**作用域**

为引擎提供了环境内每一个标识符的位置信息，引擎依赖这些信息可以迅速查找到它们定义的位置。标识符的位置信息，是你在写代码时将标识符写在哪里来决定的，在词法分析阶段( 编译中的第一个阶段 )，这些标识符的位置信息就会以有序列表的形式保存到环境的 scope 属性中( 也就是作用域 )，以供引擎使用。也可以简单理解为，作用域里面保存的信息，在你写代码的时候已经决定了，而且会一直保持这个作用域不变。

例如：

var a = 1;

function afn() {

var b = a + 1;

var m = 999;

function bar(c) {

var n = a + b + c;

console.log(n);

}

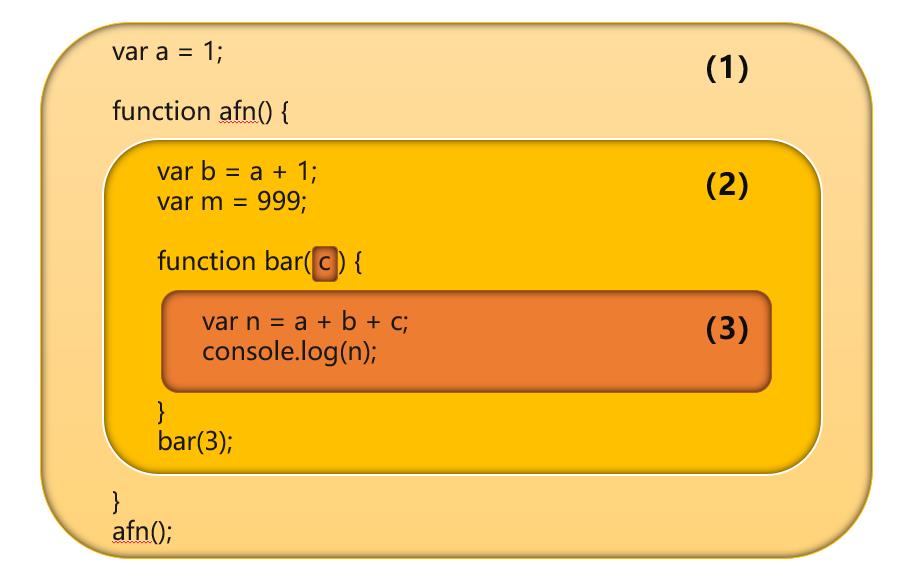
bar (3);

}

afn();

//输出：6

示意图：

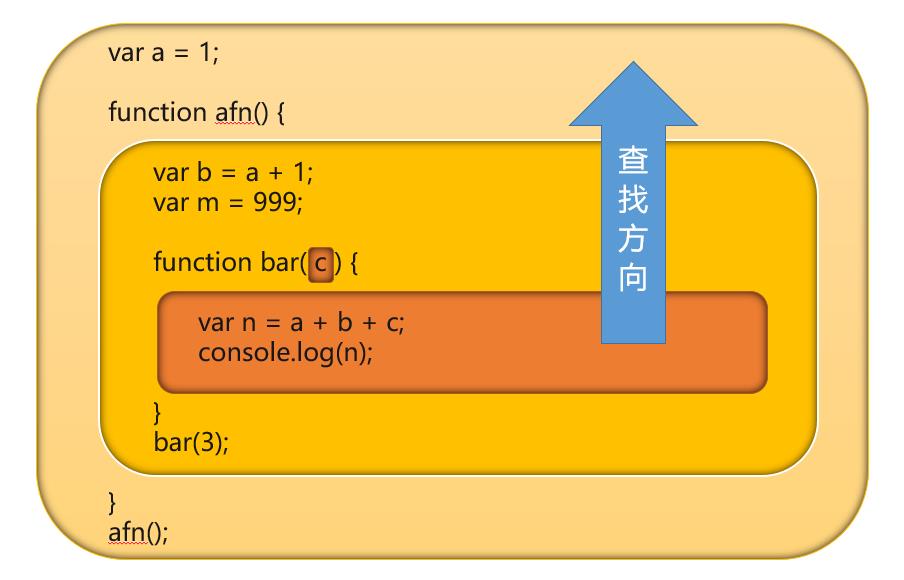


(1) 标识符 afn 和 a 位于全局作用域。  
(2) 标识符 b、m 和 bar 位于 afn 所创建的作用域。  
(3) 标识符 c 和 n 位于 bar 所创建的作用域。

**作用域的查找规则：**

1、查找标识符的过程会始终从当前作用域开始，然后逐级地向外层嵌套的作用域展开，直到找到标识符，或抵达最外层的作用域（也就是全局作用域）为止，如果找不到标识符，通常会导致错误发生。  
2、每个执行环境都可以进入到自身以外的外层作用域中查找标识符，但不能进入到自身以下内层作用域中查找标识符。

如上述例子的示意图：



**闭包概念：**

首先，闭包是一个函数；其次，也是最重要的一点，这个函数会一直保持对定义时所处作用域的引用。

在函数中定义函数是闭包最常见的形式，并且无论内部函数是在定义的作用域中被调用，还是在定义的作用域以外的地方被调用，属性 scope 中保存的信息都是一样的。如果内部函数作为闭包被返回，却不知道何时何地才会去执行，此时，可能它的父函数早就已经执行完毕，父函数中定义的变量或函数应该已经不复存在。但实际上，被返回的闭包函数通过将其父函数的作用域添加到它的作用域链中，使得当父函数执行完毕之后，由于子函数的作用域链依然在引用它的某些变量或函数阻止了这件事情的发生， 依然可以访问父函数中的变量或函数。

回调函数，实际上就是在使用闭包。

由于闭包会使一些数据无法被及时销毁，所以很多文章都会强调要尽量减少闭包的使用，其实我们在程序开发中，会选择主动把一些变量封闭在闭包中，主要考虑的是在不污染全局环境的前提下，以后还会需要使用这些变量。把这些变量放在闭包中和放在全局作用域，对内存方面的影响是一致的。  
从提高性能的角度考虑，一旦全局和闭包中的数据不再有用，最好通过将其值设置为 null 来释放其引用，以便垃圾收集器下次运行时将其回收

例如：

function fn() {

var name="Tom";

function childFn() {

console.log(name);

}

return childFn;

}

var person = fn();

person();

//输出：Tom

在 fn 执行后，我们通常会认为 fn 的作用域会被销毁，变量 name 也会被清除，但闭包阻止了这件事情的发生。之所以会这样，和 JavaScript 的垃圾回收机制有很大的关系：JavaScript 的垃圾收集器对于正处于环境中的标识符以及环境中的标识符引用的变量或函数 (包括变量或函数定义时的作用域)，不会进行清除，而是仍然留在内存中。

**一个经典的闭包问题：**

例子1：

<button>0</button>

<button>1</button>

<button>2</button>

var btns = document.querySelectorAll("button");

for (var i = 0; i < btns.length; i++) {

btns[i]`.onclick` = function() {

console.log(i);

}

}

//输出：3 3 3

在点击按钮以后，会发现三个按钮都会输出3.这是因为当点击按钮的时候，输出全局变量的i，在循环完成后i为3，所以都输出3.

例子2：

<button>0</button>

<button>1</button>

<button>2</button>

var btns = document.querySelectorAll("button");

for (var i = 0; i < btns.length; i++) {

(function(i) {

btns[i]`.onclick` = function() {

console.log(i);

}

})(i) //定义匿名函数马上调用语法

}

//输出：1，2，3

在每一个循环中，都会形成一个匿名函数并且马上调用，并且在该调用的匿名函数内部定义闭包。相当于：

(function() {

***var i = 0;***

btns[i]`.onclick` = function() {

console.log(i);

}

})()

(function() {

***var i = 1;***

btns[i]`.onclick` = function() {

console.log(i);

}

})()

(function() {

***var i = 2;***

btns[i]`.onclick` = function() {

console.log(i);

}

})()

当用户点击的时候，子函数会优先搜索到父函数中定义的变量 i，因此实现了返回各自的索引值。