昵称： 甘劭
园龄： 1年5个月
粉丝： 6
关注： 8
+加关注

< 2019年12月 >

日	一	二	三	四	五	六
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	1	2	3	4
5	6	7	8	9	10	11

搜索

找找看

谷歌搜索

常用链接

我的随笔

我的评论

我的参与

最新评论

我的标签

积分与排名

积分 - 24460

排名 - 29459

随笔分类

ES6(4)

java ee 基础(8)

java se 基础(29)

java框架(3)

js - 高级应用(10)

mysql数据库(5)

vue.js(15)

js-作用域-变量申明提升

作用域

域，表示的是一个范围，作用域，就是作用范围。

作用域说明的是一个变量可以在什么地方被使用，什么地方不能被使用。

块级作用域

JavaScript中没有块级作用域

```
{
    var num = 123;
    {
        console.log( num );
    }
}
console.log( num );
```

上面这段代码在JavaScript中是不会报错的，但是在其他的编程语言中（C#、C、JAVA）会报错。

这是因为，在JavaScript中没有块级作用域，使用{}标记出来的代码块中声明的变量num，是可以被{}外面访问到的。

但是在其他的编程语言中，有块级作用域，那么{}中声明的变量num，是不能在代码块外部访问的，所以报错。

词法作用域

什么是词法作用域?

词法(代码)作用域, 就是代码在编写过程中体现出来的作用范围. 代码一旦写好, 不用执行, 作用范围就已经确定好了. 这个就是所谓词法作用域.

在 js 中词法作用域规则:

- 函数允许访问函数外的数据.
- 整个代码结构中只有函数可以限定作用域.
- 作用域规则首先使用提升规则分析
- 如果当前作用规则中有名字了, 就不考虑外面的名字

JavaScript 预解析

JavaScript引擎在对JavaScript代码进行解释执行之前，会对JavaScript代码进行预解析，在预解析阶段，会将以关键字var和function开头的语句块提前进行处理。

关键问题是怎么处理呢?

当变量和函数的声明处在作用域比较靠后的位置的时候，变量和函数的声明会被提升到作用域的开头。

重新来看上面的那段代码

```
func();
function func(){
     alert("Funciton has been called");
}
```

由于JavaScript的预解析机制，上面的代码就等效于:

```
function func(){
    alert("Funciton has been called");
```

面试题

数据结构与算法之美(1)

随笔档案

2019年6月(3)

2019年3月(6)

2019年2月(9)

2019年1月(10)

2018年12月(18)

2018年11月(5)

2018年9月(2)

2018年8月(23)

2018年7月(2)

阅读排行榜

1. vue-路由精讲 - 二级路由和三级路由(10615)

2. list集合排序的两种方法(4858)

3. Java异或详解(4237)

4. 递归思想及几个经典题目(4153)

5. vue -- 路由精讲制作导航 -- 从无到有(2290)

推荐排行榜

1. vue-路由精讲 - 二级路由和三级路由(1)

```
}
func();
```

看完函数声明的提升，再来看一个变量声明提升的例子：

```
alert(a);
var a = 1;
```

由于JavaScript的预解析机制，上面这段代码，alert出来的值是undefined，如果没有预解析，代码应该会直接报错a is not defined，而不是输出值。

Wait a minute, 不是说要提前的吗？那不是应该alert出来1，为什么是undefined?

那么在这里有必要说一下声明、定义、初始化的区别。其实这几个概念是C系语言的人应该都比较了解的。

行为	说明
声明	告诉编译器/解析器有这个变量存在,这个行为是不分配内存空间的,在JavaScript中，声明一个变量的操作为：var a;
定义	为变量分配内存空间，在C语言中，一般声明就包含了定义，比如：int a;,但是在JavaScript中，var a;这种形式就只是声明了。
初始化	在定义变量之后，系统为变量分配的空间内存储的值是不确定的，所以需要对这个空间进行初始化，以确保程序的安全性和确定性
赋值	赋值就是变量在分配空间之后的某个时间里，对变量的值进行的刷新操作（修改存储空间内的数据）

所以我们说的提升，是声明的提升。

那么再回过头看，上面的代码就等效于：

```
var a; //这里是声明
alert(a);//变量声明之后并未有初始化和赋值操作，所以这里是 undefined
a = 1;
```

复杂点的情况分析

通过上一小节的内容，我们对变量、函数声明提升已经有了一个最基本的理解。那么接下来，我们就来分析一些略复杂的情况。

函数同名

观察下面这段代码：

```
func1();
function func1(){
    console.log('This is func1');
}

func1();
function func1(){
    console.log('This is last func1');
}
```

输出结果为：

```
This is last func1
This is last func1
```

原因分析：由于预解析机制，func1的声明会被提升，提升之后的代码为：

```
function func1(){
    console.log('This is func1');
}

function func1(){
    console.log('This is last func1');
}
```

```
func1();
func1();
```

同名的函数，后面的会覆盖前面的，所以两次输出结果都是This is last func1。

变量和函数同名

```
alert(foo);
function foo(){}
var foo = 2;
```

当出现变量声明和函数同名的时候，只会对函数声明进行提升，变量会被忽略。所以上面的代码的输出结果为

```
function foo(){}
```

我们还是来吧预解析之后的代码展现出来：

```
function foo(){};
alert(foo);
foo = 2;
```

再来看一种

```
var num = 1;
function num () {
     alert( num );
}
num();
```

代码执行结果为：

```
Uncaught TypeError: num is not a function
```

直接上预解析后的代码：

```
function num(){
     alert(num);
}

num = 1;
num();
```

预解析是分作用域的

声明提升并不是将所有的声明都提升到window对象下面，提升原则是提升到变量运行的环境(作用域)中去。

```
function showMsg()
{
    var msg = 'This is message';
}
alert(msg); // msg未定义
```

还是直接把预解析之后的代码写出来：

```
function showMsg()
{
    var msg;
    msg = 'This is message';
}
alert(msg); // msg未定义
```

预解析是分段的

分段，其实就分script标签的

```
<script>
func(); // 输出 AA2;
function func(){
    console.log('AA1');
}

function func(){
    console.log('AA2');
}
```

```
</script>

<script>
function func(){
    console.log('AA3');
}
</script>
```

在上面代码中，第一个script标签中的两个func进行了提升，第二个func覆盖了第一个func，但是第二个script标签中的func并没有覆盖上面的第二个func。所以说预解析是分段的。

tip:但是要注意，分段只是单纯的针对函数，变量并不会分段预解析。

函数表达式并不会被提升

```
func();
var func = function(){
    alert("我被提升了");
};
```

这里会直接报错，func is not a function，原因就是函数表达式，并不会被提升。只是简单地当做变量声明进行了处理，如下：

```
var func;
func();
func = function(){
    alert("我被提升了");
}
```

条件式函数声明

```
console.log(typeof func);
if(true){
    function(){
        return 1;
    }
}
console.log(typeof func);
```

上面这段代码，就是所谓的条件式函数声明，这段代码在Gecko引擎中打印"undefined"、"function"；而在其他浏览器中则打印"function"、"function"。

原因在于Gecko加入了ECMAScript以外的一个feature：条件式函数声明。

> Conditionally created functions Functions can be conditionally declared, that is, a function declaration can be nested within an if statement.
>
> Note: Although this kind of function looks like a function declaration, it is actually an expression (or statement), since it is nested within another statement. See differences between function declarations and function expressions.

Note中的文字说明，条件式函数声明的处理和函数表达式的处理方式一样，所以条件式函数声明没有声明提升的特性。

作用域链

什么是作用域链

只有函数可以制造作用域结构， 那么只要是代码，就至少有一个作用域，即全局作用域。

凡是代码中有函数，那么这个函数就构成另一个作用域。如果函数中还有函数，那么在这个作用域中就又可以诞生一个作用域。

将这样的所有的作用域列出来，可以有一个结构：函数内指向函数外的链式结构。就称作作用域链。

例如：

```
function f1() {
    function f2() {
    }
}
```

```
var num = 456;
function f3() {
    function f4() {
    }
}
```

绘制作用域链的步骤:

1. 看整个全局是一条链, 即顶级链, 记为 0 级链
2. 看全局作用域中, 有什么成员声明, 就以方格的形式绘制到 0 级练上
3. 再找函数, 只有函数可以限制作用域, 因此从函数中引入新链, 标记为 1 级链
4. 然后在每一个 1 级链中再次往复刚才的行为

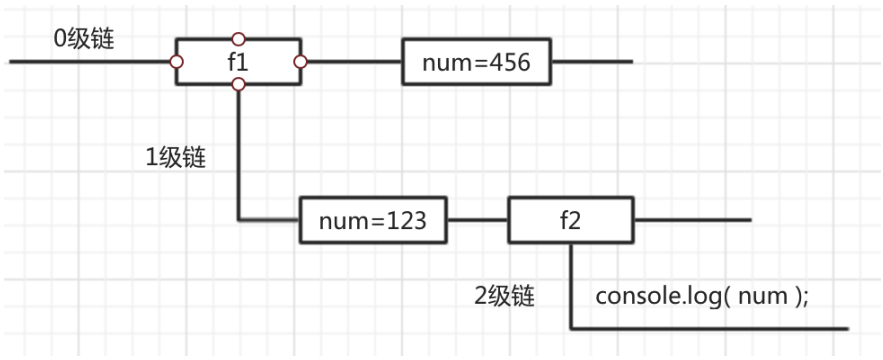
变量的访问规则

- 首先看变量在第几条链上, 在该链上看是否有变量的定义与赋值, 如果有直接使用
- 如果没有到上一级链上找(n - 1 级链), 如果有直接用, 停止继续查找.
- 如果还没有再次往上刚找... 直到全局链(0 级), 还没有就是 is not defined
- 注意,同级的链不可混合查找
    - 变量的搜索原则:
      1.在使用变量的时候
      * 首先在所在的作用域中查找
      * 如果找到了 就直接使用
      * 如果没有找到 就去上级作用域中查找
      2.重复以上步骤
      * 如果直到0级作用域链也就是全局作用域还没有找到, 报错

练习: 绘制作用域链

```
function f1() {
    var num = 123;
    function f2() {
        console.log( num );
    }
    f2();
}

var num = 456;
f1();
```



如何分析代码

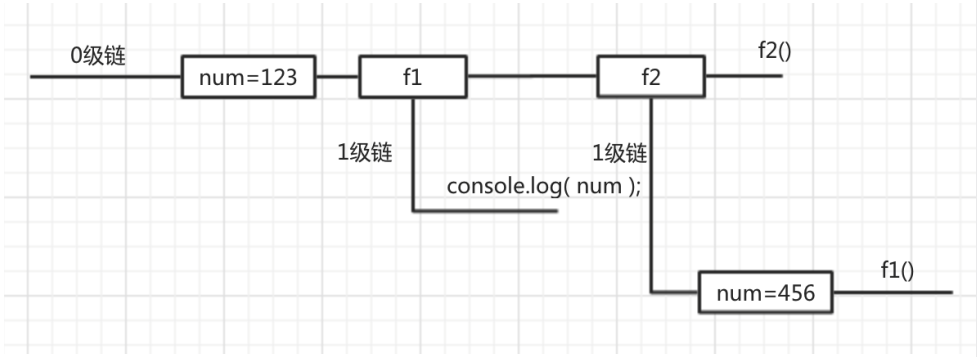
1. 在分析代码的时候切记从代码的运行进度上来分析, 如果代码给变量赋值了, 一定要标记到图中
2. 如果代码比较复杂, 可以在图中描述代码的内容, 有事甚至需要将原型图与作用域图合并分析

练习

```
var num = 123;
function f1() {
```

```
    console.log( num );
}

function f2() {
    var num = 456;
    f1();
}
f2();
```



补充

声明变量使用`var`，如果不使用`var`声明的变量就是全局变量(禁用)

因为在任何代码结构中都可以使用该语法. 那么再代码维护的时候会有问题. 所以除非特殊原因不要这么用.

下面的代码的错误

```
function foo () {
    var i1 = 1 // 局部
    i2 = 2, // 全局
    i3 = 3; // 全局
}
```

此时注意

```
var arr = [];
for ( var i = 0; i < 10; i++ ) {
    arr.push( i );
}

for ( var i = 0; i < 10; i++ ) {
    console.log( arr[ i ] );
}

// 一般都是将变量的声明全部放到开始的位置, 避免出现因为提升而造成的错误
var arr = [],
i = 0;

for ( ; i < 10; i++ ) {
    arr.push( i );
}

for ( i = 0; i < 10; i++ ) {
    console.log( arr[ i ] );
}
```

补充:

函数加载和变量

函数加载问题

JS加载的时候，只加载函数名，不加载函数体。所以如果想使用内部的成员数。

//变量问题：根据作用范围，变量可以分为局部变量和全局变量。

局部变量：只有局部能够访问的变量。

函数内部用var定义的变量。

全局变量（成员变量）：在哪里都能访问到的变量。

函数外部或者进入javascript之后立即定义的变量和函数内部不带有var的变量

```
var num3 = 333;

//函数加载的时候，只加载函数名，不加载函数体。
function fn(){
  //局部变量
  var num1 = 111;
  //全局变量（成员变量）
  num2 = 222
  console.log(num3);
}
fn();
//  console.log(num1);
console.log(num2);
console.log(num3);
```

5. 隐式全局变量

隐式全局变量就是隐藏的全局变量不好被发现。

```
function  fn（）{
var  a  =  b  =  c  =  1;   // b和c就是隐式全局变量
}
```

注意：

```
function  fn（）{
var  a  =  b  =  c  =  1;   // b和c就是隐式全局变量（等号）
var  a = 1;  b = 2;  c = 3;     // b和c就是隐式全局变量（分号）
var  a = 1 ,  b = 2 ,  c = 3;    // b和c就不是隐式全局变量（逗号）
}
```

6. 变量声明提升（出现原因：预解析）

预解析：js的解析器在页面加载的时候，首先检查页面上的语法错误。然后把变量值提升变量名，不提升变量值。而用function直接定义的方法是整体提升。

1.查看语法错误。

2.变量声明提升和函数整体提升(变量声明提升的时候，只提升变量名，不提升

3.函数范围内，照样适用。

```
var aaa;
console.log(aaa);
aaa = 111;  //结果为 undefined


fn();
function fn(bbb){
  //变量声明提升在函数内部照样实用。
  //函数的就近原则。
  var aaa;
  console.log(aaa);
  aaa = 222; // 结果为 undefined
}
function fn2(bbb){
  //两个函数中的局部变量不会相互影响。
  console.log(bbb);
```

```
}            // 结果为 undefined
```

```
var num = 10;
 fun();
 function fun(){
      console.log(num);
      num = 20;
 }          //结果为 10, 因为 num = 20 为全局变量，不是局部变量，不存在
```

没写一样的效果。

★

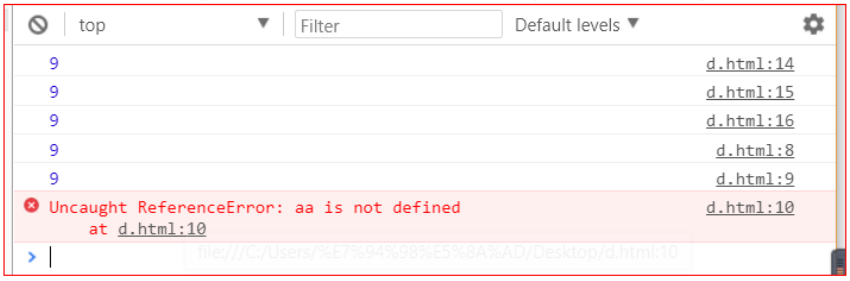
```
var num = 10;
 fun();
 function fun(){
     //变量声明提升  只提升变量名，不提升变量值var num;
     console.log(num);
     var num = 20;
 }          // 结果为 undefined
```

★

```
var a = 18;
  f1();
  function f1(){
     var b=9;
     console.log(a);
     console.log(b);
     var a = '123';
  }       // 结果为 undefined   9
```

```
6     <script type="text/javascript">
7     f2();
8     console.log(cc);
9     console.log(bb);
10    console.log(aa);
11
12    function f2(){
13        var aa = bb = cc = 9;
14        console.log(aa);
15        console.log(bb);
16        console.log(cc);
17    }
18
19    </script>
```

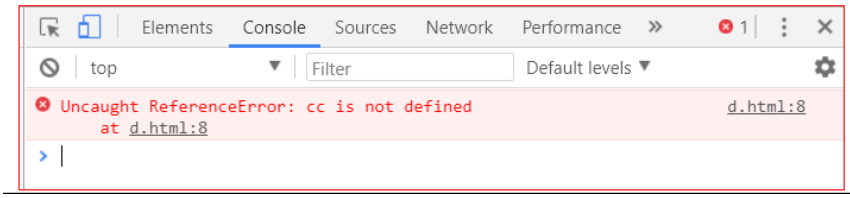
代码解析：先执行f2()函数，从而获得bb, cc 为隐式全局变量，
而aa为局部变量，故执行第十行代码的时候，aa没有定义，导致报错。




```
6     <script type="text/javascript">
7
8     console.log(cc);
9     console.log(bb);
10    console.log(aa);
11    f2();
12
13    function f2(){
14        var aa = bb = cc = 9;
```

```
16          console.log(bb);
17          console.log(cc);
18      }
19
20      </script>
```

代码解析：执行第8行代码的时候，由于cc没有定义，直接报错，后面的代码不



分类: js - 高级应用

甘劭
关注 - 8
粉丝 - 6
+加关注

0 0

« 上一篇: 递归思想及几个经典题目
» 下一篇: js-深入浅出之闭包

posted @ 2018-08-30 16:34 甘劭 阅读(149) 评论(0) 编辑 收藏

刷新评论 刷新页面 返回顶部

发表评论

昵称: 苏卡达

评论内容:

提交评论 退出 订阅评论

[Ctrl+Enter快捷键提交]