js连续赋值及js引用类型指针

JavaScript中的连续赋值：

```
var a = {n: 1}
var b = a;
a.x = a = {n: 2}
console.log(a.x); //undefined
console.log(b.x) //Object {n: 2}
```

昨天看到这个面试题，再看看结果，死活理解不了，今天终于搞明白了，现在分享一下心得

先多看几个例子

```
var a = {n: 1}
var b = a;
a = a.x = {n: 2}
console.log(a.x); //undefined
console.log(b.x) //Object {n: 2}
```

```
var a = {x:{xx:1},y:2,z:3};
var b = a.x;  //{xx:1}
var c = a;
a.w = a.x.xx = a.y = a = {x:10,y:20};
console.log(a); // a:{x: 10, y: 20}
console.log(b); // b:{xx : {x: 10, y: 20}}
console.log(c); // c:{x:{xx:{x:10,y:20}},y:{x:10,y:20},z:3,w:{x:10,y:20}}

console.log(c.x.xx.x);//10
console.log(c.y.x);//10
console.log(c.w.x);//10
```

试试看你能不能理解以上的执行结果，如果可以就不用往下看了

我们先回头看第一个例子：

```
var a = {n:1};
var b = a;
a.x = a = {n:2};
console.log(a.x);// --> undefined
console.log(b.x);// --> [object Object]
```


其实这道题看似简单但还是有一些绕，我依稀记得高中数学老师那句经典的口头禅！
遇到难题：画图啊！

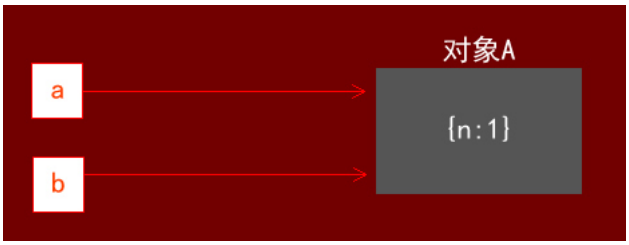
好吧，这句话可能我会受用一辈子，同时也送给看这篇文章的同学，希望能给你们编[illegible]一些新的思路。

下面来分析下这段简单代码的工作步骤，从而进一步理解js引用类型"赋值"的工作方式

首先是

```
var a = {n:1};
var b = a;
```

在这里a指向了一个对象{n:1}（我们姑且称它为对象A），b指向了a所指向的对象，也就是说，在这时候a和b都是指向对象A的：



这一步很好理解，接着继续看下一行非常重要的代码：

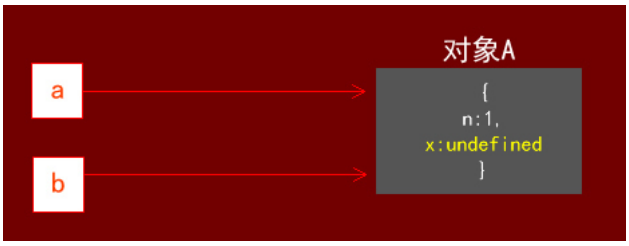
```
a.x = a = {n: 2};
```

这句话也是关键所在

画图

根据js引擎语法解析，会先去从左到右寻找有没有未声明的变量，如果有就把该变量提升至作用域顶部并声明该变量。那么恭喜js引擎他找到a.x这个属性没有声明，{n: 1}这个内存区声明一个x属性等待赋值！

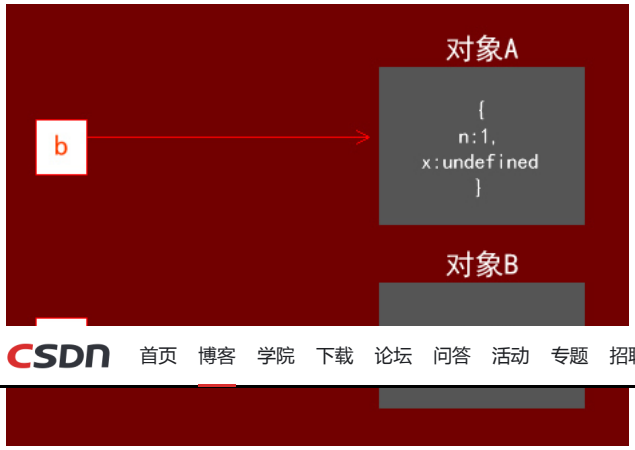
如下图：



从图上可以看到，由于b跟a一样是指向对象A的，要表示A的x属性除了用a.x，自然也可以使用b.x来表示了。

语法解析完成后，开始进行运算(ps：赋值运算),首先依循“从右往左”的赋值运算顺序先执行 a={n:2} ，这时候，将a变量的指针指向了一个新的内存区{n: 2}, (B），那么a变量脱离了对内存区{n: 1}的引用关系。

```
a.x = a = {n: 2};
```



但是此时{n:1}这个内存区并没有被GC回收因为b变量的指针依然指向它。并且因为之前就[illegible]x属性所以该内存区增加了X属性。那么X属性指向哪儿呢？

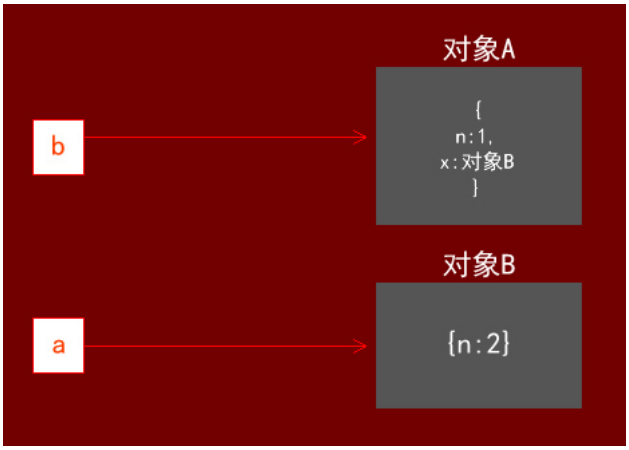
```
a.x = a = {n: 2};
```

接着继续执行 a.x=a，很多人会认为这里是“对象B也新增了一个属性x，并指向对象B

但实际上并非如此，由于一开始js已经在对象A中生成了x:undefined属性，则原来的[illegible]示上是A.x，

a.x = a = {n:2}; 但是由于赋值运算从右向左运算，所以a.x 在这个式子中 最终被指向了 {n:2} ，也就是对象B

即A.x指向 对象B




```
var a = {n: 1}
var b = a;
a.x = a = {n: 2}
console.log(a.x); //undefined
console.log(b.x) //Object {n: 2}
```

那么这时候结果就显而易见了。当**console.log(a.x)**的时候，a是指向对象B的，但对象B没有属性x。没关系，当查找一个对象的属性时，JavaScript 会
型链，直到找到给定名称的属性为止。但当查找到达原型链的顶部 - 也就是 Object.prototype - 仍然没有找到指定的属性B.prototype.x，自然也就输出u

而在**console.log(b.x)**的时候，由于b.x表示对象A的x属性，该属性是指向对象B，自然也输出了[object Object]了，注意这里的[object Object]可不是2
思，对象的字符串形式，是隐式调用了Object对象的toString()方法，形式是："[object Object]"。所以[object Object]表示的就只是一个对象罢了:)

本文章根据自己的理解，以及根据前辈留下的博客，自己加以理解和编辑梳理所得。

文章参考：

Night_Emperor https://blog.csdn.net/night_emperor/article/details/78509456

caozheng550 https://segmentfault.com/a/1190000008475665

举报

点赞 1 收藏 分享 ...