, 所以 f()返回"window"

code2:
执行 var f = obj.getFoo() 同样返回匿名函数，即：

```
var f = function(){
     return that.foo;
}
```

唯一不同的是 f 中的 this 变成了 that, 要知道 that 是哪个对象之前，先确定 f 的作用域链：f->getFoo->window 并在该链条上查找 that,此时可以发现 that 指代的是 getFoo 中的 this, getFoo 中的 this 指向其运行时的调用者，从 var f = obj.getFoo() 可知此时 this 指向的是 obj 对象，因此 that.foo 就相当于 obj.foo,所以 f()返回"obj"