**下面我们来详细解答一下基础是什么。**

**let** 知识, 基础

**if** (知识 **===** '房子') {

基础 **=** '地基'

}

**if** (知识 **===** '大树') {

基础 **=** '树根'

}

**if** (知识 **===** '天空') {

基础 **=** '阶梯'

console.log('基础 makes you up, up, up…… 直到你碰头')

}

………………

怎么样？认识到基础的重要性了吧，如果没有了基础，代码就好像无根之木，空中楼阁，虽然赏心悦目，但总是短暂的。

直到你真正领悟了底层是怎么运作的，你才能够真正做到 **他强任他强，清风拂山岗；他横自他横，明月照大江。**

扯远了，收~

回到我们的正题。今天带大家了解以下 JavaScript 中的作用域问题。请拭拭拭拭拭拭目以待。（自己 get 重点）。

先从最基础的开始讲起。

**1.什么是作用域**

作用域是什么这个问题，好多人都回答不好。**请注意：通常来说，作用域就是限制一个变量在程序中的使用范围。**

搜嘎，突然有一种 “同行十二年，不知木兰是女郎” 的赶脚。

**1.1 全局和局部**

了解了作用域的名字来由之后。我们来认识一下它。

在 JavaScript 中作用域的边界是以函数划分。有 **全局** 和 **局部** 作用域之分。

* **全局作用域**：声明在全局的变量或者**不使用var声明的变量**在整个程序中都是可用的，所以叫**全局作用域**。
* **局部作用域**：声明在**函数体内**的变量，在整个函数执行环境和其子函数内都是可用的，但是在函数外访问不到，所以叫**局部作用域**

小栗子 同学上场：

**<!DOCTYPE html>**

<html lang**=**"en">

<head>

<meta charset**=**"UTF-8">

<title>Title</title>

</head>

<body>

</body>

</html>

<script>

**var** global **=** "我是全局变量，全局都能看到我";

global2 **=** "我也是全局变量，全局都能看到我";

**function** getName() {

**var** name **=** "我是局部变量，只能在getName函数内才能找到我"

}

</script>

**1.2 预解析和变量提升**

看到标题的同学是不是会稍有一愣。预解析是什么玩意儿？变量提升又是啥？（知道答案的同学请配合这个无聊的作者一下，假装一愣神。）

咳咳~不忙，等本大神（经）来解释一下。

**预解析**是在程序执行之前，会进行一遍预检。查找当前**作用域**内由 function 和 var 。并且每次更换作用域都会在此作用域中执行**预解析**

**变量提升**是指，在查找到由 function 和 var 后，首先在当前作用域的顶端定义好并赋给默认值。var的默认值为 undefined， function的默认值为**函数本身**。

注：

​ 像 var getName = function() {} 这种代码会被当做变量定义，而不会当做函数定义。

“让一下，让一下……” 远处小栗子携大量代码滚滚而来~~

**<!DOCTYPE html>**

<html lang**=**"en">

<head>

<meta charset**=**"UTF-8">

<title>Title</title>

</head>

<body>

</body>

</html>

<script>

console.log(global) *// undefined*

console.log(getName) *// function getName(){}*

**var** global **=** 1;

**function** getName () {}

</script>

诶？怎么这样？我不是定义了global吗？怎么会输出 undefined 难道是javascript出现了bug？

不要多想，预解析和变量提升的过程中，并不会将变量赋值，而只是定义，等真正执行的时候才会赋值。修改一下代码，在global变量下方再次打印。

**var** global **=** 1;

console.log(global); *// 1*

解释：当执行到打印函数的时候，global已经被赋值为 1。此时已经在**执行代码的阶段**，而不是在预解析阶段。

**先练练手：**

猜想一下，下方程序如何输出。

console.log(a) *// 1*

**var** a **=** 1;

console.log(a) *// 2*

getName()

**function** getName () {

console.log(a) *// 3*

console.log(b) *// 4*

a **=** 2;

console.log(a) *// 5*

**var** b **=** 3;

console.log(b) *// 6*

**function** b(){}

}

此时你的答案是什么呢？

针对上方代码，通过图解的方式看下**预解析**的执行过程。

**ps:**（上图画的太复杂，看完需要耐心）

**1.3 var和function的优先级**

细心地同学可能会发现一个问题，上面的代码，在getName函数中，我既定义了 **b变量**，也定义了 **b函数**。为什么在 console.log(b) // 4 的时候会输出 undefined？

因为在预解析的过程中，会先查找 function 然后再查找 var 所以，function 会被 var 覆盖。这里我们会理解为 **在预解析过程中，function的优先级高于var。高优先级的会被低优先级的覆盖** （是不是很绕？没关系，多想想，加深下理解）

**2.作用域链**

作用域链的执行我们在之前就讲过了，有没有人注意到？有没有？

好吧，没有人回答我，看来是没人注意到了。

就在练手代码中，getName()函数内，向上查找 a 变量的过程，那个就是作用域链的查找过程。

闲言少叙，上高清大图：

特别注意：**作用域只能从下向上查找，不可逆向。**（从函数外不能访问函数内的变量）

**3. 从全局获取函数内部的变量**

上面讲到，在函数外访问函数内的变量是访问不到的，如果我坚持要访问呢？*（一般情况下，这种钻牛角尖的人都容易挨打）*。

好吧，既然你要访问，那也是有方法的，我们可以在函数内将变量返回出来，这样就可以访问到函数内的变量了。

多说无益，还是代码最实在：

console.log(getA()); *// 通过这种方式，我们就可以访问到 a 变量。*

**function** getA() {

**var** a **=** 1;

**return** a;

}

发散一下思维，你还知道其他方式吗？

**4. 块级作用域**

在ES6到来的时候，javascript迎来一个全新的概念，-- **块级作用域**。顾名思义，块级作用域可以让变量只在一块代码内生效。

举个栗子：

{

**var** a **=** 1

}

console.log(a)

上述栗子中的代码会正常输出，但是下方的代码会抛出 a 变量未定义的错误a is not defined。

{

**let** a **=** 1;

}

console.log(a)

也就是说a变量只在花括号内生效，在花括号外是访问不到的。

可以声明块级作用域的方式有两种。let 和 const 。

目前为止，我们看到了三个定义变量的方式，接下来，让我们瞅瞅他们之间的不同。

**4.1 var、let、const的异同**

**同：** 都可以声明变量。

异：

**var 存在局部作用域，可变量提升，声明的值可更改。**

console.log(a)

**var** a **=** 1;

a **=** 2;

*// 上述操作都可以*

**let 存在块级作用域，不可变量提升，声明的值可修改。（只可以先声明变量，然后再使用）**

console.log(a) *// 会报错， a is not defined*

**let** a **=** 1;

a **=** 2;

**const 存在块级作用域，不可变量提升，声明的值本身不可修改（只可以先声明变量，然后再使用）**

**const** a **=** 1;

a **=** 2; *// 会报错，a不可修改*

*// 下述情况可运行*

**const** a **=** []

a[0] **=** 1;

注意：

* const声明的叫做常量，不可以修改其本身，但如果声明的是复杂类型的对象，对象里的值是可修改的。

这里你会发现一个问题，三者的功能是逐步增强的。

**4.2 TDZ介绍（暂时性死区）为什么let和const不能变量提升。**

使用let和const声明的变量，在**预解析**的时候会将变量放入到一个暂时不可访问的区间中，此时访问变量会提示未定义错误，在给变量赋值后，将变量放入到正常的执行环境中。使变量可以正常访问。

console.log(a) *// 此时的a在TDZ中*

**let** a **=** 1; *// 将a从TDZ中移出来*

console.log(a) *// 此时可以正常访问a变量*

相信你现在已经充满能量，打开你的代码，学会分析每一步的执行顺序吧。