## 理解变量与函数的声明提升

Js支持词法作用域，除了极少数的例外，对变量foo的引用会被绑定到声明foo变量最近的作用域中。但是js却不支持块级作用域，只有函数作用域和全局作用域。变量的作用域并不是离的最近的代码块，而是包含他们的函数，或者是全局。

理解js变量声明行为的一个好办法是把变量声明看作是由两部分组成的。一部分是变量的声明，一部分是赋值行为。因为事实上，js也就是这么做的，他先将同一个作用域中的所有变量都声明一遍，然后在该赋值的地方赋值。

function myFun(){

var a = 5;

for(var i = 0; i < a ; i++){

var play = "a"+ i;

// do something...

}

}

上面这段代码其实很简单，一共声明了三个变量a、i、play，这三个变量都是在myFun的作用域下。事实上js的变量提升运作相当于以下这样：

function myFun(){

var a, i, play;

//将下面的变量提升到上面，在下面该赋值的地方赋值

a = 5;

for(i = 0; i < a ; i++){

play = "a"+ i;

// do something...

}

}

当然，在变量提升也可能导致变量重复声明的混淆，在同一函数，或者是相同作用域中（ES5），多次声明相同的变量是合法的，这在我们在一个作用域中写多个for循环，但是都用的i=0的情形中经常出现。

在编写js代码的时候，一些程序员总是喜欢通过手动提升变量和函数，把之后要用到的函数和变量，都声明在作用域最顶端。从而避免歧义，而且还易于理解。这是一个好的习惯。

Js没有块级作用域的一个例外恰好是异常处理，这个坑之前在用vue写项目的时候踩过。

Try...catch语句将捕获的异常绑定到一个变量上，该变量的作用域只是catch语句块。

说白了catch是一个块级作用域。和全局作用于和函数作用域是差不多的。

注意，try的部分依旧还是上一级的作用域，并不是一个块级作用域！

在catch的()中的x属于catch作用域中的内容。和其他的函数不同。其他函数中的参数是外部作用域环境中的。

function myFun(){

var x = "var",result = [];

result.push(x);

try{

var x = "1"

throw "强制出现了一个错误" // 手动抛出错误，错误信息就是err

} catch(err) {

// x = "catch";

}

result.push(x);

return result;

}

*console*.log(myFun()); // ["var","1"]

看看下面的代码：

function myFun(){

var x = "var",result = [];

result.push(x);

try{

throw "强制出现了一个错误" // 手动抛出错误，错误信息就是err

} catch(x) {

x = "catch";

}

result.push(x);

return result;

}

*console*.log(myFun()); // ["var","var"]

如果不明白变量提升和函数提升的话很容易不明白下面的这个是错在哪里：

var x = document.getElementById("x1").innerHTML;

function x(){

alert(x);

}

x();

这个的执行结果就是报错。



但是这样就不会报错。

function x(){

var x = document.getElementById("x1").innerHTML;

alert(x);

}

x();

弹出了弹出框，上面写着“我是方向明”。

分析错误原因，其实上面的代码经历了函数提升和变量提升后会是这样的：

// 1、函数名也相当与变量名，会出现冲突。

// 2、会出现变量提升和函数提升，且函数会比变量先提升，并且不会被变量声明覆盖

// 3、变量赋值操作会覆盖函数提升

function x(){

alert();

}

var x; // 此时这个声明并没有什么用处，无法覆盖前面名为x的函数

x = document.getElementById("x1").innerHTML; // 执行了赋值操作，覆盖了函数，x成为了一个字符串

x(); // 执行字符串会出现错误

*console*.log(x); // 注释掉上一行，打印出来看看结果，很明显就是字符串。

在《你不知道的javascript》（上卷）中明确提到过：函数会首先被提升，然后才是变量。

**函数提升优先级比变量提升要高，且不会被变量声明覆盖，但是会被变量赋值覆盖。**

还需要值得注意的是，函数在执行时会经历词法分析的过程，这个过程并没有对函数体中的程序依次执行，可以理解为只是把参数、函数、变量单独挑出来分析一手，这也正是变量提升与函数提升的原理：

首先需要知道的是，函数在执行的时候会瞬间形成一个活动对象（Active Object），简称AO。

**第一步：先分析参数**

参数肯定有形参和实参，但是函数在执行的时候并不是立即马上把实参调过来使用，而是经过了一个分析形参的过程：

1. 先分析形参，如果形参存在，比如名为age。那么当场形成一个新的AO属性AO.age。并且自动赋值undefined；
2. 之后分析实参，如果对应位置的实参存在，那么把对应位置的形参所对应的那个AO属性改为实参的值。（值不是引用）。

**第二步：分析变量的声明**

之后就是分析变量的声明，因为函数提升比变量提升优先提升，所以先分析变量的声明，之后在分析函数的声明，这样才能让后者覆盖前者，做到函数优先提升。

1. 如果变量声明存在且不和形参同名，则新生成一个AO属性，属性名就是变量名。并赋值undefined。
2. **如果变量声明存在且和形参同名，则不做任何操作。对应的AO属性依旧取形参的值。**

**第三步：分析函数声明**

1. 如果存在函数声明，且不存在相同名的AO属性。则新生成一个AO属性，属性名就是函数名，且直接将AO属性的值索引到对应函数。（注意，这里没有一个先undefined的过程）
2. 如果存在函数声明，且存在同名的AO属性，则直接覆盖之前的AO属性，将之前的AO属性的值变为这个函数的索引。

所以总结来说就是，如果函数体内存在函数声明，那么函数声明总是最后的结果，如果存在和形参同名变量声明，但是却没有赋值，那么这个声明将不会带来任何影响，依旧是参数的值，并不会把同名参数的值变为undefined（不懂变量提升或者不清楚词法分析的常常会误解）。

**另外值得注意的是，不会对函数表达式进行函数提升，它属于变量提升。**

容易然萌新误解的函数：

(function(*age*){

var age; // 并不会有任何覆盖操作

*console*.log(age);

})(18);

结果为：



参考博客 ：<https://www.cnblogs.com/zingp/p/6102561.html>

参考博客 ：http://www.cnblogs.com/pqjwyn/p/5365532.html