正课:

1. 闭包笔试题:

2. 面向对象

1. 闭包笔试题:

先判断是不是考闭包: 3个特点:

外层函数，局部变量，内层函数

如何:

1. 找两样东西:

1. 妈妈生了几个孩子(外层函数共向外抛出了几个内层函数):

可能通过return

也可能通过给全局变量赋值

2. 包了几个红包（外层函数有几个局部变量））

2. 规律： 凡是从一次妈妈调用中返回的多个孩子，共用同一个红包！任何一个函数修改红包，另一个函数一定受影响。

比如:

function fun(){

var n=999;

nAdd=function(){ n++ };

return function(){

console.log(n)

}

}

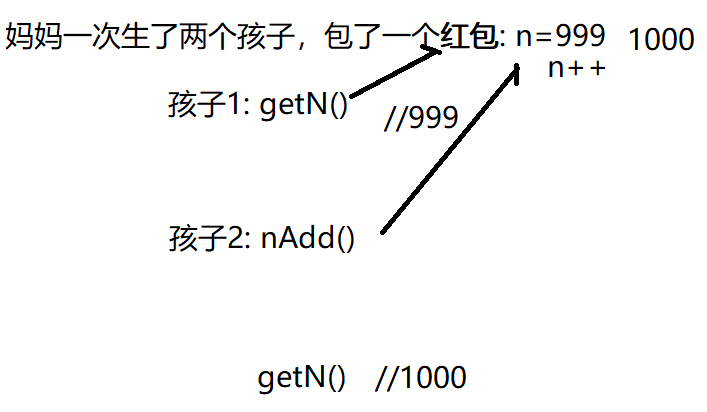
var getN=fun();

getN();//999

nAdd();

getN();//1000

画简图:



2. 面向对象:

什么是对象: 2个概念:

用途: 程序中描述现实中一个具体实物的属性和功能的程序结构

本质: 内存中一个块集中存储一个实物的多个属性和功能的存储空间。

什么是面向对象: 将来的程序都是先用对象描述出不同的事物的属性和功能，然后再按需访问对象中的属性和功能。

为什么: 便于大量数据的使用和维护

何时: 今后所有程序都是用面向对象。

如何: 面向对象三大特点: 封装，继承，多态

1. 封装:

什么是: 将一个事物的属性和功能集中定义在一个对象结构中——其实就创建一个对象

为什么: 都是为了便于大量数据的管理和使用

何时: 今后只要使用面向对象的方式编程，第一步都是创建对象。然后才是按需使用对象中的属性和功能。  
 如何:3种:

1. 用{}：

var 对象名={

属性名: 属性值,

... : ...,

功能/方法: function(){ ... },

... : ...

}

何时: 在创建对象时，已经知道对象中将要包含什么内容。

比如:

var lilei={

sname: "Li Lei",

sage: 11,

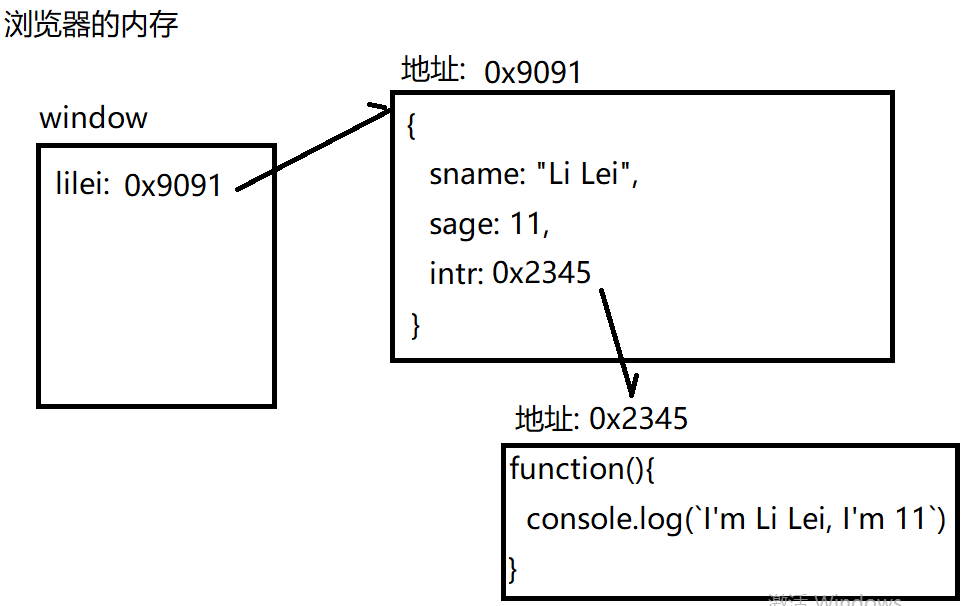
intr: function(){

console.log(`I'm Li Lei, I'm 11`)

}

}

在内存中：



问题: 对象的属性值发生变化，而方法中的内容不会跟着变化。

原因: 方法中的输出内容是写死的

错误的解决方案: 将写死的值，改为属性名，在运行时动态获得属性值。

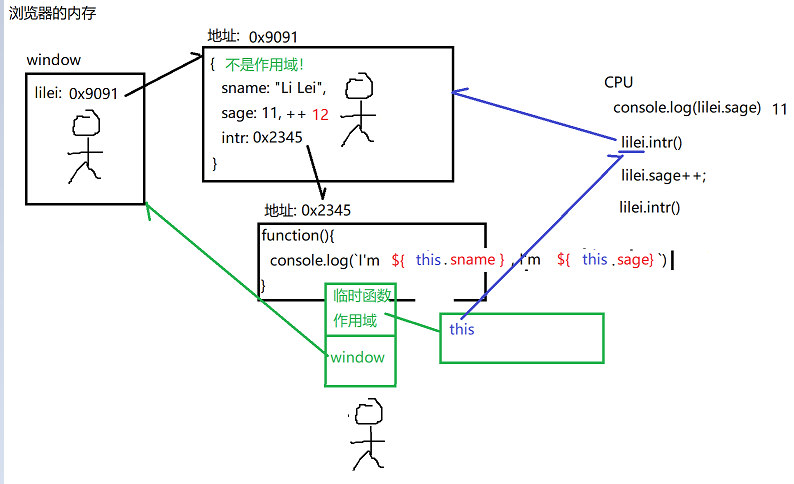
原因: 不带任何前缀的变量，默认只能在自己的函数作用域和window之间找。无权在对象里找。因为所有对象都不是作用域。

不好的解决方案: 在属性名前加固定的对象名。

问题: 对象名其实就是一个变量名，很可能被修改。一旦对象名修改，方法内忘记修改，立刻就报错！——紧耦合

最优的解决方案: 使用this关键字

什么是: 每个函数作用域中自动添加的，自动引用正在调用函数的.前的那个对象的 关键词



何时: 今后只要对象自己的方法，想使用自己的属性，也必须加this.前缀。

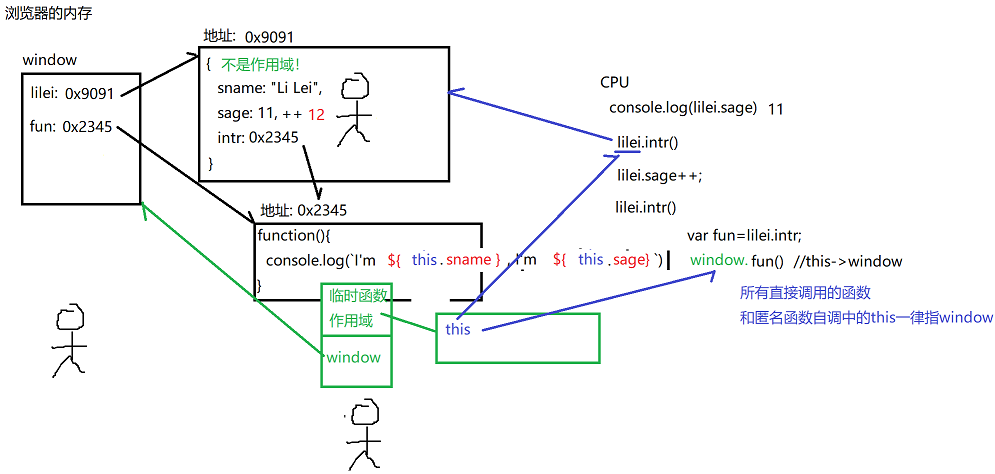
鄙视时: 判断this一定不要看定义在哪儿。必须看调用时，.前是谁！

比如: var fun=lilei.intr;

lilei将自己的intr函数的地址值给了全局变量fun。从此fun和lilei.intr共同引用同一个函数.

fun();

结果输出I'm undefined, I'm undefined。因为fun()调用时，前边没有. 。如果没有.的一切都会被自动加上window.， 所以，fun()调用时this->window。因为window中没有sname和sage属性，所以，返回undefined



访问对象的成员: 成员=属性+方法

访问属性: 属性其实就是保存在对象内部的变量而已。用法和变量完全一样。只不过查找时，要先找到对象，再用.运算符找到对象内的指定属性。

lilei.sname

的

调用方法: 方法其实就是保存在对象内部的函数而已。用法和函数完全一样。只不过查找时，要先找到对象，再用.运算符找到对象内的指定方法，然后加()调用方法。

lilei.intr()

2. 用new来创建: 2步:

1. 先用new 创建空对象:

var obj=new Object();

new可省略, ()可省略

不能同时省略

其实就可以简写为: var obj={};

2. 向空对象中添加成员:

js中想向一个对象中添加新成员，只有一个野蛮的办法，就是强行赋值！

obj.新属性=值

obj.新方法=function(){ ... }

何时: 如果创建对象时，暂时不知道对象中的成员具体内容时。

揭示: 所有对象底层其实都是关联数组！

1. 都是名值对儿的集合。

2. 都可用下标或.的方式访问每个成员

翻译

比如: ym["math"]<---ym.math

lilei["sname"] lilei.sname

如果访问成员时:

成员名是写死的，即可用.，又可用["成员名"]

成员名不是固定的，可能来自于变量或拼接，则必须用[ 变量/表达式 ]，且不要加""

3. 都可用for in遍历:

for(var key in 对象/关联数组){

key自动获得当前属性名

数组[key] 获得当前属性值

对象[key] 获得当前属性值

}

强调: [key]一定不要加引号，因为key是变量，从来没有将变量加在""中的写法！一个变量如果加上""，就不能再变化了！

4. 访问数组或对象中不存在的位置都不会报错！而是返回undefined

5. 强行给数组或对象中不存在的位置赋值，不会报错！而是自动添加该成员。

前两种方式，一次只能创建一个对象。如果反复创建多个相同结构的对象时，代码会很繁琐！

3. 用构造函数来创建:

什么是: 2个概念

1. 专门描述一类对象的统一结构的函数

2. 专门用于创建同一类型多个对象的特殊函数

何时: 只要反复创建多个相同结构的对象时，都要用构造函数来创建

如何: 2步:

1. 定义构造函数，其中描述同一类型所有对象的统一结构。不需要指定具体属性值。

function 类型名(形参列表){

this.新属性1=形参;

this.新属性2=形参;

this. ... = ...;

this.新方法=function(){

... ...

}

}

2. 调用构造函数，创建该类型的一具体个对象实例。

var obj=new 类型名(属性值列表)

↑

构造函数名

new的原理: new干了四件事:

1. 新建一个空对象

2. 自动设置当前对象继承构造函数的老公——原型对象爸爸

3. 用新对象去调用构造函数。会自动将构造函数中所有this都指向新对象。然后，再用强行赋值的方式，给新对象添加新属性

4. 返回新对象的地址给变量保存起来。

构造函数的问题: 虽然重用了结构代码，但是反复调用了function，创建了多个方法的副本。浪费了内存。

解决:

2. 继承:

什么是: 父对象的成员，子对象无需重复创建，就可直接使用！

为什么: 代码重用和节约内存！

何时: 多个子对象，都拥有相同的方法定义时。就应该找一个共同的父对象。将相同的方法定义，保存在父对象中一份。所有孩子自动继承共用。

如何: js中实现继承都是用原型对象实现的。

什么是原型对象: 集中保存同一类型下所有子对象共有成员的父对象



何时: 只要当前类型下的所有子对象都需要一批共有的成员时，就需要在原型对象中同一定义。

如何:

1. 创建: 买一赠一，不用创建

js中买妈妈赠个爹！

构造函数妈妈的prototype属性引用着自己的老公——原型对象爸爸。原型对象中的constructor属性引用着自己的老婆——构造函数妈妈

2. 何时继承: 不用自己设置！

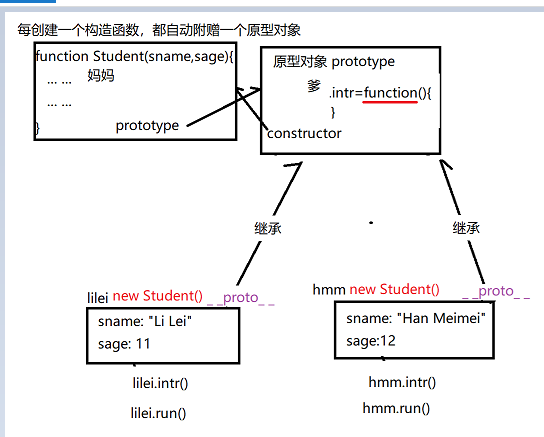
new的第2步: 自动设置当前对象继承构造函数的老公——原型对象爸爸

3. 我们做什么？决定把什么东西放到原型对象中，让所有子对象共用。

今后只要多个子对象共用的属性值和方法定义，都不要放在构造函数里。应该放在原型对象中

如何: 强行赋值:

构造函数.prototype.新成员=新值



单词列表:

1. constructor 构造器，构造函数

2. prototype 原型

作业:

1. function fun(){

for(var i=0,arr=[];i<3;i++){

arr[i]=function(){ console.log(i) }

}

return arr;

}

var funs=fun();

funs[0]();//?

funs[1]();//?

funs[2]();//?