内存空间管理

09:18 2020年03月09日, 星期一

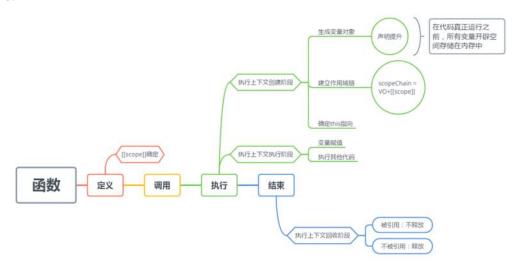
1.内存空间管理

```
var obj = { name: 'Tom' };
                                  分配内存
console.log(obj.name);
                                  使用分配到的内存
obj = null;
                                  不需要时释放内存
```

2.垃圾回收机制

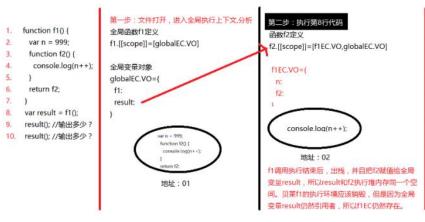
- 当一个值失去引用之后就会回收 (堆内存失去引用时,就会被回收)
- 一般的,当一个函数的执行上下文运行完毕之后,内部的所有内容就会失去引用,被垃圾回收机制回收
- 但是,一个函数的执行上下文运行完毕之后,内部的内容**仍然被引用**着,就不会被回收

3.



4.分析代码运行过程

```
function f1() {
   var \ n = 999:
    function f2() {
       console. log(n++):
    return f2;
var result = f1();
result(); //輸出多少? 999
result(); //輸出多少? 1000
```



第三步:执行第9行代码 执行result就等价于执行f2函数 f2EC.VO={ 执行f2, console.log(n++), 先在自己的VO中找,没找 到,去[[scope]]中的f1EC.VO 中找,找到了n。值为999, n++,先输出,再自加。加的 是f1EC.VO中的n.值由999变 成了1000。 第9行执行结束后,f2EC出栈并且销毁

第四步:执行第10行代码 此时又创建一个新的f2的执行环境

f2EC.VO={

执行f2.console.log(n++). 白己的变量对象中没有找到。去 f1EC.VO中,找到n,并且n的值为 1000.输出1000 , 再自加 , f1EC.VO 中的n=1001。

第10行执行结束后,f2EC出栈并且销毁。

5.

```
var fun1, fun2;
function foo() {
    var x = 10;
    fun1 = function() {
        console.log(++x);
    }
    fun2 = function() {
        console.log(--x);
    }
}
```

```
foo();
fun1(); // ??
fun2(); // ??
```

```
var fun1, fun2;
1.
       function foo() {
2.
        var x = 10;
3.
        fun1 = function() {
4.
         console.log(++x);
5.
6.
        fun2 = function() {
7.
        console.log(--x);
8.
        };
9.
      }
10.
       foo();
11.
       fun1(); // ?? 11
12.
13.
       fun1(); // ?? 12
       fun2(); // ?? 11
14.
```