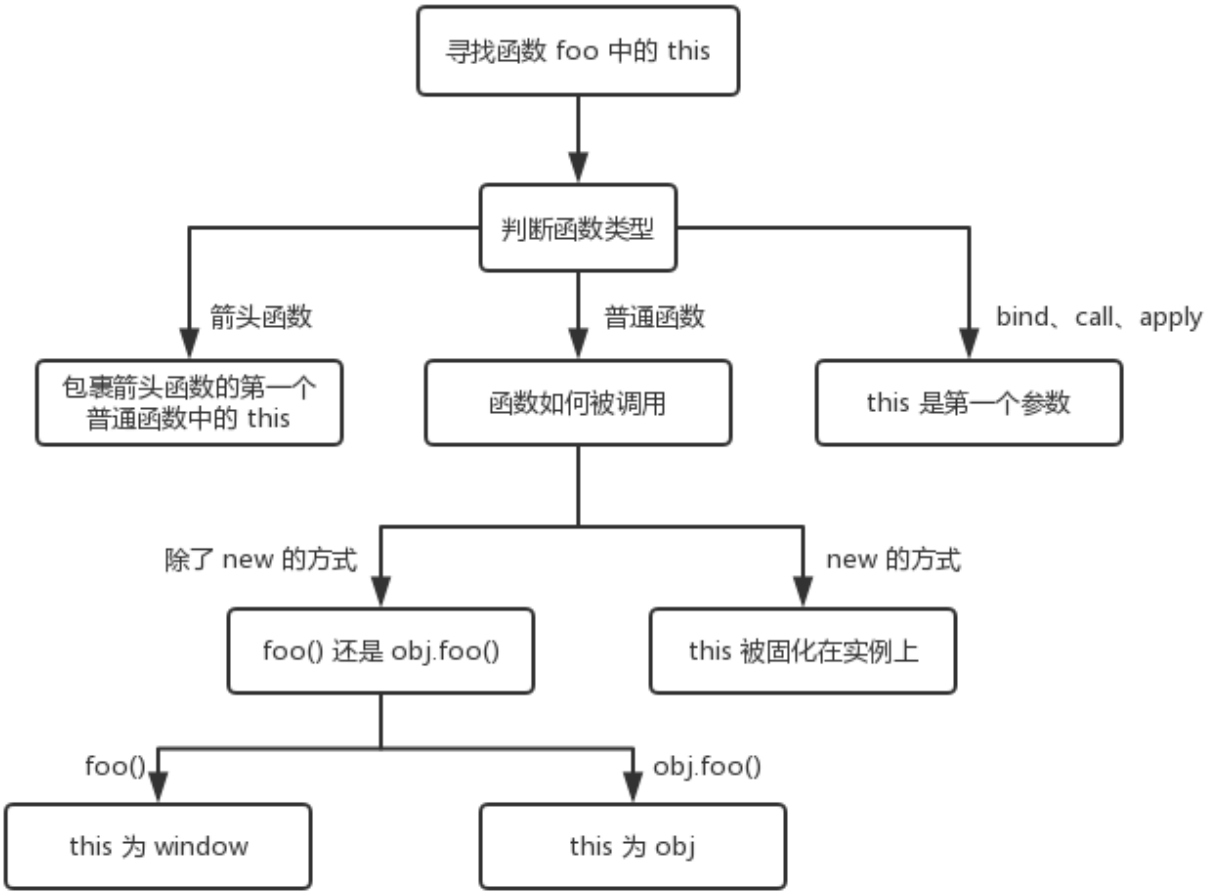
this：



作用域链和闭包

当某个函数被创建时，会创建一个执行环境以及相应的作用域链，然后使用arguments和其他命名参数来初始化函数的 活动对象 。

在作用域链中，外部函数的活动对象处于第二位，外部函数的外部函数的活动对象处于第三位.....直到作为作用域链终点的全局执行环境。

在函数执行过程中，为读取和写入变量的值，就需要在作用域链中查找变量。来看下面的例子。

```
function compare(value1, value2){
    if (value1 < value2){
        return -1;
    } else if (value1 > value2){
        return 1;
    } else {
        return 0;
    }
}

var result = compare(5, 10);
```

以上代码先定义了 compare()函数，然后又在全局作用域中调用了它。当第一次调用 compare()时，会创建一个包含 this、arguments、value1 和 value2 的活动对象。全局执行环境的变量对象（包含 this、result 和 compare）在 compare()执行环境的作用域链中则处于第二位。图 7-1 展示了包含上述关系的 compare()函数执行时的作用域链。

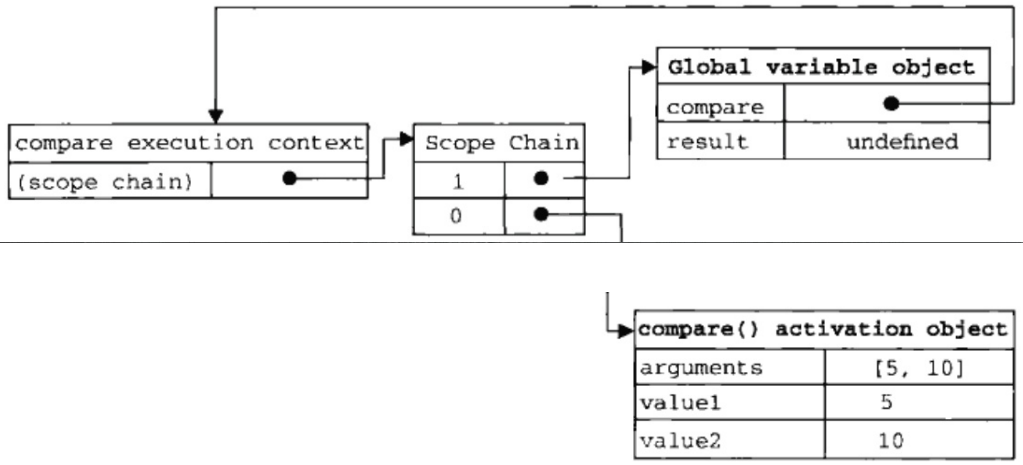


图 7-1

后台的每个执行环境都有一个表示变量的对象——变量对象。全局环境的变量对象始终存在，而像 compare()函数这样的局部环境的变量对象，则只在函数执行的过程中存在。在创建 compare()函数时，会创建一个预先包含全局变量对象的作用域链，这个作用域链被保存在内部的[[Scope]]属性中。当调用 compare()函数时，会为函数创建一个执行环境，然后通过复制函数的[[Scope]]属性中的对象构建起执行环境的作用域链。此后，又有一个活动对象（在此作为变量对象使用）被创建并被推入执行环境作用域链的前端。对于这个例子中 compare()函数的执行环境而言，其作用域链中包含两个变量对象：本地活动对象和全局变量对象。显然，作用域链本质上是一个指向变量对象的指针列表，它只引用但不实际包含变量对象。

无论什么时候在函数中访问一个变量时，就会从作用域链中搜索具有相应名字的变量。一般来讲，当函数执行完毕后，局部活动对象就会被销毁，内存中仅保存全局作用域（全局执行环境的变量对象）。但是，闭包的情况又有所不同。

闭包的定义：有权访问另一个函数作用域中的变量的函数

```
var createCompareNames = function(properName) {
    var temp = 5;
    return function (obj1, obj2) {
        return obj1[properName] === obj2[properName];
    }
}

var compare = createCompareNames('name');
```

在调用完createCompareNames之后，createCompareNames的作用域链被销毁，但是其活动对象没有被销毁，这是闭包的本质，且闭包保存的是整个活动对象，因此，其实var temp也还存留在内存中

经典面试题：