执行环境及作用域

执行环境（execution context，为简单起见，有时也称为“环境”）是 JavaScript 中最为重要的一个概念。执行环境定义了变量或函数有权访问的其他数据，决定了它们各自的行为。每个执行环境都有一个与之关联的**变量对象**（variable object），环境中定义的所有变量和函数都保存在这个对象中。虽然我们编写的代码无法访问这个对象，但解析器在处理数据时会在后台使用它。

全局执行环境是最外围的一个执行环境。根据 ECMAScript 实现所在的宿主环境不同，表示执行环境的对象也不一样。在 Web 浏览器中，全局执行环境被认为是 window 对象（第 7 章将详细讨论），因此所有全局变量和函数都是作为 window 对象的属性和方法创建的。某个执行环境中的所有代码执行完毕后，该环境被销毁，保存在其中的所有变量和函数定义也随之销毁（全局执行环境直到应用程序退出——例如关闭网页或浏览器——时才会被销毁）。

每个函数都有自己的**执行环境**。当执行流进入一个函数时，函数的环境就会被推入一个环境栈中。而在函数执行之后，栈将其环境弹出，把控制权返回给之前的执行环境。ECMAScript 程序中的执行流正是由这个方便的机制控制着。

当代码在一个环境中执行时，会创建变量对象的一个**作用域链**（scope chain）。作用域链的用途，是保证对执行环境有权访问的所有变量和函数的有序访问。作用域链的前端，始终都是当前执行的代码所在环境的变量对象。如果这个环境是函数，则将其**活动对象**（activation object）作为变量对象。活动对象在最开始时只包含一个变量，即 arguments 对象（这个对象在全局环境中是不存在的）。作用域链中的下一个变量对象来自包含（外部）环境，而再下一个变量对象则来自下一个包含环境。这样，一直延续到全局执行环境；全局执行环境的变量对象始终都是作用域链中的最后一个对象。

标识符解析是沿着作用域链一级一级地搜索标识符的过程。搜索过程始终从作用域链的前端开始，然后逐级地向后回溯，直至找到标识符为止（如果找不到标识符，通常会导致错误发生）。

代码在一个环境中执行时，会创建变量对象的一个**作用域链**（scope chain）。作用域链的用途，是

保证对执行环境有权访问的所有变量和函数的有序访问。**作用域链的前端，始终都是当前执行的代码所在环境的变量对象**。如果这个环境是函数，则将其活动对象（activation object）作为变量对象。活动对象在最开始时只包含一个变量，即 arguments 对象（这个对象在全局环境中是不存在的）。**作用域链中的下一个变量对象来自包含（外部）环境，而再下一个变量对象则来自下一个包含环境。这样，一直延续到全局执行环境；全局执行环境的变量对象始终都是作用域链中的最后一个对象。**

标识符解析是沿着作用域链一级一级地搜索标识符的过程。搜索过程始终从作用域链的前端开始，然后逐级地向后回溯，直至找到标识符为止（如果找不到标识符，通常会导致错误发生）。

```
var color = "blue";

function changeColor(){
    var anotherColor = "red";

    function swapColors(){
        var tempColor = anotherColor;
        anotherColor = color;
        color = tempColor;

        // 这里可以访问 color、anotherColor 和 tempColor
    }

    // 这里可以访问 color 和 anotherColor，但不能访问 tempColor
    swapColors();
}

// 这里只能访问 color
changeColor();
```

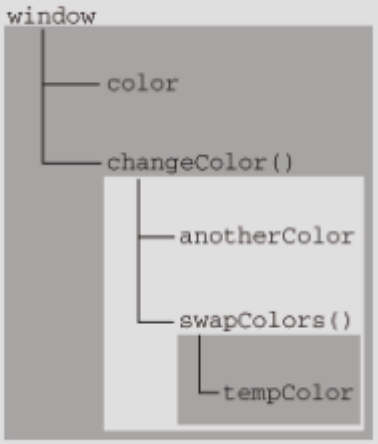

图 4-3

函数参数也被当作变量来对待，因此其访问规则与执行环境中的其他变量相同。