javascript构造函数用大写

function Person(name,age){ this.name = name;this.age = age; }

var person = new Person('xx',18);

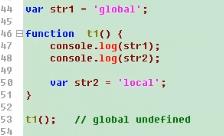
javascript 的作用域，寻找变量一般都是从内往外找. js自上而下执行,执行之前先有一个“词法分析过程”.

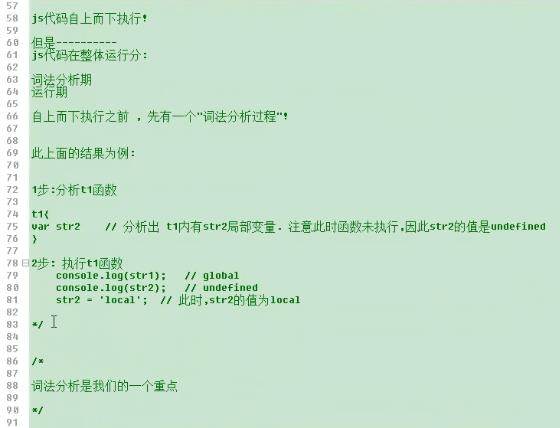
js代码在整体运行分：词法分析，运行期;

|  |
| --- |
| function t1(){      var d;      function t2(){          d = 5;          e = 6;      }      t2();  }  t1();  console.log(e); //6  console.log(d); //d is not defined错误  console.log(window.d); //undefined |

注意：以window.xxx引用全局变量,寻找不到，做为某个属性不存在,返回undefined

直接以xxx引用某命题,寻找不到，则是报xxx is not defined 错误.

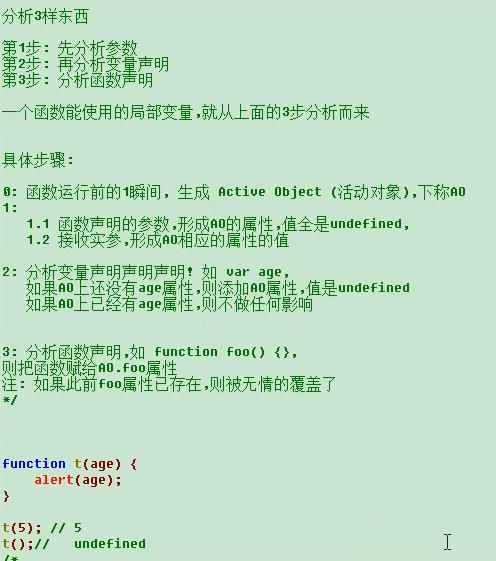




语法分析：

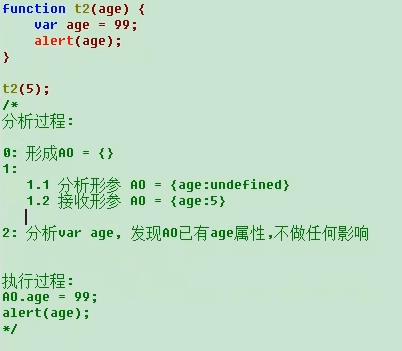
第1步：先分析参数；2：再分析变量声明；3：分析函数声明;

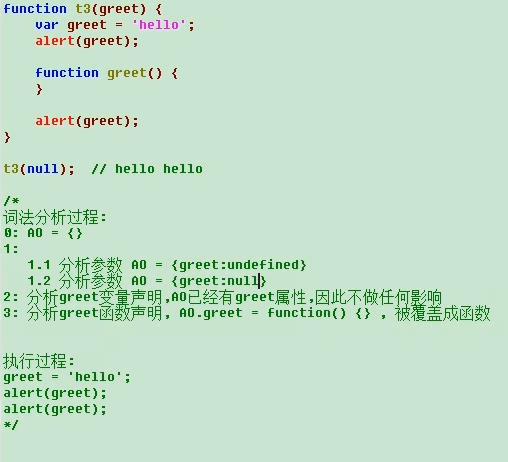
一个函数能使用的局部变量,就从上面的3步分析而来.





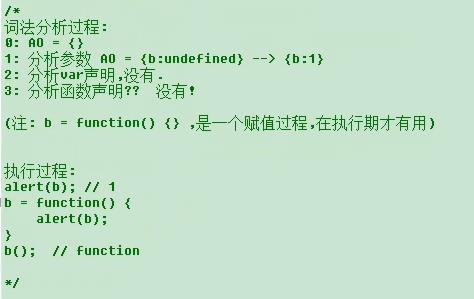






t3(null)输出hello  hello





函数可以赋值给变量，可以作为参数来传递;

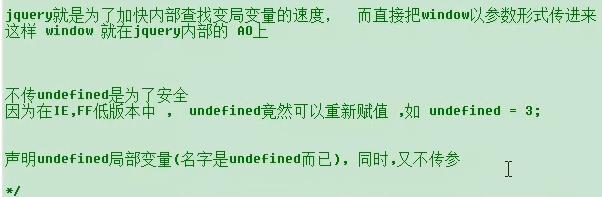
function t1(){}

t2=function(){}

//这2种方式，效果不同，t1是函数声明，虽然全局内也得到一个t1变量,值是function.

而t2只是一个赋值过程----值是谁？值是右侧的表达式，返回一个结果。就是说function(){}在js看来就和3\*2,6/3一样，是个表达式，返回一个结果。因此,t1,t2两种方式在词法分析时，有着本质的区别,前者在词法分析阶段，就发挥作用。而后者，在运行阶段，才发挥作用。





arguments.callee 属性代表“当前运行的函数”,也是指arguments属于哪个函数. callee一般用在递归计算中.

alert((function (n){

  if(n <= 1){

        return 1;

  }else {

       return n + arguments.callee(n-1);

}

})(100));

函数运行期内,关键的三个对象:

AO-->本函数AO上没有某属性,则继续去外层函数的AO上找,直到全局对象，叫做 作用域链

argument -->每个函数有自己的callee，但不向外层接着找arguments的相关属性,即不形成链.

this-->作为对象的方法来调用,this指向方法的调用者，即该对象; 一般函数的调用指向window.

 函数作为构造函数调用时，js中没有类的概念，创建对象是用构造函数来完成，或者直接用json格式{}来写对象.





call为对象冒充;

作用域链

面向对象原型链

这里简单总结下有关作用域链的问题，首先要求我们得懂的js的词法分析，关于词法分析分为两部分来思考，分别是分析阶段和执行阶段。

**分析阶段**

在分析阶段，我们要懂得依次分析三样东西，他们分别是：

先分析所传递参数

再分析变量声明

最后分析函数声明

一个函数能使用的**局部变量**，就是从上面三步分析得来的，而对于**全局变量**，我们就得沿着作用域链自下而上寻找了。

**具体步骤：**（*注意：步骤是依次执行的*）

函数运行前的一瞬间，会生成Active Object(活动对象)，简称AO。

分析传递参数

函数声明的参数，形成AO的属性，值全是undefined

接收实参，形成AO相应属性的值

分析var变量声明，如：var age

如果AO上还没有age属性，则值是undefined

如果AO上已经存在age属性，则不做任何影响

分析函数声明，如 function foo () {}

则把函数赋给AO.foo属性

如果此前foo属性已经存在，则被无情的覆盖

**执行阶段**

最后分析执行过程，比如：变量的赋值操作、打印输出等

**注意：** 函数表达式也是一个赋值过程，在此阶段执行，特此注意！

下面简单分析几个案例：

案例一：

**function** test1 (age) {

**var** age = 20;

alert(age);

}

test1();

1、分析过程：

形成 AO={}

分析形参，AO={age:undefined}

分析var age,发现AO已有age属性，不做任何影响

2、执行过程

AO.age=20

alert(age)

案例二：

**function** test2 (a) {

**var** a = 'hello';

alert(a);

**function** a () {}

alert(a);

}

test2();

1、分析过程：

AO={}

分析参数，AO={a:undefined}

分析变量声明,AO已有a属性，因次不做任何影响。

分析函数声明，AO.a=function(){},即a属性被覆盖成函数

2、执行过程

a=hello

alert(a);

alert(a);

案例三：

**function** a (b) {

alert(b);

**function** b () {

alert(b);

}

b();

}

a(1)；

1、分析过程

形成AO={}

分析参数，AO={b:undefined},接受实参，形成AO={b:1}

分析变量声明，无var变量声明

分析函数声明，AO.b=function (){}

2、执行过程

alert(b)

b(),然后alert(b)