小程序: WEB问题速查

1. 问题清单: 小程序->首页->JSCORE->day04

当老师在回答其它同学的问题时，你可以打开问题清单浏览曾经回答过的问题！

2. 高清无码小视频: 小程序->在线->JSCORE->day04

2部分: 1. 课上重点知识和案例的回放！2. 重点扩展笔试面试题讲解扩展视频

今天扩展视频:

小程序->在线->JSCORE->day04 所有"扩展笔试题"开头的视频

所有扩展视频绝\*\*\*不要求当天必须看完\*\*\*！而且和第二天课程也没有关系！

只要在\*\*\*两个月后\*\*\*鄙视面试前\*\*\*看完就行\*\*\*！

3. 单词列表: 小程序->单词->JSCORE->day04

第三阶段上课约定:

1. 上课千万不要跟着记笔记和写代码！

2. 拼写错误！自己找！一旦让我发现是拼写错误，自己还懒得找，我可以帮你找。但是，如果我找到，你需要手抄20遍，给我拍照！

3. 一个原理或知识点，只有自己可以画出图来，才算自己学会！

如何提问:

1. 上课时，案例中，笔记中那句话哪行代码看不懂，立刻微信私信问我！

2. 小程序中问题清单里哪个问题和回答看不懂！立刻微信私信问我！

3. 个人项目中哪个功能不会做！立刻微信私信问我！

4. 刷笔试题时，遇到不会的笔试题！立刻微信私信问我！

下载源代码: 右键->链接另存为！

正课:

\*\*\*\*\*面向对象\*\*\*\*\*

1. 什么是面向对象

2. 封装

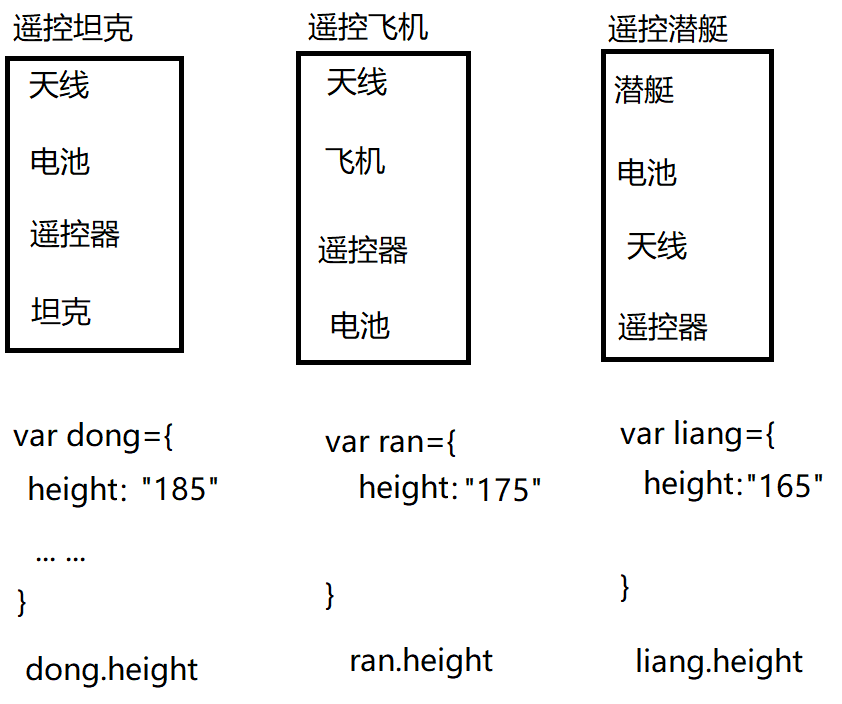
3. 继承

\*\*\*\*\*面向对象\*\*\*\*\*

一. 什么是面向对象:

1. 问题: 大量数据如果零散的管理，极其不方便查找和使用，极容易出错！

2. 解决: 今后大量的数据都是集中分类管理的——面向对象



3. 什么是: 程序中都是先将现实中一个事物的属性和功能，集中存储在一个对象中，再按需使用对象中的成员。

4. 为什么: 便于大量数据的管理和维护。

5. 何时: 今后所有软件都是用面向对象思想做出来！

6. 如何: 面向对象的三大特点/三个步骤: 封装，继承，多态

二. 封装:

1. 什么是: 创建一个对象结构，把现实中一个事物的属性和功能集中存储在对象结构中。

2. 为什么: 便于大量数据的管理和维护。

3. 何时: 今后只要使用面向对象的方式编程，第一步都要先创建对象，保存属性和功能。

4. 如何: 3种:

对应小程序视频:小程序->在线->JSCORE->day04 1. 用{}创建对象 this

(1). 用{}创建一个新对象:

a. var 对象名={

属性名: 值,

... : ... ,

方法名: function(){

... this.属性名 ...

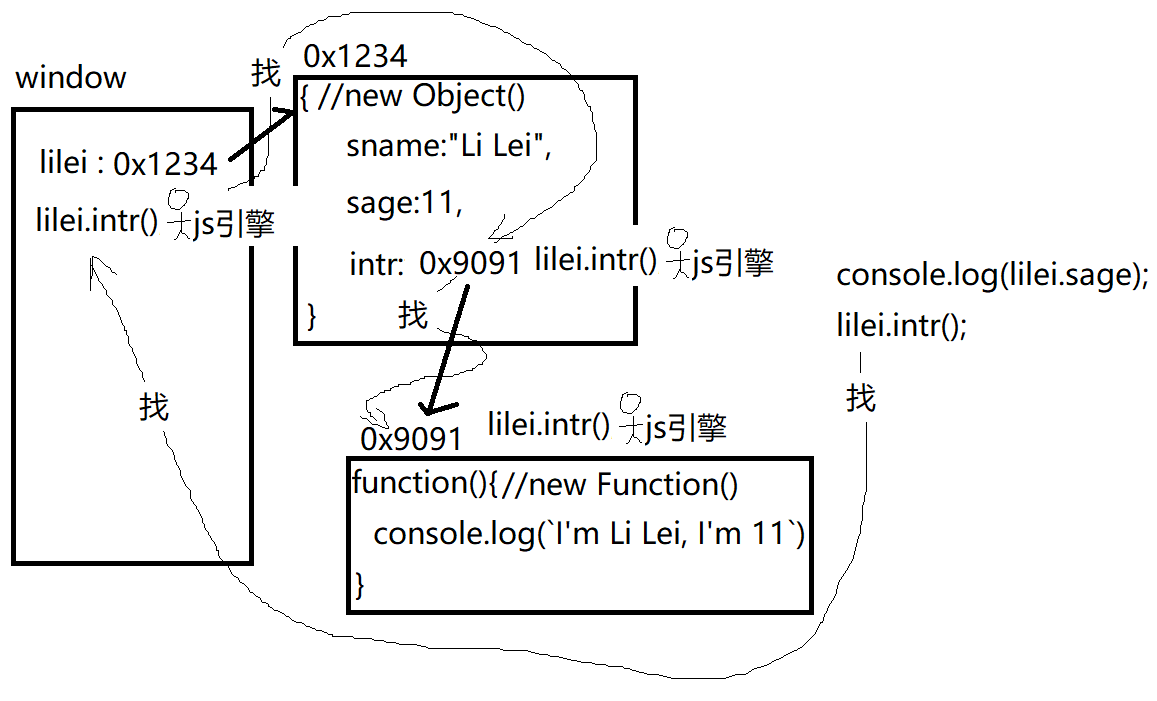
}

}

b. 访问对象的成员:

1). 访问对象中的属性值: 对象名.属性名

2). 调用对象中的方法: 对象名.方法名()



c. 问题: 对象中的方法，经常需要用到当前对象内自己的属性

d. 解决:

1). 错误: 在对象的方法中直接写属性名

i. 报错: 属性名 未定义

ii. 原因: 所有不带任何.前缀的变量，默认只能在作用域链中(函数作用域和全局作用域)查找。无权擅自进入对象内部，访问对象的属性

2). 不好的解决: 在属性名前写死"对象名."前缀  
 i. 问题: 紧耦合，外部的对象名一旦修改，必须手工同时修改内部写死的对象名。一旦忘记修改，程序立刻出错！

3). 好的解决: 今后，只要对象自己的方法，想用对象中其它的属性或方法时，都必须加"this."前缀

i. 什么是this:

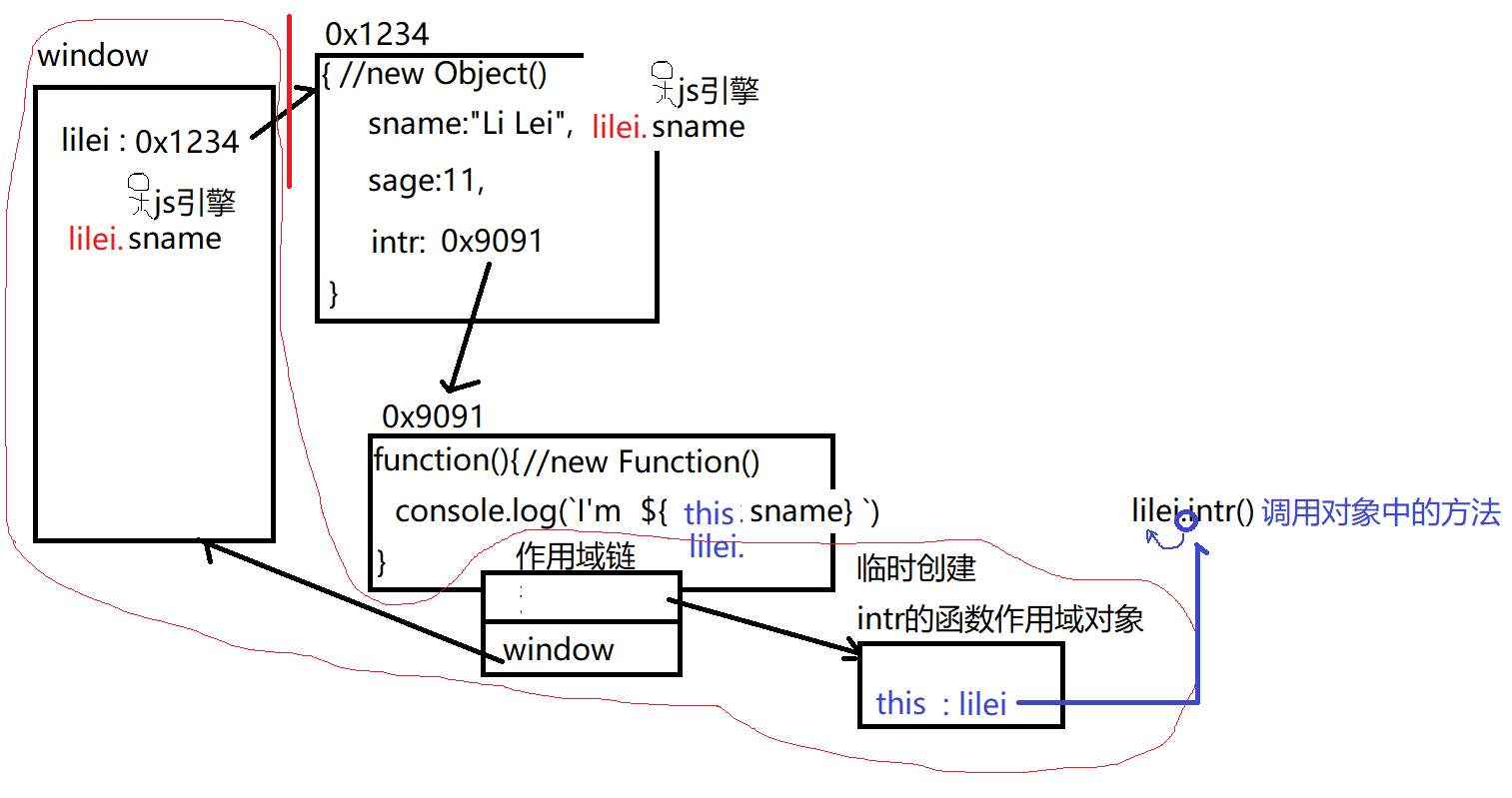
每个函数中自带的: 不用自己创建，直接使用

自动引用正在调用该函数的.前的对象的:

关键字

ii. 原理: this保存在每个函数的函数作用域对象中，可自动的引用到正在调用当前函数的.前的对象。

e. 示例: 创建一个对象保存一个学生的属性和会做的事



1\_obj.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //想创建一个对象，保存一个学生lilei的个人信息和他会做的事儿      //lilei: 姓名: Li Lei, 年龄:11,      //  会做自我介绍:      var lilei={//仅仅是new Object()的缩写                 //不是作用域的意思！        sname:"Li Lei",        sage:11,        intr:function(){          console.log(`I'm ${lilei.sname}, I'm ${lilei.sage}`)        }      }      //输出李磊的年龄，并请李磊做自我介绍      console.log(lilei.sage);      lilei.intr();      //过了一年，李磊长了一岁:      lilei.sage++;      //再输出李磊的年龄，并请李磊做自我介绍      console.log(lilei.sage);      lilei.intr();      console.log(lilei);    </script>  </body>  </html>  运行结果:  11  I'm Li Lei, I'm 11  12  I'm Li Lei, I'm 12  {sname: "Li Lei", sage: 12, intr: ƒ} |

(2). 用new来创建:

对应小程序视频: 小程序->在线->JSCORE->day04 2. 3. 对象底层是关联数组 ...

a. 2步:

1). 先创建一个空对象

var 对象名=new Object();

2). 再强行向空对象中添加新属性

对象名.属性名=属性值

对象名.方法名=function(){ ... this.属性名 ...}

b. 揭示: 其实js中所有对象底层都是关联数组

在js内存中几乎只有一种东西：关联数组

1). 存储结构: 对象和关联数组都是"名:值"对儿的集合

2). 访问成员的方式: 都有3种:

i. 标准: 数组/对象["成员名"]

自动翻译为↑

ii. 简写: 数组/对象.成员名

iii. 如果要访问的属性名不是固定，来自于变量，则必须用:

数组/对象[变量]

3). 访问不存在的成员: 都不会报错，而是返回undefined

固定套路: 如何判断一个对象是否包含某个属性

对象.属性!==undefined 就说明包含，否则说明不包含

4). 强行给一个不存在的成员赋值: 都不会报错，而是自动添加该属性

固定套路: 如果为一个对象添加一个新属性: 强行赋值

对象.新属性=新值

5). 都可用for in遍历每个成员:

for(var 属性名 in 数组或对象){ //复习第一阶段for in循环

//in会依次取出数组或对象中每个成员的:前的名字

//要想进一步访问每个成员的值，应该:

//数组或对象[属性名]

//千万不要加""，因为in前的属性名是一个变量，从来就没有把变量放在""里的情况！

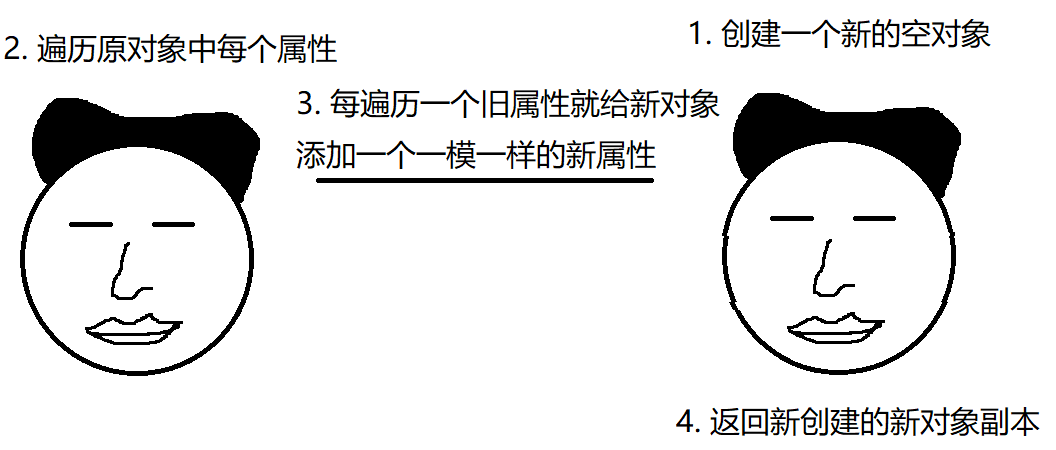
}

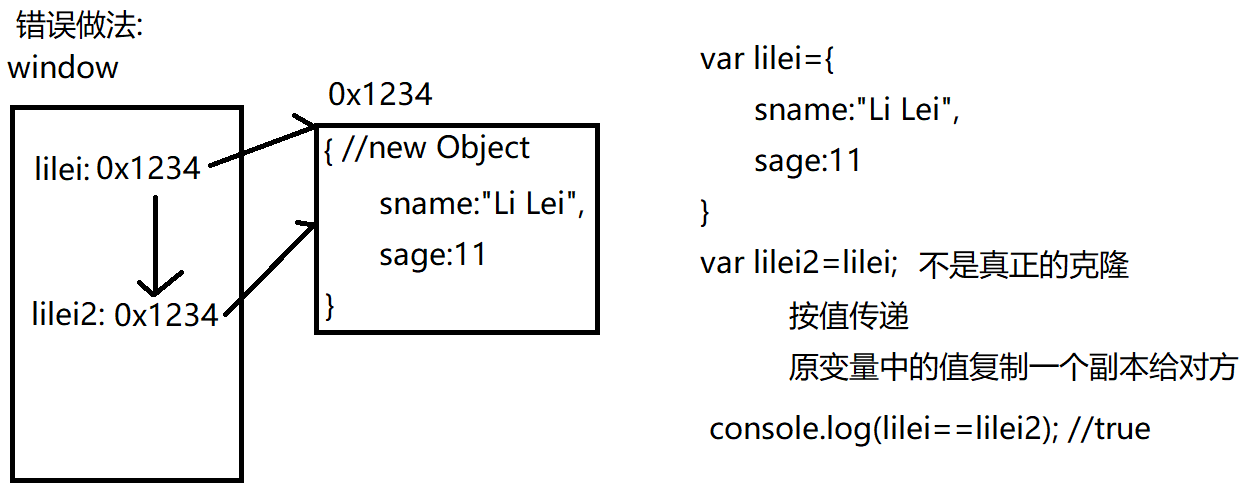
c. 示例: 分别定义对象和关联数组，并访问其中的成员:

2\_new\_Object.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var lilei=new Object();//对象      lilei.sname="Li Lei";      lilei["sage"]=11;      lilei.intr=function(){        console.log(`I'm ${this.sname}, I'm ${this.sage}`);      }      var hmm=[];//关联数组      hmm.sname="Han Meimei";      hmm["sage"]=12;      hmm["intr"]=function(){        console.log(`I'm ${this.sname}, I'm ${this.sage}`);      }      lilei["intr"]();      hmm.intr();        //同时访问两个学生不存在的班级属性      console.log(lilei.className, hmm.className);      //给两个对象都强行添加sex属性      lilei.sex=1;      hmm.sex=0;      console.log(lilei);      console.log(hmm);      //想遍历出李磊和hmm的所有成员      //复习第一阶段for in      for(var shuxing in lilei){        console.log(`${shuxing} : ${lilei[shuxing]}`);        //错误: console.log(`${shuxing}:${lilei.shuxing}`);        //错误: console.log(`${shuxing}:${lilei["shuxing"]}`);      }      for(var shuxing in hmm){        console.log(`${shuxing} : ${hmm[shuxing]}`);      }    </script>  </body>  </html>  运行结果：  I'm Li Lei, I'm 11  I'm Han Meimei, I'm 12  undefined undefined  {sname: "Li Lei", sage: 11, sex: 1, intr: ƒ}  [sname: "Han Meimei", sage: 12, sex: 0, intr: ƒ]  sname : Li Lei  sage : 11  intr : function(){  console.log(`I'm ${this.sname}, I'm ${this.sage}`);  }  sex : 1  sname : Han Meimei  sage : 12  intr : function(){  console.log(`I'm ${this.sname}, I'm ${this.sage}`);  }  sex : 0 |

d. 示例: 克隆一个对象:





3\_clone.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      var lilei={        sname:"Li Lei",        sage:11      }      //错误      //var lilei2=lilei;      //正确:      //因为要传入旧对象，所以要定义形参      function kelong(oldObj){        //1. 创建一个新对象等着        var newObj={};        //2. 遍历旧对象中每个属性        for(var shuxing in oldObj){          //3. 每遍历一个属性，就给新对象添加一个同名的新属性: 属性名和属性值都要相同          //3.1 先获得当前属性的旧属性值          var value=oldObj[shuxing];          //3.2 为新对象强行添加同名新属性          newObj[shuxing]=value;        }        //4. 返回创建好的新对象        return newObj;      }      var lilei2=kelong(lilei);      console.log(lilei);      console.log(lilei2);      console.log(lilei==lilei2);//false      //         两个对象比较时      //         不比内容，比地址值    </script>  </body>  </html>  运行结果:  {sname: "Li Lei", sage: 11}  {sname: "Li Lei", sage: 11}  false |

(3). 反复创建多个相同结构但属性值不同的新对象:

对应小程序视频: 小程序->在线->JSCORE->day04 5. 构造函数...

a. 问题: 将来很有可能反复创建多个相同结构但是属性值不同的对象。如果使用{}创建，代码会有很多重复！极其不便于维护！

b. 解决: 今后，只要反复创建多个相同结构的对象时，都用构造函数。

c. 什么是: 专门描述同一类型的多个对象统一结构的函数。

d. 如何:

1). 定义构造函数，规定这一类型所有对象统一结构

function 类型名(形参1, 形参2,...){

this.属性名1=形参1;

this.属性名2=形参2;

... = ...;

this.方法名=function(){

... this.属性 ...

}

}

2). 反复调用构造函数创建该类型的一个新的对象

var 新对象=new 类型名(属性值1, 属性值2,... ...)

e. 示例: 使用构造函数反复创建两个学生对象:

4\_constructor.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //1. 定义构造函数描述所有学生类型的对象的统一结构      //要求: 所有学生都要有姓名和年龄属性      //      所有学生都要会做自我介绍      function Student(sname,sage){        this.sname=sname;        this.sage=sage;        this.intr=function(){          console.log(`I'm ${this.sname},I'm ${this.sage}`)        }      }      //2. 反复创建2个学生对象      var lilei=new Student("Li Lei", 11);      var hmm=new Student("Han Meimei", 12);      lilei.intr();      hmm.intr();      console.log(lilei);      console.log(hmm)    </script>  </body>  </html>  运行结果:  I'm Li Lei,I'm 11  I'm Han Meimei,I'm 12  Student {sname: "Li Lei", sage: 11, intr: ƒ}  Student {sname: "Han Meimei", sage: 12, intr: ƒ} |

f. 构造函数的原理: new做了几件事: 4件事:

1). 创建一个新的空对象(买毛坯房)

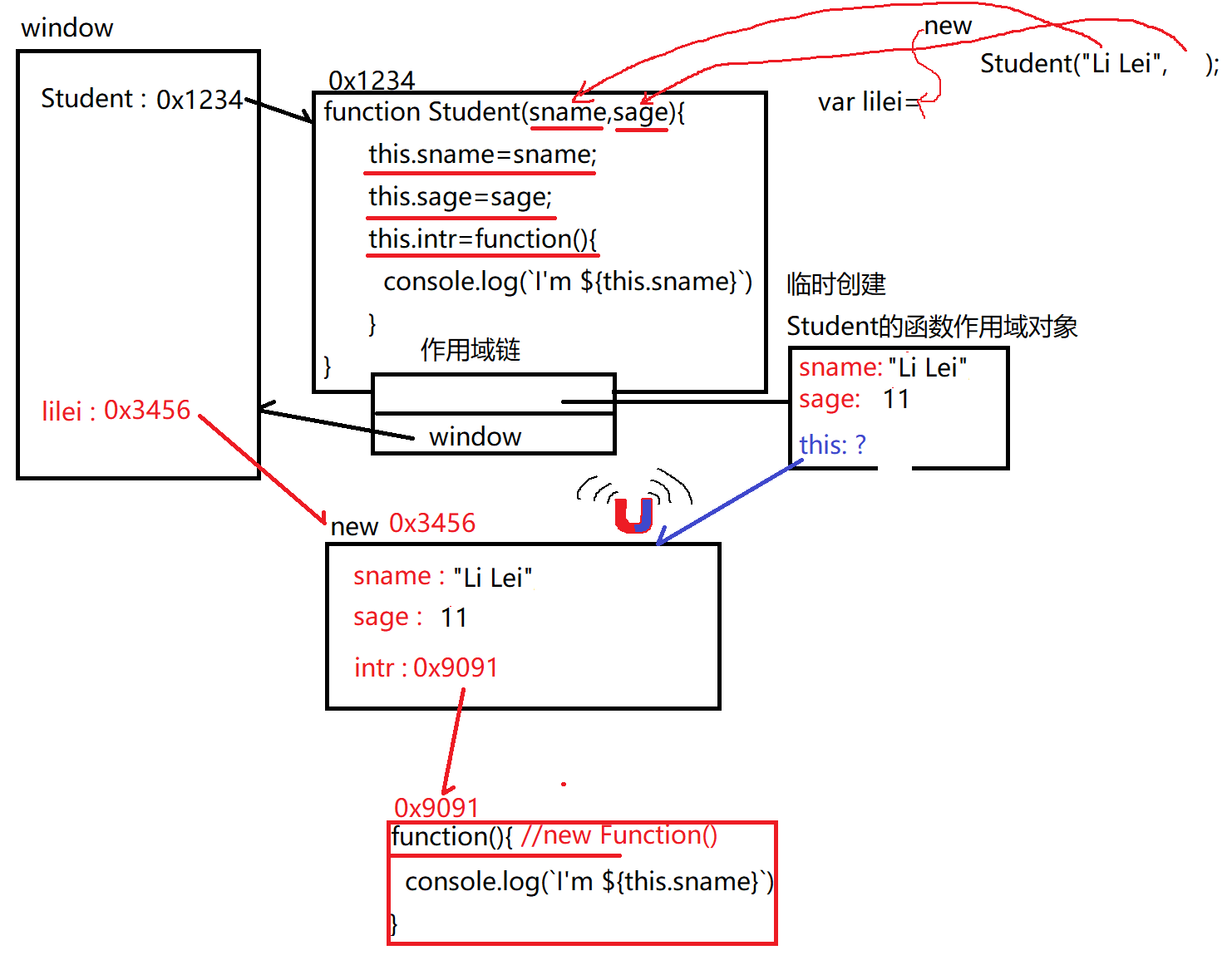
2). 自动设置新对象继承父对象。

3). 调用构造函数: (装修)

i. 强行将构造函数中的this改为当前新创建的对象

ii. 构造函数中所有"this.属性名=形参"，都变成"新对象.属性名=形参"。

iii. 因为新对象是空的，所以构造函数中，每一句"this.属性名=形参"，都变成了强行给新对象添加新属性。



4). 将新对象的地址，保存到=左边的变量中。(交钥匙)

|  |
| --- |
| 名词解释:  1. 变量: 内存中存储一个数据的存储空间，再起一个名字。  访问变量时，前边不需要加任何前缀，直接用变量名即可  2. 属性: 保存在对象结构中的特殊的变量。  访问属性时，要先找到对象，再用"对象.属性"方式来获取属性值。  3. 函数: 不属于任何对象的function。  调用函数，前边不用加任何前缀，只要"函数名()"  4. 方法: 保存在对象中的function。  调用方法，要先找到对象，再用"对象.方法名()"方式调用  5. 对象: |

三. 继承:

对应小程序视频: 小程序->在线->JSCORE->day04 6. 继承 原型对象 prototype

1. 问题: 如果将方法定义放在构造函数中，那么每调用一次new 构造函数()，都会为新对象创建一个完全相同的方法副本。——没必要！浪费内存！

2. 解决: 今后，只要同一类型的多个字对象，都要共用相同的方法定义，都要用继承来解决！

3. 什么是: 父对象中的成员，子对象无需重复创建，就可直接使用！

4. 如何: js中的继承都是通过"原型对象"来实现的。

(1). 什么是原型对象: 专门为同一类型的所有子对象，集中保存共有属性和方法的父对象。

(2). 如何创建: 不用创建。只要我们定义一个构造函数，js就会自动赠送我们一个空的原型对象。

(3). 如何找到这个原型对象:

构造函数.prototype->原型对象

因为原型对象是自动赠送的，所以不能改名

(4). 何时继承: 不用自己设置。new的第二步，自动设置当前新对象的"\_ \_proto\_ \_"属性引用构造函数的原型对象

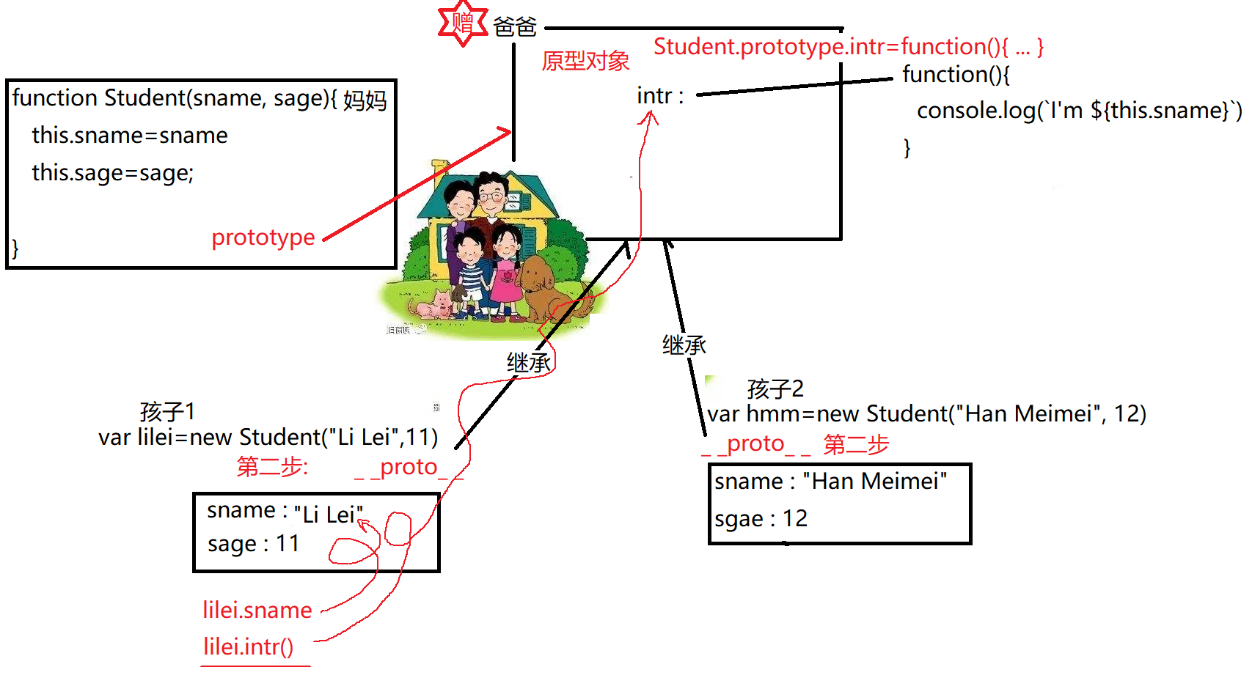
(5). 结果: 当用子对象试图访问一个成员时:

a. js引擎优先在当前子对象内查找自有属性。如果找到，就优先使用自有的！

b. 如果当前子对象中不包含要访问的成员，则js引擎会自动延\_ \_proto\_ \_去父对象(原型对象)中查找是否包含想要的成员

(6). 总结: 今后，凡是同一类型所有子对象共用的方法或属性值，都不要保存在构造函数中。一律放入构造函数的原型对象中！——强行赋值

构造函数.prototype.共有方法名=function(){ ... }

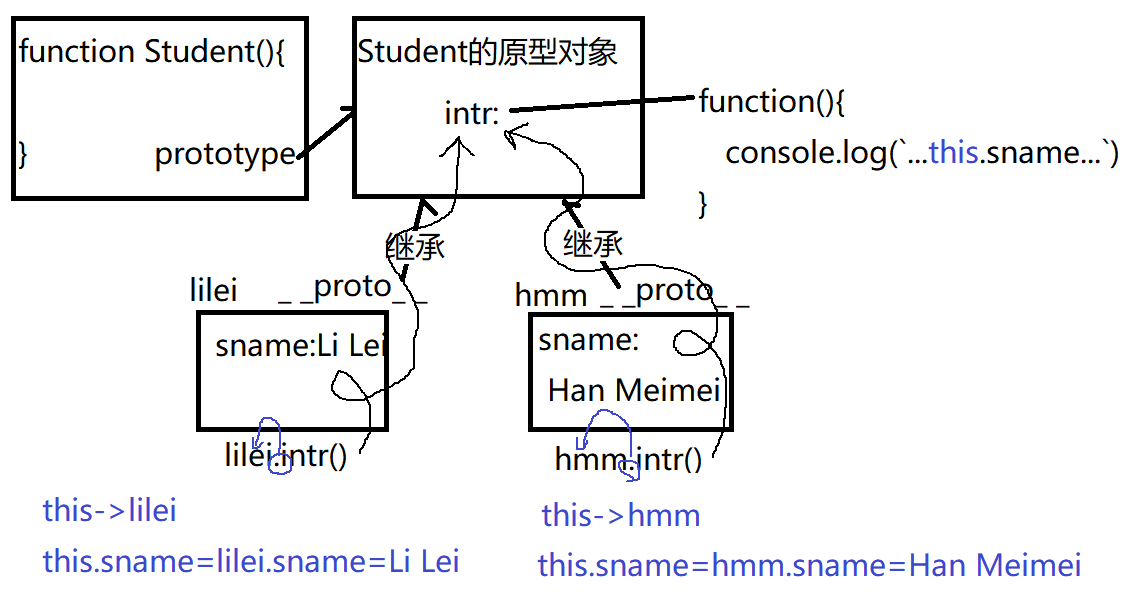


5. 示例: 将所有学生共用的方法，集中保存在原型对象中一份

5\_prototype.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //1. 定义构造函数描述所有学生类型的对象的统一结构      //要求: 所有学生都要有姓名和年龄属性      //      所有学生都要会做自我介绍      function Student(sname,sage){        this.sname=sname;        this.sage=sage;        //今后构造函数中就不应该包含方法定义！      }      //尝试输出构造函数Student赠送的原型对象      console.log(Student.prototype);      //所有方法定义一律放入原型对象中      Student.prototype.intr=function(){        console.log(`I'm ${this.sname},I'm ${this.sage}`)      }      //2. 反复创建2个学生对象      var lilei=new Student("Li Lei", 11);      var hmm=new Student("Han Meimei", 12);      lilei.intr();      hmm.intr();      console.log(lilei);      console.log(hmm);      //亲子鉴定:      //lilei的爹是不是Student的老公！      console.log(lilei.\_\_proto\_\_==Student.prototype);//true      //hmm的爹是不是Student的老公!      console.log(hmm.\_\_proto\_\_==Student.prototype);//true    </script>  </body>  </html>  运行结果:  {constructor: ƒ}  I'm Li Lei,I'm 11  I'm Han Meimei,I'm 12  Student {sname: "Li Lei", sage: 11}  Student {sname: "Han Meimei", sage: 12}  true  true |

6. 原型对象方法中的this:



7. 自有属性和共有属性:

对应小程序视频: 小程序->在线->JSCORE->day04 7. 自有属性 共有属性

(1). 自有属性: 只保存在当前子对象内，仅归当前子对象独有的属性

(2). 共有属性: 保存在原型对象中，归所有子对象共有的属性

(3). 获取属性值: 没有差别，都可用: 子对象.属性名

(4). 修改属性值:

a. 修改自有属性: 子对象.属性名=新值

b. 修改共有属性:

1). 坑: 不能用子对象随意修改原型对象中的共有属性

2). 如果强行更改，则js会自动在子对象中添加一个同名的自有属性

3). 结果：从此，这个子对象，在这个属性的使用上，与其他子对象，分道扬镳

4). 解决: 今后如果想修改共有属性值，只能通过构造函数来修改！

构造函数.prototype.共有属性名=新值

8. 内置类型的原型对象:

对应小程序视频: 小程序->在线->JSCORE->day04 8. 11种内置类型..

(1). 问题:

a. 为什么数组能new Array(); 日期也能new Date(); ...

b. 为什么所有数组都可以使用.sort(), .push(), .reverse()等

(2). 答: 其实，数组，日期，正则，函数都是浏览器已经创建好的"内置类型"。不用我们自己定义就可以直接使用。

(3). 从今以后，只要一提到一种"类型"，都有两部分组成:

a. 构造函数: 专门负责创建该类型的子对象

比如: Array其实就是一个内置的构造函数

b. 原型对象: 专门负责集中保存该类型所有子对象共有的方法。

比如: Array.prototype中集中保存了所有数组共用的函数。

(4). 问题: 如果我们经常需要的一个函数，在内置类型中没有提供！

(5). 解决: 今后，可以向内置的类型的原型对象中添加自定义的函数！——所有该类型的子对象都可使用！

(6). 如何:

内置类型.prototype.自定义共有方法=function(){

... 因为自定义共有方法，将来是以"子对象.共有方法()"方式调用的，所以this->将来.前的某个子对象。

}

(7). 示例: 为所有数组对象添加求和的函数:

7\_Array.prototye.html

|  |
| --- |
| <!DOCTYPE html>  <html lang="en">  <head>    <meta charset="UTF-8">    <meta name="viewport" content="width=device-width, initial-scale=1.0">    <title>Document</title>  </head>  <body>    <script>      //在数组的原型对象中添加一个sum()函数      Array.prototype.sum=function(){        console.log(`调用了一次我自己的sum()`)        //先定义一个变量,准备累加每个元素值        var result=0;        //遍历当前数组中每个元素值        //因为将来sum()使用"数组.sum()"方式调用的        //所以this可以自动获得将来.前的任意数组！        for(var i=0;i<this.length;i++){          result+=this[i]        }        //返回结果        return result;      }        var arr1=[1,2,3];      var arr2=[1,2,3,4,5];      console.log(arr1.sum());//6      console.log(arr2.sum());//15    </script>  </body>  </html>  运行结果：  调用了一次我自己的sum()  6  调用了一次我自己的sum()  15 |

总结:

2. RegExp对象:

(1). 创建正则表达式对象:

a. 如果正则是固定的:   
var reg=/正则/ig

b. 如果正则需要动态生成:   
var reg=new RegExp("正则",ig)

(2). 验证字符串格式:   
var bool=reg.test(str)   
reg必须同时前加^后加$

(3). 既查找每个关键词的内容又查找每个关键词的位置:

do{

var arr=reg.exec(str);

if(arr!=null){

获得本次找到的敏感词的内容(arr[0])和位置(arr.index)

}

}while(arr!=null);

3. 函数:

(1). 创建函数三种方式:

a. function 函数名(形参列表){ 函数体; return 返回值 } //会被声明提前，不好

b. var 函数名=function(形参列表){ 函数体; return 返回值 }//不会被声明提前，首选

c. var 函数名=new Function("形参1", "形参2", ... , "函数体; return 返回值")

|  |
| --- |
| 函数本质:  1). 函数也是一个对象，对象中保存着函数的函数体代码  2). 函数名只是一个普通的变量，函数名通过函数对象地址，引用着函数对象  3). function在底层等效于new Function()  function 函数名(){ ... }和var 函数名=function(){}在底层都会被翻译为  var 函数名=new Function(...)  只不过function 函数名(){}是先提前，再翻译  而var 函数名=function(){}是不提前，原地翻译 |

(2). 重载: 今后，一件事，根据传入不同的参数值，动态执行不同的逻辑时，都用重载

function 一个函数名(不写形参变量){

//arguments对象自动接住所有实参值

if(arguments.length==0){

执行一种逻辑

}else if(arguments.length==1){

执行另一种逻辑

}else{

执行其它逻辑

}

}

其中arguments是类数组对象: 和数组相比:

a. 相同点: 也有下标，length属性，也可for循环遍历

b. 不同点: 不是数组类型，无法使用数组家的函数

(3). 匿名函数:

a. 所有回调函数优先使用匿名函数——用完释放，节约内存

b. 所有js代码都应该保存在匿名函数自调中，禁止使用全局变量，避免全局污染！

(function(){

要执行的js代码

})()

结果: 匿名函数内的都是局部变量，不会产生全局变量。

局部变量随匿名函数一起释放。不会污染全局。

(4). 作用域和作用域链: (跟着视频亲自画图！！！)

a. 作用域:

1). 全局作用域:window，保存全局变量

优: 可重用，缺: 随处可用, 极易被污染

2). 函数作用域: 保存局部变量

局部变量包括2中: 函数中var出的变量和形参变量

优: 仅函数内可用，不会被污染，缺: 不可重用

3). 函数作用域对象原理:

i. 每个函数定义时都自带好友列表，好友列表里2个格子，一个是空，一个引用window

ii. 调用函数时临时创建函数作用域对象保存函数局部变量。并将函数作用域对象的地址保存到函数好友列表中离自己近的格子里。

iii. 函数执行过程中按就近原则先在自己的函数作用域对象中找局部变量使用。如果找不到，才被迫去全局window中找变量使用.

iv. 函数调用后，好友列表中离自己近的格子清空，导致函数作用域对象以及内部的局部变量被释放！——所以局部变量不可重用！

b. 作用域链: 保存一个函数所有可用的作用域对象的链式结构(好友列表)学名就叫作用域链。  
 1). 作用域链保存着一个函数可用的所有变量

2). 作用域链控制着变量的使用顺序。先局部后全局。

(5). 闭包: （跟着视频亲自画图!!!）  
a. 只要希望给一个函数保护一个可反复使用的专属变量，又防止这个变量被外界篡改时，都用闭包。

b. 闭包三步:

1). 用外层函数妈妈包裹要保护的变量和内层函数

2). 外层函数妈妈用return把内层函数孩子返回到外部

3). 外部想使用内层函数的人，必须调用外层函数，才能获得return出来的内层函数对象。并将内层函数保存在一个变量中反复使用。

c. 闭包形成的原因: 外层函数调用后，外层函数的作用域对象被内层函数引用着无法释放，形成了闭包对象

d. 闭包的缺点: 闭包比一般的函数占用多一块内存——外层函数的函数作用域对象。  
 所以，用完闭包后，应该尽快释放:   
 保存内层函数的变量=null

4. 面向对象: 封装 继承 多态

自己跟着视频，一步一步画图，自己标顺序，知识才能变成自己的

(1). 封装: 3种:

a. 用{}创建一个对象:

var 对象名={

属性名:属性值,

... : ... ,

方法名: function(){

... this.属性名 ...

}

}

b. 用new Object():

1). 2步:

i. var 对象名=new Object()

ii. 对象名.属性名=属性值;

对象名.方法名=function(){ ... }

2). 对象底层也是关联数组:

i. 都是名值对儿的集合

ii. 都可用[""]和.方式访问成员。

如果属性名来自于变量，就只能用[]，不要加""

iii. 访问不存在的属性，都不报错，返回undefined

判断是否包含某个属性:

对象.属性名!==undefined

iv. 强行给不存在的属性赋值，都不报错，而是自动添加该属性

给对象添加新属性，唯一办法，强行赋值:

对象名.新属性名=新值

v. 都可用for in遍历

c. 只要反复创建多个相同结构的对象都用构造函数:

1). 2步:

i. 定义构造函数:

function 类型名(形参1,形参2, ...){

this.属性名1=形参1;

this.属性名2=形参2;

//构造函数中不要再包含方法定义定义!

}

ii. 用new 调用构造函数:

var 对象名=new 类型名(属性值1, 属性值2,...)

2). new做了4件事:

i. 创建一个新的空对象

ii. 让新对象继承(\_ \_proto\_ \_)构造函数的原型对象

iii. 调用构造函数，传入实参，并自动替换构造函数中的this为new正在创建的新对象。构造函数中，通过强行赋值的方式为新对象添加规定的属性，并保存属性值。

iv. 返回新对象的地址，保存到=左边的变量中。

3). 优点: 重用对象结构代码

4). 缺点: 如果构造函数中包含方法定义，则每次创建新对象都会重复创建相同方法的副本。——浪费内存！

(2). 继承:

a. 今后，只要同一类型所有子对象共用的方法和属性值，都要集中保存在构造函数的原型对象中！

构造函数.prototype.属性名/共有方法名=属性值/function(){ ... }

b. 自有属性和共有属性:

1). 获取属性值:都可用"子对象.属性名"

2). 修改属性值:

i. 自有属性: 子对象.自有属性名=新值

ii. 共有属性: 构造函数.prototype.共有属性名=新值

c. 内置类型原型对象:

1). 11种内置类型/对象: String, Number, Boolean, Array, Date, RegExp, Math(对象), Error, Function, Object, global(对象)

2). 一种类型=构造函数+原型对象

i. 构造函数: 创建子对象

ii. 原型对象: 为所有子对象保存共有成员

3). 查看该类型共有哪些API: 类型名.prototype

4). 该类型缺少想用的方法: 类型名.prototype.共有新方法=function(){ ... }

总结: this 判断this时，一定不要看他定义在哪儿。必须看它在哪里以何种方式调用 4种:

1. obj.fun() this->点前的obj对象

2. fun() this->默认指window

3. new Fun() this->new正在创建的新对象

4. 类型名.prototype.共有方法=function(){ ... }

this->将来谁调用这个函数，就指谁

将来调用这个函数的.前的某个子对象

今日对应小程序视频:

小程序->在线->JSCORE->day03

8. 闭包 closure 压岁钱

小程序->在线->JSCORE->day04

1. 用{}创建对象 this

2. 3. 对象底层是关联数组 ...

5. 构造函数...

6. 继承 原型对象 prototype

7. 自有属性 共有属性

8. 11种内置类型..

作业:

1. 复习今日小程序问题清单: 小程序->首页->JSCORE->day03,04

2. 使用闭包实现区号机功能:

//通过闭包三步实现取号机函数

//....

//测试代码:

var getNum=外层函数()

getNum();//1

getNum();//2

//如果用i记录当前号码，即使写i=0

i=0;

getNum();//3

getNum();//4

答案: 小程序->首页 搜"取号"

6. 看小程序视频预习:

小程序->在线->JSCORE->day04