

一、变量提升

在全局作用域下的js代码自上而下执行之前，浏览器首先会把当前作用域（window、私有作用域）中所有带var和function关键字的，进行提前的声明或者定义，这种提前处理的机制就是变量提升（预解释）

- 带var关键字的只是提前的声明（declare）
- 带function关键字的，声明和定义（defined）都提前完成了

只声明未定义，默认值undefined

注意：

在“全局作用域”下声明的变量，相当于给全局对象window增加了一个属性（私有作用域不具备这个特点的）

代码块

```
console.log(res); //undefined: 预解释阶段，只是声明了一个变量，但是没有赋值，默认值就是undefined
console.log(sum); //当前函数本身：预解释阶段，函数的声明和定义都完成了，此时的函数是有值的 sum代表的是函数本身 sum()把函数执行，代表的是执行后的返回结果
sum(); // ->可以在这个部分执行，因为已经定义完成了
var res = null;
function sum() {
    var total = null;
    for (var i = 0; i < arguments.length; i++) {
        var cur = Number(arguments[i]);
        if (isNaN(cur)) continue;
        total += cur;
    }
    return total;
}
res = sum(12, 24, 34);
console.log(res); // -> 70
```

浏览器加载页面的时候，会提供一个供js代码运行的环境 => “全局作用域”（window/global）

预解释: var res; sum = xxxfff000

代码执行:
res = null;

再次遇到函数创建代码的时候，直接跳过

res = sum(...): ->先把sum函数执行，然后把函数执行的结果赋值给res(变量)
= 69

xxxxfff000

var total = null;
for (var i = 0; i < arguments.length; i++) {
 var cur = Number(arguments[i]);
 if (isNaN(cur)) continue;
 total += cur;
}
return total;
字符串

sum(12, 23, 34):

形参赋值

预解释: var total; var cur;

total = null;

return total: ->69

二、作用域

1、函数执行的步骤：

- 形成一个新的私有作用域，供函数体中的代码执行
- 给形参赋值
- 私有作用域下的预解释
- 私有作用域中的js代码自上而下执行
- 作用域释放或者不释放

代码块

全局下的变量提升（预解释）

```
//var a; fn = xxxfff000; //在全局作用域下

console.log(a);// ->undefined ->只声明未定义
var a = 12;
console.log(a);// ->12
function fn() {
  /*
  * fn执行形成一个私有作用域
  * 形参赋值:
  * 变量提升: var a; ->说明 a 是当前私有作用域的私有变量，和外面没有任何关系
  * */
  console.log(a);// ->undefined
  var a = 13;
  console.log(a);// ->13
  console.log(window.a);// ->12 指定获取外面的 a
}
a = fn();// ->先把函数fn执行，把执行的返回值赋值给全局变量 a（看一个函数的返回值，只需要看函数体中是否有return，有return，return后面是啥函数的返回值就是啥，没有return，默认返回值是undefined）
console.log(a);//undefined
```

```

/*
 * 变量提升:
 *   var a; <=> window.a=undefined;
 * */
console.log(a); // -> undefined
var a = 12; // <=> window.a=12; 既是一个变量，也是window的一个属性
console.log(a); // -> 12 输出变量存储的值
console.log(window.a); // -> 12 输出win属性a的属性值
console.log(b); // -> ReferenceError: b is not defined 当前这一行js代码
报错，后面的代码都不再执行了
b = 13; // -> b不是变量，它是window的一个属性名（前提：在全局作用域下操作） 就
是window.b=13;
console.log(b); // -> 13 window.b
console.log(window.b); // -> 13

```

```

/* fn1(); //
function fn1() {
console.log(1);
} */
// -> 匿名函数：函数表达式（把函数当作一个值，赋值给一个变量或者某个元素的某个事件）
// document.body.onclick = function () {};
/*
 * 变量提升:
 *   只对 “=” 左边的进行变量提升，右边的是值，不进行提升的操作
 *   var fn; -> undefined
 * */
fn(); // -> Uncaught TypeError: fn is not a function 相当于undefined(), undefined不是函数，所以报错
var fn = function () { // 真正项目中使用该方法偏多（保证先创建再执行） -> 严谨
    console.log(1);
};
fn();

```

```

/*
* 不管条件是否成立都要进行预解释（if 和 for在ES5的机制下是不存在作用域概念的，ES6的机制下是块级作用域）
* ->不管是带var还是带function，判断体中出现的内容，在变量提升阶段都仅仅是声明，没有定义
* 变量提升：
*     var a; function fn;
*
* */
console.log(a); // ->undefined
console.log(fn); // ->undefined 老版本浏览器输出的是fn，新版本做了一些处理，结果是undefined
/*if (1 === 1) { //1!=1（条件成立）
    console.log(a); // ->undefined
    console.log(fn); // ->函数体 function fn..... 条件成立，进入到判断体中的第一件事就是先把函数定义了
    var a = 12;

    function fn() {
        console.log(1);
    }
}*/
if (1!=1) { //1!=1（条件不成立）
    console.log(a); // ->undefined
    console.log(fn); // ->undefined
    var a = 12;

    function fn() {
        console.log(1);
    }
}
console.log(a);
console.log(fn); // ->条件不成立，进入不到判断体中，函数还是无法赋值

```

```

/*
 * 分析步骤：
 *   全局作用域下的变量提升：没有
 *
 * */
f = function () { return true;}; // ->window.f ->true
g = function () { return false;}; // ->window.g ->false
// ->自执行函数：函数创建和函数执行都完成了（自执行函数不参与变量提升）
(function () { // ->在前面加一个分号是为了防止上面的函数结束不加分号，导致括
号的执行顺序出现错误

    /*
     * 形成一个私有作用域
     * 形参赋值：
     * 变量提升：function g; ->undefined
     *
     * */
    if (g() && [] == ![]) { // ->g() 相当于undefined(): Uncaught Type
Error: g is not a function
        f = function f() {
            return false;
        };
        function g() {
            return true;
        }
    }
})();
console.log(f());
console.log(g());

```

思考题

```

/*
 * 思考题：
 * 在变量提升阶段，如果当前这个名字已经声明过了，则不再重新的声明，只需要重新
的定义赋值即可（变量名和函数名如果相同也算是重复）
 * 变量提升：
 *      fn = xxxfff111
 *      = xxxfff222
 *      = xxxfff333
 *      = xxxfff444 =>4
 * */
fn();//4
function fn() { console.log(1);}
fn();//4
function fn() { console.log(2);}
fn();//4
var fn = 13; // ->13
fn();// ->13();Uncaught TypeError: fn is not a function
function fn() { console.log(3);}
fn();//
function fn() { console.log(4);}
fn();//

```

面试题

```

var a = 100;
function testResult() {
    var b = 2 * a; // 2*undefined =>NaN
    var a = 200;
    var c = a / 2; // 100
    alert(b); // "NaN"
    alert(c); // "100"
}
testResult();

```

2、作用域 [scope]

全局作用域：浏览器加载页面就形成的

私有作用域：函数执行形成的私有作用域

不管是全局的还是私有的作用域，都是给js代码提供运行环境的，我们把作用域称之为“**栈内存**”

所有的引用数据类型在定义的时候，都会开辟一个新的内存空间，来存储自己的值，我们把存储东西的内存空间称之为“**堆内存**”

- 对象在自己的堆内存中存储的是“**键值对**”
- 函数在自己的堆内存中存储的是“**代码字符串**”

代码块

```
console.log(a, b, c); // undefined
var a = 12, b = 13, c = 14;
function fn(a) {
  console.log(a, b, c); // a:100 b:undefined c:14
  a = 30;
  var b = 10;
  c = 20;
  console.log(a, b, c); // a:30 b:10 c:20
}
fn(100);
console.log(a, b, c); // a:12 b:13 c:20
```

window 栈内存

变量提升: var a; var b; var c; fn=xxxxfff000

```
=>a,b,c => undefined*3
a=12;
b=13;
c=14; 20
```

=>a,b,c 12 13 20

堆内存: xxxfff000

```
"
console.log(a, b, c);
a = 30;
var b = 10;
c = 20;
console.log(a, b, c);
"
```

fn(100)

私有作用域 (栈内存)

形参赋值: a=100

变量提升: var b;

```
=>a,b,c
100, undefined, 14
```

c 在当前的私有作用域中不存在，不是私有变量，向上级作用域 (window) 查找，找到的 c 都是全局变量

```
a = 30;
b = 10;
c = 20; 把全局下的 c 修改为 20
=>a,b,c 30 10 20
```

全局变量和私有变量

私有变量：

- 函数的形参是私有作用域中的私有变量
- 在私有作用域中声明过的变量或者函数都是私有的

全局变量：

- 除了私有变量其它都是全局变量
- 在全局作用域中声明的变量肯定是全局变量

函数执行形成一个私有的作用域（栈内存）

1、形参赋值

2、变量提升

形参或者是声明过的变量是私有的变量，否则就不是私有的

私有的变量受到了保护，和外界变量没有关系，不会受到任何的干扰，我们把函数执行的这种保护机制叫做 - “闭包”

在私有作用域中执行代码的时候，如果当前这个变量不是私有的变量，则向上一级查找，也不是上一级私有的，则继续向上查找，一直找到window...，我们把这种向上一层层查找的机制叫做-“作用域链”

如何查找当前作用域的上级作用域

看当前函数是在哪里定义的，那么它执行的上级作用域就是谁，和它在哪儿执行没有关系

代码块

```
//形参 arguments return
var a = 12;
var obj = {
  a:14,
  fn: (function () {
    var a = 13;
    return function () {
      console.log(a);
    }
  })() // -> 自执行函数是在给fn这个属性名赋值的时候就执行了，把执行的返回结果赋值给fn了
};
obj.fn();
```



