JavaScript 中有两个机制可以“欺骗”词法作用域：eval(..) 和 with。

前者可以对一段包含一个或多个声明的“代码”字符串进行演算，并借此来修改已经存在的词法作用域（在 运行时）。后者本质上是通过将一个对象的引用当作作用域来处理，将对象的属性当作作用域中的标识符来处理，从而创建了一个新的词法作用域（同样是在运行时）。

**先有蛋（声明）再有鸡（赋值）**

函数声明和变量声明都会被提升。但是一个值得注意的细节（这个细节可以出现在有多个

“重复”声明的代码中）是函数会首先被提升，然后才是变量。

foo();

bar()

var foo = function bar() {

// ...

};

||

||

var foo

foo = function (){

var bar = …self…

// ...

}

解析：

①var foo; 声明foo变量但是未赋值

②foo() 由于对 undefined 值进行函数调用而导致非法操作，因此抛出 TypeError 异常。不是 ReferenceError, 而是 TypeError

③bar() 报错ReferenceError

我们习惯将 var a = 2; 看作一个声明，而实际上 JavaScript 引擎并不这么认为。它将 var a

和 a = 2 当作两个单独的声明，第一个是编译阶段的任务，而第二个则是执行阶段的任务。

这意味着无论作用域中的声明出现在什么地方，都将在代码本身被执行前首先进行处理。

可以将这个过程形象地想象成所有的声明（变量和函数）都会被“移动”到各自作用域的

最顶端，这个过程被称为提升。

声明本身会被提升，而包括函数表达式的赋值在内的赋值操作并不会提升。

要注意避免重复声明，特别是当普通的 var 声明和函数声明混合在一起的时候，否则会引

起很多危险的问题！

**闭包**

function foo() {

var a = 2;

function bar() {

console.log( a );

}

return bar;

}

var baz = foo();

baz(); // 2 ——闭包的效果。